Scalability and Performance of Random Forest based Learning-to-Rank for Information Retrieval

Muhammad Ibrahim
B.Sc. (Hons.) and M.Sc.

A thesis submitted for the degree of Doctor of Philosophy at Monash University, Australia
Faculty of Information Technology

February 2016
©Muhammad Ibrahim (2016). Except as provided in the Copyright Act 1968, this thesis may not be reproduced in any form without the written permission of the author.
Supervisors

Main Supervisor: Dr. Mark Carman
Monash University, Australia.

Associate Supervisor: Professor Dr. Manzur Murshed
Federation University, Australia.
Declaration of Authorship

This thesis contains no material which has been accepted for the award of any other degree or diploma at any university or equivalent institution and that, to the best of my knowledge and belief, this thesis contains no material previously published or written by another person, except where due reference is made in the text of the thesis.

Muhammad Ibrahim
Abstract

For a query submitted by a user, the goal of an information retrieval system is to return a list of documents which are highly relevant with respect to that query. Traditionally different scoring methods, ranging from simple heuristic models to probabilistic models, have been used for this task. Recently researchers have started to use supervised machine learning techniques for solving this problem which is then called the learning-to-rank (LtR) problem. Many supervised learning methods have been tested so far with empirical success over conventional methods.

The random forest is a relatively simple but effective and efficient learning algorithm which aggregates the predictions of a large number of independent and variant base learners, namely decision trees. Its major benefits over other state-of-the-art methods include inherent parallelizability, ease of tuning and competitive performance. These benefits attract researchers across various disciplines where a random forest is a very popular choice. However, for LtR task, the random forest has not been thoroughly investigated.

In this research, we investigate the random forest based LtR algorithms. We aim at improving the efficiency, effectiveness, and understanding of these algorithms. With respect to the first goal, we employ undersampling techniques and leverage the inherent structure of a random forest to achieve better scalability, especially for highly imbalanced datasets. We also reduce the correlation among the trees to reduce learning time and to improve performance. With respect to the second goal, we investigate various objective functions ranging from completely randomized splitting criterion to so-called listwise splitting. We also conduct a thorough study on random forest based pointwise algorithms. With respect to the third goal, we develop methods for estimating the bias and variance of rank-learning algorithms, and examine their empirical behavior against parameters of the learning algorithm.
Acknowledgements

All praise is due to the Almighty Allah Who has kindly allowed and enabled me to carry out this research.

My supervisor, Dr. Mark Carman has been a constant and reliable source of help, encouragement, and appreciation with his astute knowledge, scholarship and guidance. He has been with me in every difficulty – be it of academic matters or otherwise.

I would like to thank my associate supervisor Professor Manzur Murshed for his invaluable suggestions, which emanate from his vast experience.

My parents inspired me throughout the entire duration of this research. Whenever I felt down, they lifted me up.

Perhaps the most sacrifice has been done by my wife. How many weekends she has forsaken for this research!

The anonymous paper reviewers rightfully deserve a special thanks for their useful comments on different papers that emerged from this research.

I would like to express my gratitude towards the MCC-cluster team, especially to Dr. Shahaan Ayyub at Monash University for their support and help in running the experiments.

I would like to thank my colleagues here at Monash University with whom I oftentimes discussed my ideas, in particular Dr. Ahsan Jaleel. I would also like to thank the stuff members at Monash University for giving me a very good company and support.
Contents

Declaration of Authorship vii

Abstract ix

Acknowledgements xi

List of Figures xix

List of Tables xxiii

Abbreviations xxvii

Symbols xxix

List of Publications xxxi

1 Introduction 1

1.1 Motivation . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 2
1.2 Aims . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 4
1.3 Research Questions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
1.4 Major Contributions . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 5
1.5 Organization . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 8

2 Background, Literature Review and Experimental Setup 9

2.1 The Road to Learning-to-Rank . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.1.1 Traditional Approaches . . . . . . . . . . . . . . . . . . . . . . . . . . . 9

2.1.1.1 Vector Space Model (VSM) . . . . . . . . . . . . . . . . . . . . . 10

2.1.1.2 Probabilistic Model I: Binary Independence Model (BIM) 10

2.1.1.3 Probabilistic Model II: Language Model (LM) . . . . . . 11

2.1.2 Learning-to-Rank Approach . . . . . . . . . . . . . . . . . . . . . . . . . 11

2.2 Problem Formulation of Learning-to-Rank . . . . . . . . . . . . . . . . . . . 13

2.3 Evaluation Metrics Used in Information Retrieval . . . . . . . . . . . . . . . . . . 15

2.3.1 Discounted Cumulative Gain (DCG) and Normalized DCG (NDCG) 15

2.3.2 Mean Average Precision (MAP) . . . . . . . . . . . . . . . . . . . . . . 16

2.3.3 Expected Relevance Rank (ERR) . . . . . . . . . . . . . . . . . . . . . . 16

2.4 Existing LtR Approaches . . . . . . . . . . . . . . . . . . . . . . . . . . . . 17

2.4.1 Important Aspects of an LtR Algorithm . . . . . . . . . . . . . . . . . . 17
<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.4.2</td>
<td>Discussion</td>
<td>21</td>
</tr>
<tr>
<td>2.4.3</td>
<td>A Taxonomy of LtR Algorithms</td>
<td>21</td>
</tr>
<tr>
<td>2.5</td>
<td>Tasks Related to Learning-to-Rank</td>
<td>22</td>
</tr>
<tr>
<td>2.5.1</td>
<td>Ordinal Regression</td>
<td>22</td>
</tr>
<tr>
<td>2.5.2</td>
<td>Collaborative Filtering</td>
<td>23</td>
</tr>
<tr>
<td>2.5.3</td>
<td>Rank Aggregation</td>
<td>23</td>
</tr>
<tr>
<td>2.6</td>
<td>Theory of Learning-to-Rank</td>
<td>23</td>
</tr>
<tr>
<td>2.7</td>
<td>Why Random Forest</td>
<td>25</td>
</tr>
<tr>
<td>2.7.1</td>
<td>Advantages of Random Forests</td>
<td>25</td>
</tr>
<tr>
<td>2.7.2</td>
<td>Need for Further Research in Random Forest Based LtR</td>
<td>28</td>
</tr>
<tr>
<td>2.8</td>
<td>Scope of the Thesis</td>
<td>28</td>
</tr>
<tr>
<td>2.9</td>
<td>Experimental Protocols</td>
<td>29</td>
</tr>
<tr>
<td>2.9.1</td>
<td>Datasets</td>
<td>29</td>
</tr>
<tr>
<td>2.9.2</td>
<td>Notations</td>
<td>29</td>
</tr>
<tr>
<td>2.9.3</td>
<td>Toolkits</td>
<td>30</td>
</tr>
<tr>
<td>2.9.4</td>
<td>Significance Test</td>
<td>30</td>
</tr>
<tr>
<td>2.9.5</td>
<td>Evaluation Metrics</td>
<td>31</td>
</tr>
<tr>
<td>2.9.6</td>
<td>Default Parameter Settings</td>
<td>31</td>
</tr>
<tr>
<td>3</td>
<td>Handling the Imbalanced Training Data</td>
<td>33</td>
</tr>
<tr>
<td>3.1</td>
<td>Motivation</td>
<td>34</td>
</tr>
<tr>
<td>3.2</td>
<td>Contributions</td>
<td>36</td>
</tr>
<tr>
<td>3.3</td>
<td>Related Work</td>
<td>37</td>
</tr>
<tr>
<td>3.4</td>
<td>Approach</td>
<td>38</td>
</tr>
<tr>
<td>3.5</td>
<td>Model</td>
<td>39</td>
</tr>
<tr>
<td>3.6</td>
<td>Experiments</td>
<td>41</td>
</tr>
<tr>
<td>3.6.1</td>
<td>Setup</td>
<td>41</td>
</tr>
<tr>
<td>3.6.2</td>
<td>Using RF-point Algorithm</td>
<td>41</td>
</tr>
<tr>
<td>3.6.2.1</td>
<td>Random Undersampling</td>
<td>42</td>
</tr>
<tr>
<td>3.6.2.2</td>
<td>Deterministic Undersampling</td>
<td>43</td>
</tr>
<tr>
<td>3.6.2.3</td>
<td>Comparison between Two Approaches</td>
<td>45</td>
</tr>
<tr>
<td>3.6.3</td>
<td>Using RankSVM Algorithm</td>
<td>46</td>
</tr>
<tr>
<td>3.7</td>
<td>A More Scalable Undersampling with Random Forest</td>
<td>47</td>
</tr>
<tr>
<td>3.7.1</td>
<td>Approach</td>
<td>47</td>
</tr>
<tr>
<td>3.7.2</td>
<td>Result Analysis</td>
<td>48</td>
</tr>
<tr>
<td>3.8</td>
<td>Discussion</td>
<td>49</td>
</tr>
<tr>
<td>3.9</td>
<td>Conclusion</td>
<td>50</td>
</tr>
<tr>
<td>4</td>
<td>Reducing Correlation Between the Trees</td>
<td>53</td>
</tr>
<tr>
<td>4.1</td>
<td>Motivation</td>
<td>54</td>
</tr>
<tr>
<td>4.2</td>
<td>Contributions</td>
<td>55</td>
</tr>
<tr>
<td>4.3</td>
<td>Related Work</td>
<td>55</td>
</tr>
<tr>
<td>4.4</td>
<td>Approach</td>
<td>56</td>
</tr>
<tr>
<td>4.5</td>
<td>Result Analysis</td>
<td>58</td>
</tr>
<tr>
<td>4.5.1</td>
<td>Relatively Small Datasets</td>
<td>59</td>
</tr>
<tr>
<td>4.5.2</td>
<td>Big Datasets</td>
<td>62</td>
</tr>
<tr>
<td>4.6</td>
<td>Further Analysis of Sub-sampling Method</td>
<td>64</td>
</tr>
</tbody>
</table>
4.6.1 Effect of Ensemble Size ........................................ 64
4.6.2 Strength of Individual Trees .................................. 65
4.6.3 Further Reduction in Correlation by Disjoint Sub-sampling . 66
4.7 Theoretical Explanation of Sub-sampling Method ................. 67
  4.7.1 Estimating Correlation, Variance and Strength ............... 67
  4.7.2 Result Analysis .................................................. 69
4.8 Discussion ............................................................ 70
4.9 Conclusion ............................................................. 71

5 Pointwise and Listwise Objective Functions ...................... 73
  5.1 Motivation .......................................................... 74
  5.2 Contributions ....................................................... 74
  5.3 Related Work ........................................................ 75
    5.3.1 Learning to Rank Using Random Forest ..................... 75
    5.3.2 Direct Optimisation of Ranking Measures ................. 76
    5.3.3 Motivation for Our Approach ................................ 77
  5.4 Approach ............................................................ 78
    5.4.1 Common Properties for All RF-based Algorithms .......... 79
      5.4.1.1 Document Score During Test Phase .................... 79
      5.4.1.2 Sub-sampling Per Tree .................................. 79
      5.4.1.3 Termination criteria .................................... 80
    5.4.2 Random Forest Based Pointwise Algorithm ................. 81
    5.4.3 Random Forest Based Listwise Algorithm .................... 81
      5.4.3.1 Document Scores During Training ...................... 82
      5.4.3.2 Computing the Objective Function ..................... 83
      5.4.3.3 Ordering of Leaf Expansion ............................ 86
    5.4.4 A Hybrid Algorithm to Scale-up the Listwise Algorithm ... 87
      5.4.4.1 Time Complexity ......................................... 87
      5.4.4.2 Heuristics to Reduce Time Complexity of RF-list ....... 87
      5.4.4.3 A Combination of Listwise and Pointwise Splitting .. 88
    5.4.5 Random Forest Based Pairwise Algorithm .................... 89
  5.5 Result Analysis ..................................................... 90
    5.5.1 RF-list with Standard NDCG ................................ 90
    5.5.2 RF-list with Modified NDCG Discount ....................... 93
    5.5.3 RF-hybrid ...................................................... 93
    5.5.4 RF-pair ........................................................ 98
  5.6 Further Analysis of Splitting Criterion ......................... 99
    5.6.1 Importance of Splitting Criteria ............................ 99
      5.6.1.1 LtR With Completely Randomized Trees ............... 99
      5.6.1.2 Results and Discussion ................................ 100
    5.6.2 Overfitting .................................................... 101
    5.6.3 Strength of Individual Trees ................................. 102
    5.6.4 Remarks ........................................................ 106
  5.7 Discussion .......................................................... 107
  5.8 Conclusion .......................................................... 108

6 Detailed Comparison with State-of-the-Art Rank-Learners ....... 111
### 6.1 Determining the Right Settings of RF-based Algorithms

**6.1.1 Motivation**  
111

**6.1.2 Contributions**  
112

**6.1.3 Classification or Regression?**  
112

#### 6.1.3.1 Motivation and Methodology
112

#### 6.1.3.2 Result Analysis
113

**6.1.4 Ground Truth Labels: Absolute Relevance or Scaled?**  
114

#### 6.1.4.1 Motivation and Methodology
114

#### 6.1.4.2 Result Analysis
115

**6.1.5 Controlling Individual Tree Complexity**  
116

#### 6.1.5.1 Motivation and Methodology
116

#### 6.1.5.2 Result Analysis
116

**6.1.6 Weighting the Trees**  
118

#### 6.1.6.1 Motivation and Methodology
118

#### 6.1.6.2 Result Analysis
119

**6.1.7 Discussion**  
119

### 6.2 Comparison with Other Algorithms

**6.2.1 Baselines**  
120

**6.2.2 Smaller Datasets**  
122

**6.2.3 Big Datasets**  
125

#### 6.2.3.1 Absolute Performance
126

#### 6.2.3.2 Learning Curve
129

**6.2.4 Discussion**  
130

### 6.3 Conclusion
132

### 7 Understanding Bias and Variance of Rank-Learners

**7.1 Motivation**  
133

**7.2 Contributions**  
134

**7.3 Background**  
135

**7.4 Understanding Random Forests from a Bias-Variance Perspective**  
136

#### 7.4.1 Bagging for Variance Reduction
137

#### 7.4.2 Adding Randomness to Make a Random Forest
138

**7.5 Related Work**  
140

**7.6 Methodology**  
141

#### 7.6.1 Two Types of Analyses
141

#### 7.6.2 Model
142

**7.7 Pointwise Analysis**  
142

#### 7.7.1 Estimating Bias-Variance Using Multiple Bootstrap Samples
143

#### 7.7.2 Estimating Bias-Variance Using Two-Fold Cross-Validation
144

**7.8 Listwise Analysis**  
145

#### 7.8.1 Preliminaries
146

#### 7.8.2 Bias-Variance Using Permutations of Documents
146

#### 7.8.3 Bias-Variance Using the Dissimilarity between Score Lists
149

#### 7.8.4 Bias-Variance Using an IR Metric-based Loss
150

#### 7.8.5 Formulae for Estimating Listwise Bias and Variance
154

#### 7.8.5.1 Estimating Quantities Using Bootstrap Samples
154
List of Figures

1.1 An information retrieval system interacting with user. ..................... 2
1.2 Training set of LtR task .................................................. 13
1.2 Different components of an LtR system. ................................ 15
1.3 NDCG is either flat or discontinuous with respect to model parameters. 20
1.4 A taxonomy of LtR algorithms. ........................................... 22

2.1 An LtR-based IR system. .................................................... 35
2.2 Relevance label distribution of different datasets. TD/(HP/NP)-04, MQ-
07/(08) and MSLR are abbreviations for TD/(HP/NP)2004, MQ2007/(2008)
and MSLR-WEB10K respectively. ......................................... 35
2.3 Undersampling in our approach. ......................................... 38
2.4 Results of random undersampling approach with RF-point algorithm. .. 42
2.5 Training time of RF-point against the percentage of training set used to
learn the ensemble. .......................................................... 44
2.6 Results of deterministic undersampling approach with RF-point algo-
thesis. ............................................................................. 45
2.7 Performance comparison of RF-point between the random undersam-
pling and deterministic undersampling on different datasets in terms of
NDCG@10. ......................................................................... 46
2.8 With RankSVM algorithm, results for random undersampling approach
across six datasets. .......................................................... 46
2.9 Tree-level undersampling with RF-point. .................................. 47
2.10 Comparison between the ensemble-level and the tree level undersampling
with a random forest. The bars indicate the amount of training data (per
tree) required to achieve 98% of NDCG@10 of the baseline (i.e., RF learnt
with all training data); cf. Table 3.2. ..................................... 49

3.1 Sub-sampling at tree level of a random forest. ............... 57
3.2 Performance in terms of NDCG@10 (left column) and corresponding stan-
dard deviation (right column) for Ohsumed, MQ2008 and MQ2007 datasets
(from top to bottom) using validation sets. In the left plots, the points
along a vertical line represent NDCG@10 values of 10 independent runs
of a configuration (using the % of training queries per tree indicated by
the x-axis). A smooth-spline is fitted (with smoothing parameter 0.01).
The bar plots in the right indicate the standard deviations of the corre-
sponding NDCG@10 values across the 10 runs. ......................... 59
3.3 Performance in terms of NDCG@10 (left column) and standard deviation
(right column) of TD2004, HP2004 and NP2004 (from top to bottom)
using validation sets. For description, see caption of Figure 4.2. ....... 60
4.4 Training time to learn a tree against % of queries (used to learn a tree) with validation sets. .................................................. 61
4.5 Comparison of performance (in terms of NDCG@10, cf. Table 4.1) between sub-sampling approach and baseline. Numerical values of increase in performance are (from left to right): 2.9%, 1.4%, 1.9%, 5.5%, 1.5%, and 2.5%. ................................................................. 62
4.6 Performance (NDCG@10 and MAP) of MSLR-WEB10K (Fold 1) and Yahoo datasets as the % of queries per tree increases. ............ 63
4.7 Training time (per tree) on Yahoo and MSLR-WEB10K (Fold 1) datasets as sub-sample size per tree increases. ...................... 64
4.8 Effect of ensemble size on performance of performance on RF-point-S%. (The % denotes the percentage of training queries used to build each tree.) 65
4.9 Varying $K$ (i.e., number of features randomly selected at each node) with (RF-point-S%). ......................................................... 66
4.10 Correlation between trees, single tree variance, ensemble variance and performance (in terms of NDCG@10) as sub-sample size per tree is varied on MSLR-WEB10K (Fold1) (top 2 rows) and Yahoo (bottom 2 rows) datasets. ............................................................. 69
5.1 Relationship between (existing) RF-point and (proposed) RF-list ...... 82
5.2 Local decision in a pointwise tree vs global decision in a listwise tree ... 82
5.3 RF-hybrid-L2: Splits up to level 2 (i.e., 3 splits) are listwise, the rest are pointwise. ............................................................. 88
5.4 RF-hybrid with different levels of tree up to which listwise splitting has been performed versus performance (left figures) and computational time (right figures). .................................................... 95
5.5 Effect of sub-sampling method on RF-hybrid-L6 in terms of performance and (normalized) training time. Left figure: RF-hybrid-L4 with 30% and 63% sub-sample (per tree). Right figure: RF-hybrid-L6 with 30% and 63% sub-sample (per tree). .............................................. 96
5.6 With varying $M$, performance of RF-point and RF-hybrid-L6 on Yahoo dataset. The rightmost point corresponds to the original training set (i.e., $M = 519$) ................................................................. 97
5.7 Performance (in terms of NDCG@10) of RF-point-S5 and RF-point-S5 as $n_{\text{min}}$ (i.e., minimum # instances required to consider splitting a node) varies. ................................................................. 102
5.8 Plots of progressive strength of individual trees (in terms of training and test Expected NDCGs) against the order of nodes expanded. The blue and red curves refer to RF-point and RF-list (both with breadth-first enumeration) respectively. The solid and dashed lines are for train and test curves respectively. .................................................. 103
5.9 Performance difference between RF-point-S5 and RF-list-S5 as $n_{\text{min}}$ varies.105
6.1 Performance variation as $K$ (i.e., number of candidate features at each node) is increased. ...................................................... 117
6.2 Performance of different algorithms. .......................................... 127
6.3 Effect of training sample (in log scale) on performance of RF-point, RF-list and LambdaMart. Recall that for these datasets RF-list is computationally not feasible to learn from the entire training data (which is the last point of a curve), so we show performance of RF-hybrid-L6 for that point. .................................................. 129

7.1 Performance (NDCG@10) of MSLR-WEB10K (Fold 1) and Yahoo datasets as the % of queries per tree varies. ................................. 134

7.2 Different types of errors for a generic loss function. ......................... 146

7.3 Evaluating a candidate ranked list: comparison between the conventional IR metrics (left figure) and the rank-aggregation metrics (right figure). 148

7.4 Bias and variance estimation for ranking error. ............................... 153

7.5 Calculation of systematic ranking error (SRE) (left figure) and variability in predictions in terms of ranking error (VRE) (right figure) for a query (using different samples). ................................. 154

7.6 Pointwise analysis with both the methods of bootstrapping and repeated twofold CV: ensemble bias and ensemble variance estimates on MSLR-WEB10K (top row) and Yahoo (bottom row) datasets with RF-point (regression setting). .......................................................... 157

7.7 Listwise analysis with both the methods of bootstrapping and repeated twofold CV: ensemble bias and ensemble variance estimates on MSLR-WEB10K (Fold1) (top row) and Yahoo (bottom row) datasets with RF-point (regression setting). .................................................. 159

7.8 Ranking error with the methods of bootstrapping (top row) and repeated twofold CV (bottom row). ................................. 160
List of Tables

1.1 Research sub-questions, objectives and the chapter numbers where they are addressed. .......................................................... 5

2.1 Some existing LtR algorithms grouped by loss functions and learning models. ................................................................. 21

2.2 Statistics of the datasets (sorted by # queries). In the last row, 973/15 for TD2004 means that there are 973 and 15 documents of label 0 and 1 respectively. .......................................................... 30

2.3 Datasets sorted by different properties. (MSLR-W stands for MSLR-WEB10K.) ................................................................. 30

3.1 Random undersampling with RF-point: % of training data required (per ensemble) to achieve a given percentage of performance of baseline NDCG@10 (i.e., with 100% training data) along with training time improvement. .......................................................... 42

3.2 Comparison between standard random undersampling (i.e., ensemble level) and tree level random undersampling with RF-point: % of training data required (per tree) to achieve a given percent of baseline (i.e., 63% training set) performance (NDCG@10) along with training time improvement. The best value is in bold and italic font. .......................................................... 48

4.1 On test sets, comparison between our sub-sampling method and the baseline in terms of training time and performance (using the best values of \( p \) found from validation sets). \( p \) is the number of queries used to learn a tree. Performance (NDCG@10 and MAP) is computed as the average of 10 independent runs. Standard deviation is also reported. .......................................................... 62

5.1 LtR algorithms using different approaches and frameworks. ................................. 78

5.2 An example of Expected NDCG computation. .................................................. 84

5.3 Performance comparison among pointwise and listwise approaches. RF-p and RF-l stands for RF-point and RF-list respectively. The bold and italic and bold figures denote that the best performance is significant with p-value less than 0.01 and 0.05 respectively. .......................................................... 91

5.4 Results of RF-point-S5 and RF-list-S5 on big datasets. The bold and italic and bold figures denote that the best performance is significant with p-values less than 0.01 and 0.05 respectively. .......................................................... 92

5.5 Results of RF-list-S5 with different discount functions. Pairwise significance test is performed treating the standard NDCG (with discount 1.0) as baseline. The bold and italic and bold figures denote that the performance difference is significant with p-value less than 0.01 and 0.05 respectively. † means significantly poorer than baseline. .......................................................... 94
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.6</td>
<td>Results of RF-point and RF-list-S5/RF-hybrid-L6. Significance test is conducted between (1) RF-point-S5 and RF-list-S5, and (2) RF-point and RF-hybrid-L6. The <strong>bold and italic</strong> and <strong>bold</strong> figures denote that the performance difference is significant with $p$-value less than 0.01 and 0.05 respectively. † means significantly poorer.</td>
</tr>
<tr>
<td>5.7</td>
<td>Training time improvement for RF-hybrid with sub-sampling method.</td>
</tr>
<tr>
<td>5.8</td>
<td>Comparison between RF-point, RF-pair, and RF-list/hybrid.</td>
</tr>
<tr>
<td>5.9</td>
<td>RF-point Vs RF-rand on small to moderate datasets.</td>
</tr>
<tr>
<td>5.10</td>
<td>Big datasets with RF-point, RF-rand and a complete random ranking.</td>
</tr>
<tr>
<td>5.11</td>
<td>Tree size, leaf size and leaf score of the trees of RF-point-S5 (p-S5) and RF-list-S5 (l-S5), averaged over the trees of the ensemble. The performance of these settings were given in Table 5.4.</td>
</tr>
<tr>
<td>5.12</td>
<td>Performance comparison between RF-point-S5 (p-S5) and RF-list-S5 (l-S5) with different branching probabilities on Yahoo dataset. Pairwise significance test is conducted by treating the setting with branch probability = 0.0 as the baseline (within each of the two groups).</td>
</tr>
<tr>
<td>6.1</td>
<td>RF-point with classification (RF-p-cla) versus regression (RF-p-regr) settings. For two larger datasets (MSLR-WEB10K and Yahoo), the <strong>bold and italic</strong> and <strong>bold</strong> figures denote that the best performance is significant with $p$-value less than 0.01 and 0.05 respectively. For smaller datasets, an average over 5 independent runs is reported (and each run is the result of 5-fold cross-validation), and the winning value is given in <em>italic</em> font.</td>
</tr>
<tr>
<td>6.2</td>
<td>For regression setting, RF-point with standard relevance (RF-p-SR) versus mapped relevance (RF-p-MR) approaches.</td>
</tr>
<tr>
<td>6.3</td>
<td>For classification setting, RF-point with standard relevance (RF-p-SR) versus mapped relevance (RF-p-MR) approaches.</td>
</tr>
<tr>
<td>6.4</td>
<td>Varying $K$ with for regression setting on validation set, $K \in {\log(M) + 1, \sqrt{M}, 1/4M, 1/2M, 3/4M, M}$. Significance test result is performed between every two consecutive settings of $K$.</td>
</tr>
<tr>
<td>6.5</td>
<td>With test sets, the best values of $K (=1/4M)$ found from validation sets.</td>
</tr>
<tr>
<td>6.6</td>
<td>Comparison between RF-point and RF-point-weighted.</td>
</tr>
<tr>
<td>6.7</td>
<td>Performance of various RF-based algorithms on smaller datasets. Each metric is the average of five independent runs, and each run is the average of five folds. For RF-point(/list)-S%, the sub-sample (per tree) percentage (and size in terms of queries) are (for the six datasets, in the order of top to bottom): 10, 10, 6, 7, 9, and 4. The algorithms are: RF-rand (rfr), RF-point-classification (rfpc), RF-point-classification-S% (rfpcs), RF-point-regression (rfpr), RF-point-regression-S% (rfprs), RF-list (rfl), RF-list-S% (rfls).</td>
</tr>
<tr>
<td>6.8</td>
<td>Lambdamart (LMart), Coordinate Ascent (CooAsc), AdaRank (AdaR), Mart, RankBoost (RankB), RankSVM (rSVM) on small to moderate-sized datasets.</td>
</tr>
<tr>
<td>6.9</td>
<td>Using Tables 6.7 and 6.8, the ranks and aggregate ranks of all algorithms (in terms of NDCG@10) across the six datasets. The lower the rank, the better.</td>
</tr>
</tbody>
</table>
6.10 BM25 and Language Model (LM) performance on the six datasets along with the best and worst LtR performers among the algorithms we have investigated (cf. Tables 6.7 and 6.8). The percentage increase (or decrease) of performance of the best LtR method over BM25 is also reported in bracket. ................................................. 126

6.11 Performance of various RF-based algorithms on big datasets. Abbreviations are as follows: P for RF-point, H for RF-hybrid, cla for classification, reg for regression, rand for RF-rand. ................................................. 126

6.12 Performance of various algorithms on big datasets. The algorithms are: Mart, RankSVM (rSVM), AdaRank (AdaRa), CoorAsc, RankBoost (RBoost), LambdaMart (LmMart), and BM25 score. ................................................. 126

6.13 Significance test results for comparison of different algorithms from Tables 6.11 and 6.12. A pairwise test is performed, and the winning algorithm (among the two given in the heading of a column) is mentioned in a cell (along with the corresponding $p$-value). The examined algorithms are: RF-point-cla (cl), RF-point-reg (rg), RF-point-reg-K (rk), RF-hybrid-L6-cla (hc), RF-hybrid-L6-reg (hr), LambdaMart (lm), and RF-hybrid-L4-reg-K (hrk). ................................................. 127

7.1 Using permutations of documents, instantiations of different quantities. ................................................. 147

7.2 Using score lists of documents, instantiations of different quantities. ................................................. 150

7.3 Using IR metrics on score lists of documents, instantiations of different quantities. ................................................. 152

7.4 Comparison between RF-point (RF-p) and LambdaMart (LMart) using SRE and VRE on MSLR-WEB10K and Yahoo datasets. ................................................. 160

B.1 Performance comparison among RF-point (baseline), RF-point with oversampling (overall), and RF-point with oversampling (per query). An average over five independent runs is reported (and each run is the result of five-fold cross-validation), and the winning value is given in *italic* font. 173

D.1 Results of RF-list-S5 with different node exploration strategies on MSLR-WEB10K (fold 1) dataset. ................................................. 177
# Abbreviations

<table>
<thead>
<tr>
<th>Abbreviation</th>
<th>Full Form</th>
</tr>
</thead>
<tbody>
<tr>
<td>IR</td>
<td>Information Retrieval</td>
</tr>
<tr>
<td>LtR</td>
<td>Learning to Rank</td>
</tr>
<tr>
<td>VSM</td>
<td>Vector Space Model</td>
</tr>
<tr>
<td>BIM</td>
<td>Binary Independence Model</td>
</tr>
<tr>
<td>LM</td>
<td>Language Model</td>
</tr>
<tr>
<td>SVM</td>
<td>Support Vector Machine</td>
</tr>
<tr>
<td>RF</td>
<td>Random Forest</td>
</tr>
<tr>
<td>TF</td>
<td>Term Frequency</td>
</tr>
<tr>
<td>IDF</td>
<td>Inverse Document Frequency</td>
</tr>
<tr>
<td>DCG</td>
<td>Discounted Cumulative Gain</td>
</tr>
<tr>
<td>NDCG</td>
<td>Normalized Discounted Cumulative Gain</td>
</tr>
<tr>
<td>IDCG</td>
<td>Ideal Discounted Cumulative Gain</td>
</tr>
<tr>
<td>MAP</td>
<td>Mean Average Precision</td>
</tr>
<tr>
<td>ERR</td>
<td>Expected Reciprocal Rank</td>
</tr>
<tr>
<td>MSE</td>
<td>Mean Squared Error</td>
</tr>
<tr>
<td>SRE</td>
<td>Systematic Rank-prediction Error</td>
</tr>
<tr>
<td>VRE</td>
<td>Variability in Rank-prediction Error</td>
</tr>
</tbody>
</table>
Symbols

\( N \) \# instances (query-doc pairs)

\( M \) \# features (base rankers)

\( C \) \# classes (relevance labels)

\( D \) A Dataset

\( q \in Q \) A query \( q \) amongst a set of queries \( Q \)

\( n_q \) \# docs for query \( q \)

\( d_{q,i} \) \( i \)th document of query \( q \) (\( 1 \leq i \leq n_q \))

\( l_{q,i} \) Relevance label for \( i \)th doc of query \( q \)

\( Q_D \) Set of queries present in dataset \( D \)

\( \bar{x}_{q,i} = \vec{\psi}(q, d_{q,i}) \) Feature vector corresponding to \( i \)th doc of query \( q \)

\( \vec{\psi}(\cdot) \) \( i \)th base ranker (\( 1 \leq i \leq M \))

\( t \) Term (i.e., word)

\( C \) Collection of unlabelled documents

\( C_t \) Number of docs where \( t \) appears at least once

\( f \in \mathcal{F} \) Retrieval function \( f \) amongst set of models \( \mathcal{F} \)

\( s_f(q, d) \) Score assigned by \( f \) to document \( d \) for query \( q \)

\( \text{rank}_f(d_{q,i}) \) Rank position of doc \( d_{q,i} \) in a list produced by \( f \)

\( \mathcal{L}(D; f) \) Loss function applied to dataset \( D \) for function \( f \)

\( E \) Size of the ensemble (\# trees)

\( K \) \# features randomly chosen at a node

\( n_{\text{min}} \) Minimum \# instances needed to split a node

\( p \) \# queries (and associated docs) in a sub-sample per tree

(default value is \( 0.63|Q| \))

\( n_{\text{irrel}} \) \# non-relevant docs to sample per query

\( \sigma^2 \) Variance of a model at a datapoint
Symbols

\( b \)  Bias of a model at a datapoint.
\( \rho \)  Correlation between trees of an RF
\( B \)  \# ensembles/models
\( x_k \)  \( k \)th feature vector, \( 1 \leq k \leq N \)
\( l_k \)  \( k \)th true relevance label, \( 1 \leq k \leq N \)
\( IDCG(q) \)  Ideal DCG of query \( q \)
\( DCG(q) \)  DCG of query \( q \)
\( NDCG(q) \)  NDCG of query \( q \)
\( s_i(x_k) \)  Score produced by \( i \)th tree \( T_i \) for feature vector \( x_k \)
\( Err \)  Generalization error
\( loss(.) \)  Loss of an LtR algorithm
List of Publications

The following papers have been published during the course of this research.


• **Conference papers:**


• **Under preparation:** “Understanding Bias-Variance Profiles of Rank-learners”, Muhammad Ibrahim and Mark Carman. Being prepared for a top-tier journal. Based on Chapter 7.
Chapter 1

Introduction

In the last few decades, there has been an overwhelming surge in the volume of digital data due to the proliferation of information and communications technology [127]. Getting the required information from this vast ocean of data has eventually become so formidable that people started using machines from late 1970s to get assistance, thereby giving rise to information retrieval (IR) systems (i.e., search engines). Broadly, the task of an IR system is to return a list of items to the users in response to specific information need, and the items are sorted by their degree of relevance [101, Ch. 1]. This task appears in many domains such as document ranking, recommender systems, automatic question answering, automatic text summarization, online advertising, sentiment analysis, web personalization and so on. In fact, any task which presents the user a list of items on-demand ordered by a utility function is a ranking task. In this thesis, we address the document ranking problem without any loss of generality, as the discussed techniques are applicable to most of the other ranking domains as well.

The working procedure of an IR system can be described as follows. Suppose a user is in need of some information as shown in Figure 1.1. She concocts a query in a natural language which reflects her information need. She then submits the query to an IR system. The system estimates the information need from the given query, and then ranks all the documents at its disposal in descending order of their degree of relevance to the information need. Since the user is interested in only highly-relevant documents, retrieving all documents that are highly-relevant to the query and placing them in the top part of the presented list is crucial from the retrieval system’s perspective.

Traditionally, relatively simple scoring functions (also known as base rankers) have been used to address the ranking problem. A scoring function takes a document and a query representation as input and produces a relevance score for that document. The documents are then sorted in descending order of these scores before being presented to the
user. In the past decade, however, the use of machine learning techniques for computing the relevance scores has received much attention, both in academia and industry [94] due to their high effectiveness which is called the learning-to-rank (LtR) problem. This framework takes a statistical learning approach which uses labelled data (i.e., relevance predictions given by a set of base rankers along with the true relevance labels of the documents) to predict the degree of relevance of unseen documents with respect to a query. Today most of the commercial search engines are known to be using LtR methods. In academia, the use of LtR algorithms in a variety of tasks (some of which are mentioned above) is ever-increasing [94, Ch. 14]. Learning-to-rank algorithms are thus at the core of a modern IR system.

1.1 Motivation

The following shortcomings of the existing literature of LtR systems have motivated us to conduct research presented in the thesis.

- **Accuracy.** It is true that the accuracy of commercial search engines of our era is astonishing. That said, the performance of their ranking algorithms can still be improved. In a recent challenge organized by a well-known commercial search engine, Yahoo\(^1\), the organizers commented on the accuracy of the state-of-the-art LtR algorithms: “there is probably still some room for improvement” [25]. In IR, improvement of accuracy by as small amount as 0.01-0.02 (1-2%) in terms of popular evaluation metrics such as NDCG and MAP (these metrics will be discussed in Chapter 2) is treated to be considerably better performance [27, 165]. It is believed by many IR researchers (such as Manning et al. [101, Sec. 8.5.1]) that despite their simplified assumptions, these metrics are good enough to capture

\(^1\)https://search.yahoo.com/
the user satisfaction. Hence improving these metrics could in turn yield greater profitability.

- **Efficiency.** In existing literature the accuracy issue of the rank-learning algorithms has received much of the attention, while scalability and other issues (such as robustness) have comparatively been less-explored. Computational efficiency and scalability is an important issue in LtR, especially when data size is big [26], [90, Ch. 7], [94, Ch. 20]. Many state-of-the-art algorithms suffer from this problem [1, 22, 140]. For smaller enterprises with limited computing resources, improving computational efficiency becomes even more important [126].

- **Robustness.** It is well-known in IR that a system can be said to outperform another system only if this phenomenon is observed across a wide range of document collections [136]. Diverse properties of data may cause variations in performance of ranking functions [85]. In real life scenarios, properties of the dataset do vary greatly. For example, two publicly available LtR datasets released by two search engine giants, Microsoft and Yahoo, have drastically different properties (as will be discussed in Section 2.9.1). In particular, the training data size is significant because acquiring human-labelled data is costly and time-consuming [6, 40], especially in domain-specific search and enterprise search [147]. Hence sufficient training data may not always be available to LtR algorithms. The reality is, while some algorithms are known to perform well on big data, their performance on datasets with unusual properties (e.g., small size, highly imbalanced label distribution) are not reported often in the literature (e.g., the case of LambdaMart [159] algorithm). In addition to this, data of smaller enterprises may possess different properties than that of web search\(^4\), and their search requirements may vary from commercial search engines [67, 70, 102, 115]. Similarly, the search requirements of domain-specific search (also known as vertical search) may not necessarily match with that of traditional general-purpose web search [67, 167]. For these reasons there is a need to design techniques that can perform well under many different situations.

A good number of ranking algorithms have been investigated in the literature. In this thesis we focus on a specific machine learning framework called a random forest (RF)

\(^2\)Even though commercial search engines possess huge computational resources, they need to frequently test newly developed systems (before releasing them to relevant departments) to check if the new setup works well. As additional computation comes at a cost, the training time of the algorithms is crucial to them.

\(^3\)Despite the fact that research is being conducted to automatically predict labels from click-through data [164], these methods come with their own limitations in terms of reliability as well as the fact that they could possibly cause feedback.

\(^4\)Enterprise search is also sometimes termed as federated search.
which is an effective and efficient framework for supervised machine learning. It learns an ensemble of decision trees, and predicts the label for a datapoint based on the individual predictions of the trees. (Details of this framework will be discussed in Sections 2.7 and 3.5.) We concentrate on RF-based LtR algorithms due to the following reasons:

- A random forest has several benefits over many other state-of-the-art techniques. These include inherent scalability and ease of use (in terms of sensitivity to the parameters).
- A random forest has been found to be highly effective in many domains.
- In IR, a random forest based ranking algorithms (a few though) have already demonstrated competitive performance as compared to other (oftentimes more complex) algorithms.

(Detailed discussion on these points is given in Section 2.7).

Despite the above-mentioned potential, the random forest based algorithms in LtR domain has not yet been thoroughly investigated. This thesis aims at filling this gap in the existing literature.

1.2 Aims

This thesis aims at investigating the potentials of RF-based LtR algorithms with a focus on alleviating the bottlenecks of the existing methods as alluded earlier. Specifically, we are interested in the following key objectives:

1. To investigate scalability issues of existing RF-based LtR algorithms.
2. To design RF-based novel LtR algorithms for achieving better accuracy.
3. To identify robust LtR algorithms across datasets with diverse properties by comparing performance of RF-based LtR algorithms with relevant state-of-the-art algorithms using publicly available datasets.
4. To develop a better theoretical understanding of RF-based LtR algorithms.
1.3 Research Questions

The main research question tackled in this thesis is the following:

*How can we improve efficiency, effectiveness and understanding of random forest based rank-learning algorithms?*

In particular, the following sub-questions are investigated:

1. *Given that the relevance label distribution of LtR data is heavily skewed, how can we leverage the undersampling techniques in conjunction with the settings of random forest to achieve better scalability?*

2. *How can we reduce the correlation between the trees of a random forest in favor of performance and scalability?*

3. *What is the effect of using diverse objective functions in a random forest?*

4. *How can we better understand the theoretical underpinnings (e.g. bias-variance profiles) of these algorithms?*

Table 1.1 shows the topics covered by the research sub-questions along with the chapter numbers where they are investigated.

**Table 1.1:** Research sub-questions, objectives and the chapter numbers where they are addressed.

<table>
<thead>
<tr>
<th>Sub-question No.</th>
<th>Objective</th>
<th>Chapter</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Efficiency</td>
<td>3 and 4</td>
</tr>
<tr>
<td>2</td>
<td>Effectiveness and efficiency</td>
<td>4 and 6</td>
</tr>
<tr>
<td>3</td>
<td>Effectiveness</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>Theoretical understanding</td>
<td>7</td>
</tr>
</tbody>
</table>

1.4 Major Contributions

Below we summarize the major contributions of this research grouped by the key objectives.

**Efficiency**

- The LtR data are usually highly imbalanced in terms of their relevance label distributions. We investigate the optimum amount of non-relevant documents
required for learning an effective ranking function by employing undersampling techniques. This investigation improves scalability of LtR algorithms, especially for highly imbalanced datasets. [Chapter 3.]

- We utilize the inherent structure of a random forest to sample a subset of non-relevant documents. That is, we sample the non-relevant documents on a per-tree basis instead of per-model basis. We show that this approach improves scalability over the standard approach (i.e., the previously mentioned one). [Chapter 3.]

- The correct amount of correlation between the trees is crucial for performance of an RF-based LtR algorithm. Driven by this theory, we reduce the correlation among the trees of an ensemble by learning a tree using a smaller sample. We demonstrate that this method improves scalability and performance, especially for relatively smaller datasets. By objectively measuring different quantities such as correlation, we validate our method. [Chapter 4.]

**Effectiveness**

- While optimising a surrogate metric yields good performance for LtR, it is known that optimising the exact evaluation metric is likely to yield better accuracy (the former and latter class of algorithms are known as pointwise and listwise algorithms respectively). We, therefore, design an RF-based based listwise algorithm which directly optimises a rank-based metric (instead of a surrogate metric such as classification accuracy). We show that in many datasets performance of this algorithm is better than its (existing) pointwise counterpart. [Chapter 5.]

- The listwise LtR algorithms usually are computationally more intensive than their pointwise counterparts, and our proposed RF-based listwise algorithm is no exception. Hence we develop a random forest based hybrid algorithm which can incorporate a variety of splitting criteria in a single tree. This is useful when some splitting criteria are computationally more expensive (e.g. rank-based ones) than others (e.g. classification accuracy-based ones). We empirically show that its performance is at least as good as its pointwise counterpart. [Chapter 5.]

- To better understand the role of splitting criteria in a tree of an RF-based LtR algorithm, we devise an LtR algorithm with purely-randomized trees. This algorithm has huge computational benefit over many other LtR algorithms. From theoretical point of view, this algorithm resembles with the classical nearest-neighbor approach. For small to moderate-sized datasets, this algorithm performs close to
its pointwise counterpart. Also, it performs better than a few state-of-the-art algorithms which is somewhat counterintuitive given its completely random splitting criterion. [Chapter 5.]

- To complement our investigation into the objective functions of RF-based LtR algorithms, we develop a random forest based pairwise but greedy algorithm. This algorithm minimizes the number of miss-ranked pairs of documents during training in a greedy fashion. On small and moderate-sized datasets, we show that this algorithm tend to perform slightly better than its pointwise counterpart. [Chapter 5.]

- In the existing literature it is not thoroughly investigated as to whether classification or regression setting performs better for RF-based pointwise algorithms. Hence we perform an empirical comparison between the two approaches. We also examine the effect of using absolute relevance judgement versus mapped relevance. In addition, we design a weighted version of RF-based LtR algorithms where each tree is assigned a weight based on its individual ranking accuracy. [Chapter 6.]

- The single parameter to which RF-based algorithms may be sensitive (especially for big datasets) is the number of candidate features ($K$) at each node of a tree. From theoretical point of view, this parameter can be related to the tree complexity, variance and correlation. We investigate the optimal choice of this parameter. [Chapter 6.]

- We conduct an extensive comparison between the investigated RF-based algorithms and several state-of-the-art algorithms on publicly available eight LtR datasets. This comparison reveals that the (good) performance of RF-based algorithms is maintained across datasets with diverse properties (such as smaller datasets, imbalanced datasets etc.) as compared to other algorithms. [Chapter 6.]

**Theoretical Understanding**

- We explain the evolution of a random forest from a single tree from a bias-variance perspective. [Chapter 7.]

- We design a framework to measure the systematic ranking error and the ranking error due to variability in predictions of rank-learners across different samples. This framework is able to explain relative performance of different algorithms in terms of bias and variance. [Chapter 7.]

- We examine the bias and variance of an RF-based rank-learner as we vary the sub-sample size per tree (this parameter was investigated in Chapter 4). [Chapter 7.]
Chapter 1: Introduction

1.5 Organization

The rest of the thesis is structured as follows.

Chapter 2 highlights the basic background knowledge required for this thesis, and reviews the relevant literature. It then elaborates the benefits of random forest over many other state-of-the-art methods. Finally, the experimental settings of the thesis are explained. A extended version of the first part of this chapter has been published as a Chapter in Handbook of Research on Innovations in Information Retrieval, Analysis, and Management [79].

Chapter 3 examines the importance of non-relevant documents (which comprise the vast majority) present in a typical LtR training set. It employs undersampling techniques in favor of better scalability of the LtR algorithms. Initial thoughts of this work have been presented in 14th Asia Information Retrieval Society Conference [78].

Chapter 4 investigates the optimal amount of correlation between individual trees of an RF. We investigate, in conjunction with the parameter sub-sample size per tree, the reduction of correlation between the trees of an RF-based LtR algorithm. The aim here is to improve both the performance and scalability. Preliminary results of this work have been presented in 12th Australasian Data Mining Conference [77].

Chapter 5 compares various objective functions for random forest based LtR algorithms. Objective functions play a central role in the accuracy of LtR algorithms. This chapter introduces a listwise objective function, and designs a few novel algorithms. The chapter also conducts deep and thorough investigation into a number of aspects of the designed listwise objective function. This entire work has been accepted for publication in ACM Transactions on Information Systems [76].

Chapter 6 serves two purposes. Firstly, this chapter conducts further investigation into performance of RF-based pointwise algorithms from a number of perspectives. Secondly, it thoroughly compares the performance of the investigated LtR algorithms (in the previous chapters) with state-of-the-art algorithms on a number of datasets.

Chapter 7 elucidates on the theoretical underpinnings of RF-based rank-learners. The reduction of correlation using sub-sample size per tree was investigated in Chapter 4, but the relationship with bias and variance was out of the scope of that chapter. In this chapter we analyse the intrinsic relationship between bias-variance and error of rank-learners as we vary the above-mentioned parameter, i.e., the sum-sample size per tree.

Chapter 8 summarizes the thesis, and concludes it.
Chapter 2

Background, Literature Review and Experimental Setup

In this chapter we first describe the evolution of the LtR approach from conventional ranking functions. We then discuss the common evaluation metrics used in IR as their knowledge is essential for reviewing the LtR literature. After that we formalize the LtR problem, which is followed by a brief review of the existing LtR algorithms. We then emphasize the benefit of a random forest over many other state-of-the-art techniques as this partly motivates us to investigate RF-based rank-learners. Finally we explain the datasets which will be used for experimentation, and experimental settings which are largely maintained throughout the thesis.

2.1 The Road to Learning-to-Rank

The working procedure of an IR system was explained in the beginning of the previous chapter. In this section we first highlight a few traditional approaches used to solve the IR problem. We then introduce the LtR framework.

2.1.1 Traditional Approaches

In this subsection we briefly discuss some conventional ranking methods. More detailed discussion of these and other traditional models can be found in Manning et al. [101].
2.1.1.1 Vector Space Model (VSM)

This model \cite{133, 134} is mainly based on two simple but yet convincing ideas: (1) the more the occurrences of the words of a query a document contains, the more likely is that the document be relevant with respect to the query; and (2) the importance of a rare term (across the whole collection of documents) present in the query is higher than that of a comparatively more frequent term. The intuition for these assumptions is that a document containing a more frequent term may be relevant to many queries because a frequent term across the collection is also likely to be frequent in the queries; but a rare term in a document is likely to be rare in the queries as well. When a user mentions a rare term in the query, it is highly likely that she is looking for the few documents that contain the said rare term. This model treats the query as a separate document (using the bag-of-words assumptions), and then computes the cosine similarity among the documents and the query. For a document \(d\) and query \(q\), the score of \(d\) is defined as follows:

\[
\text{score}(d, q) = \sum_{t \in q} tf(t, d) idf(t, C),
\]

(2.1)

where \(tf(t, d)\) is term frequency – the number of occurrences of \(t\) in \(d\), and \(idf(t, C)\) is inverse document frequency – a measure which computes the rarity of term \(t\) in the collection\(^1\) of term \(t\) in the collection \(C\). After computing scores of all the documents, they are ranked in decreasing order of the scores, and presented to the user.

2.1.1.2 Probabilistic Model I: Binary Independence Model (BIM)

This model assumes that there is a binary random variable \(R\) which takes the value 1 if a document \(d\) is relevant with respect to a query \(q\); and 0 otherwise. The relevance of each document is assumed to be independent of that of other documents. The documents and queries are represented by binary vectors of terms of the collection (occurring in the document/query or not), and assumes that the terms are independent of each other. Using Bayes rule, it computes the following:

\[
P(R = 1|d, q) = \frac{P(d|R = 1, q)P(R = 1|q)}{P(d|q)}. \tag{2.2}
\]

The probabilities in the right hand side of the above equation are approximated from the collection using the bag-of-words model. The documents are then ranked in decreasing order of these probabilities. Taking this model as a general framework, many

\(^1\)For example, \(idf(t, C) = \log(C_t/|C|)\), \(C_t\) is the number of documents where \(t\) appears at least once, \(|C|\) is the total number of documents in the collection.
scoring functions have been proposed in the literature. Among these, Okapi BM25 (i.e., Best Match 25) model [145] which incorporates the term frequency (and hence, unlike BIM, not binary) and length normalization has been widely used until date by the IR community due to its empirical superiority.

### 2.1.1.3 Probabilistic Model II: Language Model (LM)

This model [121] is also probabilistic, but takes a different approach from BIM. It assumes that when a user wants to devise a query, she already has some relevant document(s) in her mind that she wants the IR system to retrieve. She then creates a query consisting of some distinctive (i.e., discriminative) terms present in those documents by which it is easy to single out them from the other documents of the collection. Since the user is assumed to be devising the query using some relevant documents in her mind, the query can be thought of being “generated” exclusively from those relevant documents, and not by other (non-relevant) documents of the corpus. As such, if we model the probability $P(q|d)$, then for the relevant documents in the user’s mind, these probabilities would be higher. Hence this probability can be used as a relevance score for ranking a document. Alternatively, we can model $P(d|q)$ using Bayes rule; in doing so, we can incorporate the prior probability of the document being relevant in general (for example, popularity of the document). Concretely:

$$P(d|q) = \frac{P(q|d)P(d)}{P(q)}.$$  (2.3)

The probabilities are computed from the collection, and the independence of the terms is assumed for simplicity in a probability calculation. These probabilities are then ranked in decreasing order.

Regarding a general comparison between the probabilistic models and tf-idf scorers, while some researchers deem the probabilistic models to be better than the tf-idf scorers [129, Sec. 3.4.3], some others disagree [150].

### 2.1.2 Learning-to-Rank Approach

The conventional approaches discussed in the previous sub-section have an inherent bottleneck: several parameters need to be tuned. For example, in VSM, one can use either $tf\cdot idf$ (cf. Equation 2.1) or some other variations of it such as $log(tf\cdot idf)$ [149]. Regarding BM25 model, Lv and Zhai [97] suggest that the $k_1$ parameter (which controls the $tf$ normalization component) of standard BM25 formula should be term-specific instead of term-independent. Language models are also known to be sensitive to their
smoothing parameters [174]. Traditionally the best settings of these parameters are selected on a trial-and-error basis. This strategy has been found to work with acceptable performance in general [37, 105]. But the following reasons have paved the way to design more complex techniques.

• **No single model always wins.** Although some researchers such as Taylor et al. [149] applaud the traditional models by saying: “They are robust and generalize well” (which means that these models often work well on new corpora with the default parameter settings), no one claims that a particular model always wins over the others in all types of collections and queries [85, 98, 104, 136]. Hence a natural inclination would be towards using more than one model at the same time in order to get better accuracy.

• **Too many useful signals.** Now-a-days incorporating more than one relevance measure has become not only a choice, but also a necessity due to the emergence of an increasing number of relevance signals in the research community. For example, one can compute separate similarity scores (e.g., Equation 2.1) for different segments of a document such as the title, abstract and the body [130]. Also, term proximity based features have turned out to be important [106]. For web search, the link information (e.g. the popularity of a webpage as measured by the prominent PageRank algorithm [117]) has been found to be an influential signal along with standard relevance measurements like the frequency based ones. Moreover, the availability of vast user log data from commercial web search engines started to lure the designers of search engines to develop more complex methods because most of the traditional models cannot incorporate these additional information collected from the users.

Given the importance of using more than one model (i.e., relevance measurements, also known as base rankers) simultaneously, if we want to use a linear combination of different models, we need to optimally select the weights (i.e., coefficients) of each measurement in the combination. Or, possibly better, we can think of using a nonlinear combination. Such a combination for a document \( d \) with respect to a query \( q \) can be computed in the following way:

\[
\text{score}(d, q) = \omega_1 g_1(\psi_1(d, q)) + \omega_2 g_2(\psi_2(d, q)) + \ldots + \omega_M g_M(\psi_M(d, q)),
\]  

(2.4)

where \( M \) is the number of base rankers, \( \psi_i(d, q) \) is the relevance score predicted by \( i \)th model, \( \omega_i \) is the weight of \( i \)th base ranking model, and \( g(\cdot) \) is a function which decides how the score of \( i \)th model will be used – for example, if \( g(\psi_1(d, q)) = \psi_1(d, q) \),
then the combination of the models is linear. Now the task is to find a solution to Equation 2.4. To this end, supervised machine learning is employed which leads to the notion of learning-to-rank.

### 2.2 Problem Formulation of Learning-to-Rank

Suppose we have a set of queries \( Q = \{q_i\}_{i=1}^m \), where each query \( q \) is associated with a set of \( n_q \) documents \( D_q = \{d_{q,i}\}_{i=1}^{n_q} \). Each query-document pair \( \langle q,d_{q,j} \rangle \) is further associated with a feature vector \( \vec{x}_{q,j} = \vec{\psi}(q,d_{q,j}) = \langle \psi_1(q,d_{q,j}),...,\psi_M(q,d_{q,j}) \rangle \in \mathbb{R}^M \) and a relevance label \( l_{q,j} \), such that a dataset \( D \) consists of \( N = \sum_q n_q \) feature vectors and corresponding labels.

The elements of the feature vector (i.e., \( \psi_i(.) \)) are scores of other simple rankers, for example the cosine similarity over tf-idf vectors for the particular query-document pair, or BM25 score for the same pair, etc. These scores are usually normalized at the query level, such that each feature value lies in the range 0 to 1 [124].

Pictorially the training set looks like Figure 2.1.

A ranking function is then a function \( f:\mathbb{R}^M \to \mathbb{R} \) that assigns scores to feature vectors, i.e., query-document pairs. Given a query \( q \), let \( \vec{s}_f(q) = \langle f(\vec{x}_{q,1}),\ldots,f(\vec{x}_{q,n_q}) \rangle \) denote the vector of scores assigned by \( f \) to the documents for that query. The vector can then be sorted in decreasing order to produce a ranking:

\[
\text{sort}(\vec{s}_f(q)) = \langle \text{rank}_f(d_{q,1},q),\ldots,\text{rank}_f(d_{q,n_q},q) \rangle,
\] (2.5)
where \( \text{rank}_f(d_{q,i}, q) \) returns the position of document \( d_{q,i} \) in a sorted list based on the scores generated by \( f \).

Given a training set \( D \), the goal of an LtR algorithm is to find the function \( f \) amongst a set of functions \( F \) that minimises a loss function \( L \) over the training set:

\[
f^* = \arg\min_{f \in F} L(D; f)
\]  

(2.6)

Ideally, the loss function should optimise the metric which will be used for evaluation. As such, a typical example of a loss function is one based on Normalised Discounted Cumulative Gain (NDCG):

\[
L(D; f) = 1 - \frac{\text{NDCG}(Q; f)}{\text{max}_{f'} \text{DCG}(Q; f')}
\]  

(2.7)

\[
\text{NDCG}(Q; f) = \frac{1}{|Q|} \sum_{q \in Q} \text{NDCG}(q; f)
\]  

(2.8)

The NDCG for a particular query is defined as the normalised version of the Discounted Cumulative Gain (DCG), as follows:

\[
\text{DCG}(q; f) = \sum_{r=1}^{n_q} \frac{\text{gain}(l_{q,\text{doc}(r,q,f)})}{\text{discount}(r)}
\]  

(2.9)

\[
\text{NDCG}(q; f) = \frac{\text{DCG}(q; f)}{\text{max}_{f'} \text{DCG}(q; f')}
\]  

(2.10)

where \( \text{doc}(r, q; f) = \text{rank}^{-1}_f(r, q) \) denotes the inverse rank function, i.e. the document at rank \( r \) in the list retrieved by model \( f \), and thus \( l_{q,\text{doc}(r,q,f)} \) denotes the relevance judgement for document in position \( r \) for query \( q \).

Sometimes NDCG is truncated at a value \( k \leq n_q \), and is denoted by NDCG@\( k \). Throughout the thesis we shall use NDCG to refer to the un-truncated version.

The gain and discount factors used to weight rank position \( r \) are usually defined as:

\[
\text{gain}(l) = 2^l - 1
\]  

(2.11)

\[
\text{discount}(r) = \log(r + 1)
\]  

(2.12)

We note that LtR algorithms mainly differ in defining the loss function \( L(D; f) \). Equation 2.7 and other ranking loss functions based on IR metrics are difficult to optimise directly because (i) they are non-differentiable with respect to the parameters of the retrieval function (due to the sorting in Equation 2.5) and (ii) they are almost invariably
non-smooth and non-convex. Hence the researchers have attempted a range of heuristics to define $\mathcal{L}(\mathcal{D}; f)$ which have given rise to a large number of LtR algorithms.

Figure 2.2 depicts the overall scenario of an LtR-based IR system. Its task can be broadly divided into two subtasks, namely training phase and test/evaluation phase. During the training phase, the first job is to create the training data from a query collection and a document collection (this procedure will be elaborated in Section 3.1). A model $f(x)$ is then learnt from the training data. During evaluation phase, a query is taken from the user, and then the same base ranker(s) (which was used during the training data creation process) is applied to retrieve the top $k$ documents for the query. A feature vector is then computed for each of these documents with respect to the query. After that, the learnt function $f(x)$ is applied to predict a relevance score for each of these documents. Finally, the documents are sorted in decreasing order of their scores, and then presented to the user.

2.3 Evaluation Metrics Used in Information Retrieval

Evaluation metrics in IR depend on the ranks of the documents retrieved for a query. These ranks are generated when the documents are sorted in descending order of their relevance scores predicted by an IR system – be it conventional or LtR based. Below we briefly discuss three commonly used metrics.

2.3.1 Discounted Cumulative Gain (DCG) and Normalized DCG (NDCG)

If a ranked list contains $n_q$ documents sorted in descending order by their degree of relevance with respect to a query $q$, and $l_{q,i}$ is the true relevance judgement of $i$th
document (in the sorted list), then DCG at position $k$ is given by:

$$DCG@k = \sum_{i=1}^{k} \frac{gain(i)}{discount(i)} = \sum_{i=1}^{k} \frac{2^{q_i} - 1}{\log_2(i + 1)}; \quad k \leq n_q. \quad (2.13)$$

Discounted cumulative gain works reasonably well at capturing the human judgement, but it is inappropriately biased towards queries having more relevant documents. To mitigate this drawback, normalized DCG (NDCG) is used, which normalizes DCG of a list by the DCG of the most accurate ranking possible for that list. Formally, given IDCG is the DCG of the corresponding ideal (i.e., perfect) ranking, NDCG at position $k$ is expressed by:

$$NDCG@k = \frac{DCG@k}{IDCG@k}. \quad (2.14)$$

### 2.3.2 Mean Average Precision (MAP)

If $Rel_k$ is the number of relevant documents among the top $k$ positions, then precision up to position $k$, $p@k$ is expressed by:

$$p@k = \frac{Rel_k}{k}. \quad (2.15)$$

Average precision (AP) is defined as the average of the precisions at each position of a ranked list where a relevant document is found. If the set of positions of relevant documents is denoted by $rel_{positions}$, then AP is defined as:

$$AP = \frac{1}{|rel_{positions}|} \sum_{i \in rel_{positions}} p@i. \quad (2.16)$$

MAP is simply the average of APs across all queries of a test set.

### 2.3.3 Expected Relevance Rank (ERR)

If the probability that the user has found his/her required document at $i$th position of a ranked list is $P(i)$, then ERR [28] is defined as:

$$ERR = \sum_{i=1}^{n_q} \frac{1}{i} P(\text{user stops at } i\text{th document}) = \sum_{i=1}^{n_q} \frac{1}{i} R_i \prod_{j=1}^{i-1} (1 - R_j), \quad (2.17)$$
where \( R_j = \frac{q^j_{\text{maxRel}}}{\text{maxRel}}, \text{maxRel} \) is the numerical value of the highest relevance label.

More details on these and other metrics are discussed in [28, 82, 135].

## 2.4 Existing LtR Approaches

This section overviews the existing LtR algorithms.

### 2.4.1 Important Aspects of an LtR Algorithm

Two major components of a supervised learning framework are: (1) the type of loss function which is optimised during training, and (2) the learning model which is used to optimise the loss function (e.g. SVM, neural network etc.). Variations in these aspects have given rise to a large number of LtR algorithms. Below we review the existing LtR algorithms in the context of these two dimensions. More details of the algorithms can be found in survey papers such as [90] and [94].

1. The question here is, how is the loss function defined? For example, does it take the query-document structure present in the data into account, i.e., is a loss defined for a query, or does it simply use the loss functions of standard classification or regression problem, i.e., is a loss defined for a query-document pair? In the existing literature, three ways of defining a loss function have been studied. These are described below.

   (a) **Pointwise approach.** The loss function can be defined based on the individual documents (i.e., feature vectors corresponding to query-document pairs). Thus the total loss over a training set which a learning algorithm minimizes during training is written as:

   \[
   \mathcal{L}_{\text{Training}} = \sum_{q=1}^{|Q|} \sum_{j=1}^{n_q} \text{loss}(f(\mathbf{x}_{q,j}), l_{q,j}),
   \]  

   or, in other words, as follows:

   \[
   \mathcal{L}_{\text{Training}} = \sum_{i=1}^{N} \text{loss}(f(x_i), l_i),
   \]

   where, \( \mathbf{x}_{q,j} \) is, in accordance with the problem formulation discussed in Section 2.2, the feature vector, and \( l_{q,j} \) is the corresponding relevance judgement.
(ground truth label), and $N$ is the total number of query-document pairs present in the training set, i.e., $N = \sum_{q=1}^{Q} n_q$ (note that in Equation 2.19 we have slightly abused the notations of $x$ and $l$). The function $\text{loss}(f(\tilde{x}_{q,j}), l_{q,j})$ can be defined as the loss function of a classification problem (e.g. [63, 92, 110]) or a regression problem (e.g. [31, 63, 109]).

This approach has an advantage which is, it can use any classification or regression algorithm off-the-shelf. Also, their computational time and space requirement is oftentimes less than other complex algorithms (as will be discussed shortly). Moreover, it has a theoretical foundation since it has been proven that the IR ranking error in terms of IR metrics (such as DCG) is bounded by both classification error [92] and regression error [31]. If the classification setting is used, the algorithm treats the distinct relevance labels as distinct classes [92, 110]. If the regression setting is used, the algorithm assumes that the relevance labels are the target values [63], and sometimes before training, the absolute relevance labels are mapped onto the range of 0 to 1 [18, 63]. Some of the works that employ a pointwise loss function (besides the already mentioned ones) are Acharyaa et al. [1], Mohan et al. [109], Pavlov et al. [119], Song [144] etc.

Now we describe the differences that exist between classification/regression and the LtR problem. (1) In LtR, the predicted labels themselves are not important, rather the relative ranking of documents is of interest. (2) The IR evaluation metrics are biased towards the top portion of a ranked list. (3) The LtR data are partitioned by queries. Hence a concern with the pointwise algorithms is whether directly using a surrogate loss function for solving the LtR problem is appropriate for the rank-learning task. Nevertheless, empirical results show that several pointwise algorithms do compete with the state-of-the-art systems [1, 25, 63, 109, 119, 144] – mainly due to the fact that both classification and regression errors (as mentioned earlier) are upper bounds to the ranking error.

(b) Pairwise approach. The loss can be defined based on the pairs of instances. The two instances of a pair to be considered should be associated with a particular query, and should have different ranks as ground truth labels. The loss pertaining to the pair of instances $\langle \tilde{x}_{q,j}, \tilde{x}_{q,k} \rangle$ is written as:

$$L_{\text{Training}} = \sum_{q=1}^{Q} \sum_{j=1}^{n_q} \sum_{k=j+1}^{n_q} \text{loss}(f(\tilde{x}_{q,j}), f(\tilde{x}_{q,k}), l_{q,j}, l_{q,k})$$

An example of the function $\text{loss}(f(\tilde{x}_{q,j}), f(\tilde{x}_{q,k}), l_{q,j}, l_{q,k})$ is a binary indicator function which tells that whether the pair of the documents at hand are
properly ranked or not by the current settings of the model parameters (used by Joachims [83]). Some systems of this category are Burges [18], Freund et al. [49], Quoc et al. [125] etc.

An advantage of this approach is, we can apply an existing binary classification algorithm (after modifying the training set – either explicitly or implicitly). Another merit is, the loss function is defined based on the query-document structure present in the LtR data. This approach, if properly utilized, sometimes show better performance than the pointwise algorithms [90, 94, 124].

The main drawback of this approach is, its computational complexity shoots up in quadratic order of the training set size. Similarly, the (negative) effect of noisy examples increase in quadratic order. Another issue is, although it takes the query-document structure into account, a pairwise loss does not fully represent the IR metrics (NDCG etc.) because it gives no emphasis on the top part of a ranked list.

(c) **Listwise approach.** The loss can be computed from all the documents pertaining to a query. It is, therefore, written as follows:

\[
L_{\text{Training}} = \sum_{q=1}^{\mid Q \mid} \text{loss}(f(x_{q,1}), f(x_{q,2}), \ldots, f(x_{q,n_q}), l_{q,1}, l_{q,2}, \ldots, l_{q,n_q}) \quad (2.21)
\]

An example of the function \(\text{loss}(f(x_{q,1}), f(x_{q,2}), \ldots, f(x_{q,n_q}), l_{q,1}, l_{q,2}, \ldots, l_{q,n_q})\) is the degree of disagreement between the true ranked list of all the documents associated with a query and the predicted ranked list of the same documents using K-L divergence measure (used by Cao et al. [22]) - these two ranked lists are created by sorting the true labels \(l_{q,1}, l_{q,2}, \ldots, l_{q,n_q}\) and the predicted labels \(f(x_{q,1}), f(x_{q,2}), \ldots, f(x_{q,n_q})\) during training.

The advantage of this approach is, it tackles the LtR problem more closely to its spirit in the sense that the loss function (\(\text{loss}(.)\)) is defined over all the documents associated with a query. Note that the pairwise approach also deals with the query-document structure, but in a listwise loss function we have more flexibility over how to choose \(\text{loss}(.)\). This flexibility allows the researchers to use smooth approximations to the IR metrics [125, 149]), or to use smooth surrogate (but still listwise) loss functions [22, 161]). Hence its empirical performance in general is said to be better as compared to the pointwise and pairwise approaches [90, 94, 124].

A drawback of this approach is, these algorithms usually computationally expensive (e.g. [22]), and also are conceptually more complex. Another concern is that the listwise algorithms may be relatively more prone to overfitting.
Why IR Metrics are Difficult to Optimise Directly. As discussed earlier, most of the LtR algorithms optimise a surrogate loss function instead of the original one (e.g., Equation 2.7). Now we explain why it is the case. For a particular assignment of the values of the model parameters, a learning algorithm generates a real-valued score for each query-document pair of the training set; and the documents associated with a query are then ranked in decreasing order of their scores so as to get a ranked list for each of the training queries. If the model parameters are changed, the scores of the documents change smoothly, but the order of the documents in the ranked list do not change until the score of at least one document which is below in the ranked list becomes greater than the score of another document which is above the former document. This is because only after this incident, the first document can topple the second document’s rank.\(^2\) Now if we analyse NDCG, we find that it can take on only certain values.\(^3\) Therefore, if we alter the values of the parameters of a learning algorithm, either NDCG does not change, or it takes a steep (discontinuous) jump to the next possible NDCG value. Figure 2.3 depicts the scenario. For this reason, most of the IR metrics such as NDCG are flat or discontinuous everywhere with respect to the parameters of a model. This is a problem for many learning algorithms since to optimise a loss function these algorithms need to calculate the gradient of the loss function. That is why a loss function which directly uses of IR metrics is difficult to optimise.

2. After defining the loss function, we need to decide: which model should be used for optimising the loss function? The existing LtR algorithms employ several such models. The prevalent models are: SVM \[21, 73, 83, 110, 143\], neural network \[18, 149\], boosted tree ensemble \[49, 92, 159, 166\], regression \[31, 176\], and random forests \[63, 109\].

\(^2\)Moreover, these two documents must be associated with the same query and must have different relevance labels; otherwise no preference relation between them should be assumed by the learning algorithm.

\(^3\)If the number of distinct relevance labels is \(r\), then there are \(r^k\) possible values of NDCG@\(k\).
Table 2.1 summarizes the previous discussion, i.e., categorizes some existing algorithms into the pointwise, pairwise and listwise approaches along with their learning models.

<table>
<thead>
<tr>
<th>Model</th>
<th>Loss Function</th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Pointwise</td>
<td>Pairwise</td>
<td>Listwise</td>
</tr>
<tr>
<td>Co-ordinate ascent</td>
<td>-</td>
<td>-</td>
<td>[105]</td>
</tr>
<tr>
<td>Adaboost</td>
<td>-</td>
<td>[49]</td>
<td>[166]</td>
</tr>
<tr>
<td>Gradient boost</td>
<td>[92]</td>
<td>-</td>
<td>[159]</td>
</tr>
<tr>
<td>Neural Net</td>
<td>-</td>
<td>[18], [125]</td>
<td>[22], [149]</td>
</tr>
<tr>
<td>SVM</td>
<td>[110]</td>
<td>[73], [83]</td>
<td>[172]</td>
</tr>
<tr>
<td>Random forest</td>
<td>[63], [109]</td>
<td>-</td>
<td>-</td>
</tr>
<tr>
<td>Linear regression</td>
<td>[31]</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

2.4.2 Discussion

The initial inclination of the researchers was towards pointwise and pairwise approaches, which later seemed to have largely shifted towards the listwise approaches. In general, the empirical performance of the listwise approaches are said to be better than the two other approaches. The pointwise approach, however, enjoys the benefit of having less computational complexity and ease-of-use as compared to the other two approaches. Thus listwise and pairwise approaches usually suffer from the high computational cost. Besides accuracy, computational efficiency and ease-of-use are important factors in LtR algorithms as noted by numerous researchers such as [1, 17, 156]. That is why the pointwise approaches are still employed if their performance is found to be comparable to the state-of-the-art methods [63, 92, 109, 119]).

Another point worth-discussing is that, as hinted by Chapelle and Cheng [25], if the hypothesis space is limited (e.g. linear models), pairwise/listwise loss functions tend to perform better. In contrast, non-linear models (e.g. additive tree ensembles) can already capture very complex relationship among features, that is why these models work quite well with pointwise loss functions.

2.4.3 A Taxonomy of LtR Algorithms

In Figure 2.4 we present a graphical view of the categorization of LtR algorithms. The existing categorization in the literature is based on only the first level of the tree shown in the figure, whereas we have added another layer for better clarification.

Let us explain the figure. First of all, if the loss function is pointwise, then any existing (multi-class) classification or regression framework can be used to optimise the loss function. If the loss function is pairwise, then it may (e.g. LambdaRank [125]) or may
not (e.g. RankBoost [49]) incorporate rank-based information - the former of these two avenues is less explored in the literature, and we conjecture that most of the classification and regression models are amenable to be used. As for the latter field, i.e., if the pairwise loss function uses no rank-based information, then any binary classification model can be used. Finally, if the loss function is listwise, then it can be one of two types: (a) a smooth approximation of the true loss function, or (b) the true loss function itself. In the former case, any standard optimisation method which finds a global optimum point of a smooth loss function can be employed (e.g. neural network [149]). In the latter case, a few learning algorithms such as coordinate ascent and random forest can be used which do not necessarily find a global optimum, but still perform well in practice in many cases (e.g. co-ordinate ascent [105]).

### 2.5 Tasks Related to Learning-to-Rank

In Section 2.4.1 we have highlighted the differences between classification/regression and LtR problem. We now explain three other problems that are related to LtR.

#### 2.5.1 Ordinal Regression

Ordinal regression shares properties of both the classification and regression [162]. Like classification (and unlike regression), here the set of labels is finite. Like regression (and unlike classification), there is an ordering relationship between the labels, i.e., the instances have a preference relationship with one another. The main differences between ordinal regression and LtR are as follows. (1) In LtR we emphasize on the top portion

---

4Recall that an example of the true loss function is $1 - NDCG$. 

---
of a ranked list, whereas ordinal regression has no emphasis. (2) The LtR data are partitioned by queries. (3) In LtR, predictions of instances are not important, rather the ranked list is. In ordinal regression, predictions themselves may or may not be important depending on the application.

2.5.2 Collaborative Filtering

Collaborative filtering is the task of predicting a particular user’s degree of preference of items in terms of a discrete and finite set of values [19]. The main differences between this task and LtR include the first two differences mentioned in the previous sub-section, namely the collaborative filtering has no such emphasis on top portion of a ranked list, and the training data are not partitioned by queries. In addition, there are two more differences: (1) features in collaborative filtering are always discrete with a small range, and (2) unlike LtR, the numerical values of predictions are important.

2.5.3 Rank Aggregation

Given a set of ranked lists (called candidate lists), the task of rank aggregation is to produce a resultant list that minimizes the number of miss-ranked pairs between the final list and the candidate lists [139]. The differences between this task and LtR include: (1) “features” are ranked lists of items, so the feature space required for LtR is missing, and (2) there is no emphasis on top portion of the final (i.e., aggregated) ranked list.

Due to the differences outlined above, LtR has emerged as a separate field of research.

2.6 Theory of Learning-to-Rank

In this subsection we briefly discuss the theory of LtR problem [88, 89, 160, 161].

As mentioned in Section 2.2, the LtR data can be partitioned by query where an instance is all the documents associated with a query. That is, we can treat the query-document pairs \((q_i, d_{q_i,j})\) with all \(j\) as a single instance corresponding to query \(q_i\). Let \(X\) be such a query-level instance, and \(Y\) be the relevance judgements for the documents associated with the query. We assume that the instances \((X, Y)\) are iid (independent and identically distributed) and each such instance comes from an unknown probability distribution \(P(X, Y)\). The goal is to learn a function \(Y = f(X)\) such that some loss function \(\mathcal{L}_{\text{per-instance}}(f(X), Y)\) is minimized across the entire population. Thus the total loss
that we want to minimize is:

$$ L_{total} = \mathbb{E}_{X,Y}[L_{per\_instance}(f(X), Y)] = \int_{X,Y} L_{per\_instance}(f(X), Y) P(X,Y) dX dY $$

(2.22)

During learning, we want to find a function $f^*(X)$ which minimizes the total loss. Therefore,

$$ f^*(X) = \arg\min_{f^*(X)} L_{total} $$

(2.23)

In order to find the optimal function $f^*(X)$ we need to solve for Equation 2.22. However, since the distribution $P(X,Y)$ is unknown, computing Equation 2.22 is not possible. Instead, an empirical loss over the training data is computed. Using the Empirical Risk Minimization framework [152], if the number of training instances is $m$, the empirical loss can be written as:

$$ \hat{L}_{total} = \sum_{i=1}^{m} L_{per\_instance}(f(X_i), Y_i) $$

(2.24)

Within this setting, during learning we want to find the function $f^*(X)$ which minimizes the total empirical loss:

$$ f^*(X) = \arg\min_{f(X)} \hat{L}_{total} $$

(2.25)

We now need to make two primary choices: (1) what should be the definition of the loss function $L_{per\_instance}(f(X_i), Y_i)$, and (2) how can we learn $f^*(X)$ using the loss function. Regarding the first question mentioned above, an example of a perfect loss function is:

$$ L_{per\_instance}(f(X_i), Y_i) = 1 - NDCG(q_i; f) $$

(2.26)

As we have discussed in Section 2.4.1 that NDCG is either flat or discontinuous everywhere with respect to the parameters of the learning algorithm, so is $1 - NDCG$. That is why using Equation 2.26 as a loss function in order to get the optimal ranking function $f^*(X)$ is not a feasible choice. Hence practitioners use surrogate loss functions which are relatively easy to optimize, thereby giving rise to different types of algorithms such as pointwise, pairwise etc.

As for the second question, i.e., given a loss function how we can learn $f^*(X)$, different learning models such as support vector machines, neural networks etc. are used.

This wraps up the general discussion on LtR that is required to understand the rest of the thesis - we have discussed the evolution of LtR methods, different categories of LtR algorithms and the theoretical foundation of the LtR problem. The next section expounds as to why we have chosen a random forest to tackle the LtR problem.
2.7 Why Random Forest

In this section we first discuss the potential benefits of a random forest over many other methods. We then highlight the fact that in spite of having several appealing characteristics, the use of random forests for solving the LtR problem has not been, unlike many other methods, thoroughly investigated so far – and thus we justify our intention to investigate RF-based LtR algorithms.

2.7.1 Advantages of Random Forests

Some benefits of a random forest over many other state-of-the-art methods are given below.

1. It can capture complex interactions amongst features by learning a non-linear combination of them.
2. It can work equally well with continuous, discrete or even missing values with no/little modification.
3. It is relatively robust to outliers, i.e., noisy examples, and also to noisy labels.
4. It is completely parallelizable because the trees are independent of each other.
5. Very few parameters must be tuned (if at all).
6. No normalization of the features is needed.
7. It is less affected by uninformative features.
8. It has an embedded mechanism to select the most useful features.

As for the performance of a random forest, it is, despite having conceptual simplicity, comparable to or quite often better than state-of-the-art algorithms ([69, Ch. 15], [7, 24, 32, 48]). To mention a few comments by notable researchers, Hastie et al. [69, Ch. 15] write: “In our experience random forests do remarkably well, with very little tuning required.” According to Biau [11], “random forests have emerged as serious competitors to state-of-the-art methods such as boosting and support vector machines”. Similar comments can be found in other literature.

A limitation of the random forests is, unlike a single decision tree, an actionable plan is not directly evident given an instance. That is, the outcome of the model is not directly
human-interpretable. Eliciting a human-interpretable solution is useful in different domains such as medical research. However, research is emerging to mitigate this drawback (e.g. [35]).

We now briefly discuss the comparison between the random forest and boosting which is another additive tree ensemble framework [50, 52]. Boosting methods have been shown to be highly effective for different tasks in the literature [69, Ch. 10]. Specifically, in LtR, boosting-based algorithms have been found to be amongst the most effective algorithms (as discussed in the Section 2.4.1). However, our motivation to conduct research on random forest is driven by the following two reasons: (1) in general a random forest has some potential benefits over boosting (as will be explained next), and (2) boosting methods have already been thoroughly investigated for LtR whereas the use of random forest in this discipline is still quite under-explored.

In the literature, most of the comparisons between the random forest and boosting are based on particular tasks or particular datasets (e.g. [113]). A few works which conduct extensive empirical experiments (with tens of algorithms and tens of datasets) are Caruana et al. [24], Banfield et al. [7] and Fernandez et al. [48]. Based on these and other works, below we describe the potential benefits of a random forest over boosting.

1. **Performance.** Performance of random forests has been found to be quite competitive to boosting [7], [69, Ch. 15]. Caruana et al. [24] perform extensive empirical comparison between a good number of learning algorithms, and conclude that the boosted tree ensemble and the random forest are the two top performers in general. Their findings also reveal that both of these methods perform quite well in terms of the rank-related metrics such as average precision, which makes them strong candidate for rank-learning task. Fernandez et al. [48] conduct probably the most extensive comparison ever with 179 classifiers and 121 datasets, and summarize their findings by saying: “The classifiers most likely to be the bests are the random forests”.

2. **Ease of use.** A random forest is simpler to train and tune [69, Ch. 15]. For many state-of-the-art LtR algorithms, tuning of (hyper-)parameters becomes a burden mainly due to the inherent large size of the data.\(^5\) On this note, Ganjisaffar et al. [56] have to resort to the MapReduce framework to select effective parameter settings for their boosting-based LtR algorithm; despite this, they still warn that overusing the methodology of tuning the parameters may cause overfitting. In contrast, a random forest is built not to overfit [15], [69, Ch. 10], [141] – usually

---

\(^5\)We have already explained in Section 1.1 that the less tuning of LtR algorithms is desirable even to commercial industries.
one can increase the number of unfettered trees (in terms of depth) in an ensemble without any concern of overfitting. Caruana et al. [24] report from their experiments that a random forest is more stable across various datasets as an off-the-shelf learning algorithm than boosting, as they find the boosting to be more susceptible to overfitting.

3. **Scalability.** A random forest is embarrassingly parallelizable in the sense that trees of an ensemble can be learnt independently of each other. Hence the random forest based LtR algorithms have an inherent potential to be useful for large scale LtR. Scalability is an emerging topic in LtR research [26], [90, Ch. 7], [94, Ch. 20]. In boosting, each tree is learnt based on the previous trees of the ensemble, so employing parallelization is difficult. As such, Mohan et al. [108] and Breiman [15] note that the training phase of a random forest is usually computationally faster than boosting when the number of trees is kept fixed. Moreover, there are even faster (modified) versions of a random forest such as extremely randomized trees [62] and random ferns [116].

To complement the comparison, the benefits of boosting over random forests are given below.

1. From a theoretical perspective, a random forest is comparatively less studied than boosting. However, recently Biau [11], Wager [154], Scornet [137], Scornet et al. [138] (and the references therein) have notably contributed to this field. As for bagging which can be thought as a predecessor of the random forest, Bühlmann and Yu [16] theoretically explain its working mechanism. Steinber and Cardell [146] establish that a single tree of a random forest is a form of \( k \)-nearest neighbour classifier which is one of the oldest and theoretically most robust machine learning technologies.

2. If the trees of a boosted ensemble of trees are kept significantly smaller (which is usually the case because boosting is mainly a bias reduction technique whereas a random forest is mainly used to reduce variance), then boosting is likely to take less computational time during evaluation. However, as research on reducing the evaluation time of a generic tree ensemble based learning algorithms is emerging [4, 20, 96], this is becoming a lesser problem day by day. As for the training time, the amount of available computational resources decide which one needs less time – if these resources are plenty, then the random forest is likely to win because of its inherent parallelizability (for instance, this phenomenon is observed by Mohan et al. [109] during their experiments on large scale LtR).
It can be mentioned here that random forests have been found to be very effective, and hence quite popular in a variety of applications since its inception in 2001, such as bioinformatics [39], digital image processing [32], human computer interaction [42], geographical data analysis [64] etc.

2.7.2 Need for Further Research in Random Forest Based LtR

In a recent LtR challenge organized by Yahoo! [25], many of the top ranked participants used some form of randomized tree ensemble methods. (Geurts and Louppe [63] and Mohan et al. [109] are two of them.) The organizers [25], after their experience in the challenge, point out that the success of tree ensemble based algorithms may be attributed to the fact that these models can capture complex non-linear interactions among the features (which are oftentimes individually weak).

As explained in Section 1.1 that in order to improve the accuracy, scalability and robustness, it is imperative to conduct further research in LtR. In the previous sub-section we have discussed that the random forest enjoys several benefits over many other methods. We also discussed that the random forest based (relatively straightforward) LtR algorithms have already exhibited competitive performance to the state-of-the-art methods. In the literature significant research has been conducted on other state-of-the-art techniques in the context of LtR such as SVM, neural networks and boosting, which is not, however, the case with the random forest. Considering these facts, we believe it to be imperative to thoroughly investigate the random forest based LtR algorithms.

2.8 Scope of the Thesis

Regarding the scope of this research, the following points can be noted:

- Dynamic ranking (dependent on queries) is addressed, not static ranking (e.g. PageRank [117]).
- General document ranking are addressed with publicly available datasets.
- No query refinement is performed.
- No relevance feedback is assumed.
- Relevance labels of documents are assumed to be fixed, i.e., given a priori.
Chapter 2: Background, Literature Review and Experimental Setup

• No manipulation on feature values is performed, i.e., no feature normalization is performed except the default case of the datasets. No manipulation on missing values is conducted (in fact, the missing value issue is already dealt in the datasets we use).

• Features are treated as continuous, but they may be discrete.

• Only binary trees are considered.

• Axis-parallel split is considered in a tree.

2.9 Experimental Protocols

In this section we describe the datasets and experimental setup used in the thesis.

2.9.1 Datasets

We use several publicly available datasets. Table 2.2 shows their statistics. Further details can be found in Qin et al. [124], in Letor website\(^6\), in Microsoft Research website\(^7\), and in Chapelle et al. [25]. The datasets contain both the navigational (HP2004 and NP2004) and informational queries. All the datasets except the Yahoo are released with a pre-division of five folds. The features of all these datasets are mostly used in academia. However, features of the Yahoo dataset (which was published as part of a public challenge [25]) are not disclosed as these are used in a commercial search engine. This one comes, unlike the others, in a single fold, but pre-divided into a training, a validation, and a test set. For the sake of better compatibility with the existing literature, all the algorithms in this paper are trained using these pre-defined training sets.

Table 2.3 sorts the datasets based on various characteristics.

2.9.2 Notations

We have already listed the useful notations in the preamble of the thesis. Throughout the thesis, if the meaning is obvious, we sometimes omit the subscripts/superscripts.

\(^6\)http://research.microsoft.com/en-us/um/beijing/projects/letor/
\(^7\)http://research.microsoft.com/en-us/projects/mslr/
Table 2.2: Statistics of the datasets (sorted by # queries). In the last row, 973/15 for TD2004 means that there are 973 and 15 documents of label 0 and 1 respectively.

<table>
<thead>
<tr>
<th>Characteristic</th>
<th>TD2004</th>
<th>HP2004</th>
<th>NP2004</th>
<th>Ohsumed</th>
</tr>
</thead>
<tbody>
<tr>
<td>Task</td>
<td>Topic</td>
<td>Homepage</td>
<td>Named Page</td>
<td>Medical</td>
</tr>
<tr>
<td># Queries (overall)</td>
<td>75</td>
<td>75</td>
<td>75</td>
<td>106</td>
</tr>
<tr>
<td># Queries (train)</td>
<td>50</td>
<td>50</td>
<td>50</td>
<td>63</td>
</tr>
<tr>
<td># Features</td>
<td>64</td>
<td>64</td>
<td>64</td>
<td>45</td>
</tr>
<tr>
<td># Rel. labels</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td># Query-doc pairs (overall)</td>
<td>75000</td>
<td>75000</td>
<td>75000</td>
<td>16000</td>
</tr>
<tr>
<td># Query-doc pairs (train)</td>
<td>50000</td>
<td>50000</td>
<td>50000</td>
<td>9684</td>
</tr>
<tr>
<td># Docs per query</td>
<td>988</td>
<td>992</td>
<td>984</td>
<td>152</td>
</tr>
<tr>
<td># Docs of diff. labels (0/1/2/3/4) per query</td>
<td>973/15</td>
<td>991/1</td>
<td>983/1</td>
<td>106/24/21</td>
</tr>
</tbody>
</table>

Table 2.3: Datasets sorted by different properties. (MSLR-W stands for MSLR-WEB10K.)

<table>
<thead>
<tr>
<th>Task</th>
<th>Web</th>
<th>Web</th>
<th>Web</th>
<th>Web</th>
</tr>
</thead>
<tbody>
<tr>
<td># Queries (overall)</td>
<td>784</td>
<td>1692</td>
<td>10000</td>
<td>29921</td>
</tr>
<tr>
<td># Queries (train)</td>
<td>470</td>
<td>1015</td>
<td>6000</td>
<td>19944</td>
</tr>
<tr>
<td># Features</td>
<td>46</td>
<td>46</td>
<td>136</td>
<td>519</td>
</tr>
<tr>
<td># Rel. labels</td>
<td>3</td>
<td>3</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td># Query-doc pairs (overall)</td>
<td>15211</td>
<td>69623</td>
<td>1200192</td>
<td>709877</td>
</tr>
<tr>
<td># Query-doc pairs (train)</td>
<td>9630</td>
<td>42158</td>
<td>723412</td>
<td>473134</td>
</tr>
<tr>
<td># Docs per query</td>
<td>19</td>
<td>41</td>
<td>120</td>
<td>23</td>
</tr>
<tr>
<td># Docs of diff. labels (0/1/2/3/4) per query</td>
<td>15/2.5/1</td>
<td>30/8/2</td>
<td>42/39/16/2/0.9</td>
<td>6/8/7/2/0.4</td>
</tr>
</tbody>
</table>

2.9.3 Toolkits

We implement our algorithms on top of Weka machine learning toolkit. Sometimes third-party libraries and implementations of baseline algorithms were used which will be referenced in appropriate places.

2.9.4 Significance Test

For the sake of better compatibility with the existing LtR literature, scripts provided by the Letor repository are used to perform pairwise significance test. Values of the metrics in bold and italic and bold fonts denote that the performance difference (between the two systems at hand) is significant with p-value less than 0.01 and 0.05 respectively. Sometimes a † sign in front of a value indicates that the performance is significantly poorer than the baseline.

†http://www.cs.waikato.ac.nz/ml/weka/
2.9.5 Evaluation Metrics

For small to moderate sized data (HP2004, TD2004, NP2004, Ohsumed, MQ2008 and MQ2007) we employ two widely used rank-based metrics, namely NDCG@10 and MAP. For the two larger ones, we add another metric, namely ERR [28] since these two datasets are of our main interest. Details of these metrics have been discussed in Section 2.3.

We use the evaluation scripts provided by the Letor repository for datasets TD2004, HP2004, NP2004, Ohsumed, MQ2008 and MQ2007. The Yahoo dataset does not come with any script, so we implement the formulae given by the releaser of the data [25], and use them for both the larger datasets (MSLR-WEB10K and Yahoo). The reported NDCG@10 for a dataset is the average over all queries of a test set. If a query does not have any relevant document, we assume its NDCG to be 0.0.

Some details of our designed evaluation test-bed are given in Appendix A.1.

2.9.6 Default Parameter Settings

Throughout the experiments, the following settings are maintained for a random forest unless otherwise stated.

- Regarding tree size, we follow the practice of the vast majority of the existing literature which is, the only condition that prevents a leaf from being split is if no gain is found from doing so. We do not limit the tree size by a maximum height or minimum number of instances.

- The number of random features to select at each node (denoted by $K$) is set to $\log(#\text{features}) + 1$ (as advocated by Breiman [15]).

- As for the ensemble size, 500 trees are used for each ensemble.

- As for the sub-sample size per tree, query-level sub-sampling without replacement is used with two variants: (1) the default case: for each tree 63% queries are sampled (uniformly at random) without replacement, and (2) for larger datasets: in some cases, less than 63% queries are sampled (details will be provided in relevant places).

- When listwise splitting is used (in Chapters 5 and 6), nodes of a tree are explored using breadth-first enumeration.

Having laid out the background knowledge of LtR approach, the properties of the datasets and parameter settings, from the next chapter we dive deep into the techniques
to improve scalability, performance and understanding of the random forest based LtR algorithms.
Chapter 3

Handling the Imbalanced Training Data

In this chapter we investigate the imbalanced nature of LtR training sets, which generally contain very few relevant documents as compared to the non-relevant ones. Since the relevant documents are rare in a document collection, the need to include as many relevant documents as possible in the training set is well-established. However, the lower bound of the number of non-relevant documents needed to learn a good ranking function is still largely uninvestigated. This chapter aims at addressing this question. We employ both random and deterministic undersampling techniques to reduce the number of non-relevant documents. Minimization of training set size reduces the learning time which is an important factor, especially for large scale LtR. Extensive experiments on Letor benchmark datasets reveal that in many cases the performance of a LtR algorithm trained on a much smaller training set (in terms of presence of non-relevant documents) remains largely similar to that of a model learnt from the original training set. Thus this study suggests that for large scale LtR tasks, we can leverage undersampling techniques to reduce training time with oftentimes negligible effect on performance. We further examine the potential benefit of a random forest based LtR algorithms which demonstrates that the inherent structure of a random forest is conducive to using undersampling, and thereby yields even more scalability.

The rest of the chapter is organized as follows. Section 3.1 explains why do we need to investigate the skewed distribution of relevance labels in an LtR training set. Section 3.2 lists the major contributions of this investigation. Section 3.3 discusses existing works related to the problem. Section 3.4 illustrates the methodology of our approach.
Section 3.5 explains the model. Section 3.6 analyses the experimental results. Section 3.7 shows the efficacy of a random forest framework in yielding further scalability. Section 3.8 summarizes the findings.

3.1 Motivation

As briefly explained in Section 2.2, the task of developing an LtR-based IR system can be viewed as a two stage process [37, 94, 100]. The first stage involves the following steps:

1. **Top-\(k\) retrieval.** An initial retrieval approach (involving one or more base rankers such as BM25 score) is used to retrieve top-\(k\) documents for each query from the whole collection.

2. **Human labelling and feature extraction.** Relevance judgements for the retrieved \(k\) documents are collected usually from human judges [5, 120]. Also, features are extracted for each of these query-document pairs. The features are then normalised on a per query basis. These features along with the relevance labels constitute a training set.

The following step is then performed in the second stage:

3. **Learning.** An LtR algorithm is employed to learn a ranking function \(f(x)\) from the training set.

Once the system has been trained, the following steps are performed during real time evaluation:

1. **Top-\(k\) retrieval.** For a query submitted by the user, the same top-\(k\) retrieval approach (i.e., step 1 of the learning phase) is applied.

2. **Feature extraction.** Features are extracted for the retrieved documents.

3. **Application of the learnt model.** A relevance score for each document is generated using the learned model. The documents are then ranked using these scores, and the ranked list is returned to the user.

Figure 3.1 depicts the complete scenario. (Note that this figure is a finer grained description of Figure 2.2.)
Most of the existing work in LtR is focused on developing better algorithms for learning a ranking function given a training set, while relatively little research has been devoted to the out-of-the-box engineering topics such as how to improve the quality of the training data, or how to tackle the scalability issue for large scale LtR.

Some studies have been conducted on the first stage (i.e., how to improve the initial retrieval approach) of LtR which will be discussed in the Section 3.3. We are, however, interested in an aspect that lies between the first and second stages. After applying the initial approach to retrieve the top \( k \) documents from a document collection, the resultant training set usually contains very few relevant documents associated with a query as compared to the non-relevant ones. To demonstrate this characteristic, in Figure 3.2 we show the relevance label distribution of the LtR datasets introduced in Chapter 2. We see that for all eight datasets the discrepancy in label distribution exists.
Chapter 3: Handling the Imbalanced Training Data

Now the question becomes, why might imbalanced training data be a problem for LtR? Commercial IR systems have billions of documents in their collections. As such, a representative training set (found after applying the initial top-$k$ retrieval) is typically very large. It is, therefore, computationally challenging for a learning algorithm to learn a ranking function from such a large training set.\(^2\) Hence keeping the training set size smaller is lucrative from the perspective of scalability. On the other hand, the larger the training set, the more information is given to a learning algorithm, and hence sufficient amount of training data are needed to learn a good ranking function. Since the number of relevant documents is limited (while the non-relevant documents for a query is practically unlimited), all the relevant documents of a training set are considered as useful. Thus the following research question can be raised: are all of the non-relevant documents from top-$k$ retrieval (which are currently included in the LtR training sets) necessary to learn a good ranking function? That is to say, could a small subset of non-relevant documents be sufficient for learning a ranking function capable of distinguishing between relevant and non-relevant documents? If we find that we can use such a smaller subset without compromising the accuracy of the learning algorithm significantly, the time complexity for LtR algorithms (which is an important issue for large scale LtR \cite{26}, \cite{90, Ch. 7}, \cite{94, Ch. 20}) would be significantly reduced.

### 3.2 Contributions

The following contributions are made in this chapter:

- We employ both random and deterministic undersampling techniques to examine if a ranking function can successfully recognize patterns of relevant and non-relevant documents with a smaller (than usual) subset of non-relevant documents in the training set. The results suggest that random undersampling of non-relevant documents on a per-query basis works quite well. In many cases, much smaller training data (in some cases as little as 19-32\% of the training data) can be used without significant degradation of performance.

- While our investigation suggests that all LtR algorithms are likely to benefit from the undersampling techniques, the inherent structure of a random forest offers

---

\(^1\)Note that for graded relevance labels, the usual practice (e.g. \cite{109}) is to treat only some of the higher relevance labels as relevant while the rest as non-relevant. Specifically, for labels in the range 0-4, labels 3 and 4 would be considered as relevant, and for labels in the range 0-2, labels 1 and 2 can be considered as relevant.

\(^2\)As discussed in Section 1.1 that despite possessing huge computational resources, the commercial search engines are still in need of better scalability for their learning algorithms.
a more effective way to achieve this as compared to other LtR algorithms. By properly utilizing the tree-forest structure of an RF-based LtR algorithm we can achieve even more scalability without significant performance degradation.

3.3 Related Work

Aslam et al. [5] investigate different methods for top-\(k\) retrieval using a large corpus. That is, they study techniques for generating a better training set from a large unlabelled document collection. The training sets produced by their methods are, however, still highly imbalanced. Hence our intended work is complementary to theirs.

McDonald et al. [100] focus on the properties of a good training set through extensive empirical study on several large document collections. They examine the number of documents per query (i.e., different values of \(k\)) to be used in top-\(k\) retrieval stage, and empirically search for the optimal values of \(k\) for different tasks and datasets. Their conclusions include: retrieval performance in general increases with increasing size of training sample (i.e., the values of \(k\)) up to a certain point (depending on the datasets), and afterwards the performance plateaus.

Dang et al. [37] develop an improved initial retrieval method in the sense that it retrieves more relevant documents than the existing methods such as BM25. Their method utilizes some complex features like proximity based retrieval functions [8].

Long et al. [95] propose a technique based on Active Learning framework that includes an (unlabelled) example in the training set if this inclusion minimizes a ranking loss over the training set. The main motivation of using active learning is to reduce the large cost associated with manual labelling of documents. Their proposed method is compared with mainly the depth-\(k\) pooling method. Some related works such as Donmez et al. [43] and Yu [171] also try to find the (unlabelled) examples which, if added to a training set, increase the quality of the learned ranking function. This category of works do not differentiate between relevant and non-relevant documents, whereas our intended investigation is specifically concerned with the importance of non-relevant documents. That is, the training set generated by these methods is still likely to be highly imbalanced, and thus eligible for our investigation.

We emphasize the point that although one of the goals of most of the abovementioned works is to retrieve as many relevant documents as possible in a training set, in practice the disparity between the numbers of relevant and non-relevant documents is still high. As an example, in an attempt to increase the relevant documents, Chapelle and
Cheng [25] employ the depth-$k$ pooling method which uses more than one base ranker in the initial retrieval stage, but still get an imbalanced training set.

From our literature review we observe that none of the studies that investigate into improving the quality of the training data are primarily concerned with the non-relevant documents, i.e., none of them are concerned with the imbalanced nature of the training data. As our techniques are applicable after applying those methods (and before the learning phase), this work is distinctive and complementary to the existing works.

### 3.4 Approach

Our goal is to reduce the number of non-relevant documents which comprise the vast majority of the training set, and then to examine the effect of learning from the (reduced) training set on a separate (non-reduced) test set. Pictorially, Figure 3.3 describes the procedure. In order to achieve this goal we exploit the concept of undersampling techniques for imbalanced data from the machine learning literature [72]. Undersampling techniques aim at making the training set more balanced in terms of the number of instances from each class. In this study we investigate two approaches: (1) a random undersampling, and (2) a deterministic undersampling.

The existing literature on improving the classification accuracy in the presence of imbalanced data can broadly be divided into two categories [72]: (1) a data-level approach, also known as sampling techniques, and (2) an algorithmic-level approach, also sometimes called as cost-sensitive learning. In this work we investigate the undersampling techniques from the first category – we do not consider oversampling techniques (and related techniques such as SMOTE [29]) as these increase the training set size.

Since the LtR data are divided by queries, we perform undersampling at the query level. That is, for each query, we retain all the relevant documents in the new training set, and then include a subset of non-relevant documents according to a criterion (either randomly or using a deterministic technique). We then learn a model on this (reduced)
training set, and evaluate the model on a (non-reduced) test set to compare performance at different amounts of training data.

The procedure of random undersampling is, as the name implies, to randomly select a subset of non-relevant documents. As for the deterministic undersampling, our goal is to retain the most “informative” non-relevant instances (i.e., query-document pairs) so that the learning algorithm can effectively learn to distinguish between relevant and non-relevant documents with fewer non-relevant documents in the training set. We identify the informative non-relevant documents using an effective feature which is BM25 score. The hope is that the lower the BM25 score of an non-relevant document, the “more non-relevant” the document, and hence the more informative this document is (as an non-relevant document).

The configuration which uses the original training set is considered as the baseline.

3.5 Model

The focus of this thesis is the random forest based LtR algorithms. Below we describe the model which will also be used throughout the subsequent chapters.

A random forest [15] is a conceptually simple but effective and efficient learning algorithm that aggregates the outputs of a large number of independent and variant base learners, usually decision trees. It is an (usually bagged) ensemble of recursive partitions over the feature space, where each partition is assigned a particular class label (for classification) or a real number (for regression), and where the partitions are randomly chosen from a distribution biased toward those giving the best prediction accuracy. For classification the majority class within each partition in the training data is usually assigned to the partition, while for regression the average label is usually chosen, so as to minimise zero-one loss and mean squared error respectively.

In order to build the recursive partitions, a fixed-sized subset of the attributes is randomly chosen at each node of a tree, and then for each attribute the training data (at that node) is sorted according to the attribute and a list of potential split-points (midpoints between consecutive training datapoints) is enumerated to find the split which minimises expected loss over the child nodes. Finally the attribute and associated split-point with the minimal loss is chosen and the process is repeated recursively down the tree. For classification the entropy or gini function\(^3\) is used to calculate the loss for each split, while for regression the mean squared error is used.

\[^3\text{If } p(c|\text{leaf}) \text{ denotes the estimated probability of a class at a leaf then } \mathcal{L}_{\text{entropy}}(\text{leaf}) = -\sum_c p(c|\text{leaf}) \log(p(c|\text{leaf})) \text{ and } \mathcal{L}_{\text{gini}}(\text{leaf}) = \sum_c p(c|\text{leaf})(1 - p(c|\text{leaf}))\]
Procedure: \textit{BuildTree}(\mathcal{D})

Data: Dataset \mathcal{D}

Result: A tree root

\begin{algorithmic}
\State \textbf{begin}
\State \quad root \leftarrow \text{createNode}(\mathcal{D});
\State \quad nodeList \leftarrow \{\text{root}\};
\While{\mid \text{nodeList} \mid > 0}
\State \quad node \leftarrow \text{nodeList}[1];
\State \quad nodeList \leftarrow \text{nodeList[1]} \setminus \{\text{node}\};
\If{\mid \mathcal{D}_{\text{node}} \mid > 1}
\State \quad \text{best} \leftarrow (0, \text{null}, \text{null});
\For{\text{feature} \in \text{randomSubset}\left(\{1, \ldots, M\}, \log M\)}
\For{\text{split} \in \text{midpoints}(\text{sort}(\mathcal{D}_{\text{node}}, \text{feature}))}
\State \quad \mathcal{D}_{\text{left}} \leftarrow \{\vec{x} \in \mathcal{D}_{\text{node}} | x_{\text{feature}} < \text{split}\};
\State \quad \mathcal{D}_{\text{right}} \leftarrow \{\vec{x} \in \mathcal{D}_{\text{node}} | x_{\text{feature}} \geq \text{split}\};
\State \quad \text{gain} \leftarrow \text{Gain}(\mathcal{D}_{\text{node}}, \mathcal{D}_{\text{left}}, \mathcal{D}_{\text{right}});
\EndFor
\EndFor
\If{\text{gain} > \text{best.gain}}
\State \quad \text{best} \leftarrow (\text{gain}, \text{feature}, \text{split});
\EndIf
\EndIf
\If{\text{best.gain} > 0}
\State \quad \text{leftChild} \leftarrow \text{createNode}(\vec{x} \in \mathcal{D}_{\text{node}} | x_{\text{best.feature}} < \text{best.split});
\State \quad \text{rightChild} \leftarrow \text{createNode}(\vec{x} \in \mathcal{D}_{\text{node}} | x_{\text{best.feature}} \geq \text{best.split});
\State \quad \text{nodeList.add}(\{\text{node.leftChild}, \text{node.rightChild}\});
\EndIf
\EndWhile
\State \textbf{return} \text{root};
\State \textbf{end}
\end{algorithmic}

Procedure: \textit{Gain\textsubscript{entropy}}(\mathcal{D}, \mathcal{D}_{\text{left}}, \mathcal{D}_{\text{right}})

Data: Data at node and left/right child: \mathcal{D}, \mathcal{D}_{\text{left}}, \mathcal{D}_{\text{right}}

Result: Change in Entropy resulting from split.

\begin{algorithmic}
\State \textbf{begin}
\State \quad \text{return } \text{Entropy}(\mathcal{D}) - \left( \frac{|\mathcal{D}_{\text{left}}|}{|\mathcal{D}|} \text{Entropy}(\mathcal{D}_{\text{left}}) + \frac{|\mathcal{D}_{\text{right}}|}{|\mathcal{D}|} \text{Entropy}(\mathcal{D}_{\text{right}}) \right);
\State \textbf{end}
\end{algorithmic}

Where the entropy of data \mathcal{D} over \( C \) class labels (with \( \mathcal{D}_c \subseteq \mathcal{D} \) denoting examples of \( c \)th class) is computed as: \( \text{Entropy}(\mathcal{D}) = -\sum_{c=0}^{C-1} \frac{|\mathcal{D}_c|}{|\mathcal{D}|} \log \frac{|\mathcal{D}_c|}{|\mathcal{D}|} \).

\textbf{Algorithm 1:} Generic tree building procedure for a random forest.

Since it has been proven that the ranking error is bounded by both the classification error [92] and regression error [31], practitioners oftentimes address the LtR problem using a classification or a regression model. In this setting, by treating the relevance judgements of documents as the target variable, a classification (or regression) algorithm learns to predict the relevance label of an individual query-document pair. During evaluation, the documents associated with a query are ranked in decreasing order of their predicted relevance scores. This approach is called \textit{pointwise} in the literature [90] because it treats the instances (i.e. feature vectors corresponding to the query-document pairs) independently from one another even if two documents are associated with the same query. The benefits of this approach include lower computational resource requirement and conceptual simplicity.

Inspired by a successful adaptation by Geurts and Louppe [63], the LtR algorithm that we employ in this chapter can be interpreted as a blend of the classification and regression settings. To split a node of a tree, a classification setting (entropy-based gain)
is used. To assign a label to a data partition (i.e., during evaluation of a test instance), a regression setting is used as follows: when the test instance lands in a data partition (i.e., a leaf node), unlike classification where the mode of different class labels is assigned as the predicted label, the algorithm assigns (as the score to the test instance) the average of the relevance labels of all the documents of that data partition. Finally, in order to produce a ranked list, the documents are sorted in decreasing order of the predicted scores. Thus the training and the testing phase adopts a classification and a regression setting respectively. The procedure for building a tree is given in Algorithm 1. Each tree is learnt using a bootstrapped (without replacement) sample of the training set, and the bootstrapping is performed on a per-query basis. We call this algorithm RF-point.

3.6 Experiments

This section describes the experimental settings and analyses the results.

3.6.1 Setup

As rank learners, in addition to the RF-point algorithm discussed in the previous section, we employ another rank-learner called RankSVM which is a popular pairwise LtR algorithm that formalizes LtR as a problem of binary classification on instance pairs, and solves the problem using support vector machines (SVM).

As evaluation metrics, we use two widely known measures, namely NDCG@10 and MAP. We use eight different datasets from a variety of tasks: domain-specific search (Ohsumed), topic distillation (TD2004), homepage finding (HP2004), named page finding (NP2004), and general web search (MQ2007, MQ2008, MSLR-WEB10K and Yahoo).

3.6.2 Using RF-point Algorithm

This section discusses results of RF-point algorithm with random and deterministic undersampling approaches.

---

4While in the literature most of the implementations of a tree uses a depth-first (i.e., recursive) exploration of the nodes, the implementation shown here uses a breadth-first exploration mainly because we shall use this version in some later chapters. For an entropy-based objective function, the node exploration strategy does not affect the tree structure, i.e., the data partitions. More on this will be discussed in Chapter 5, in particular, in Section 5.4.3.

Table 3.1: Random undersampling with RF-point: % of training data required (per ensemble) to achieve a given percentage of performance of baseline NDCG@10 (i.e., with 100% training data) along with training time improvement.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>% Rel. Docs</th>
<th>% of training set required (per ensemble) for 98%-99% of baseline performance</th>
<th>Random undersampling</th>
<th>Training time improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ohsumed</td>
<td>30</td>
<td>37 - 56 - 98</td>
<td>2.8 - 1.8 - 1 times</td>
<td></td>
</tr>
<tr>
<td>MQ2007</td>
<td>26</td>
<td>30.7 - 33.1 - 40.3</td>
<td>3.5 - 3.2 - 2.6 times</td>
<td></td>
</tr>
<tr>
<td>MQ2008</td>
<td>19</td>
<td>24.5 - 24.5 - 95.2</td>
<td>3 - 3 - 1 times</td>
<td></td>
</tr>
<tr>
<td>TD2004</td>
<td>1.5</td>
<td>4.5 - 10.6 - 21.7</td>
<td>13 - 10 - 6 times</td>
<td></td>
</tr>
<tr>
<td>HP2004</td>
<td>0.3</td>
<td>0.31 - 7.2 - 32.4</td>
<td>50 - 18 - 4 times</td>
<td></td>
</tr>
<tr>
<td>NP2004</td>
<td>0.1</td>
<td>0.82 - 4.2 - 24.4</td>
<td>40 - 21 - 6 times</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>% Rel. Docs</th>
<th>% of training set required (per ensemble) for 98%-99% of baseline performance</th>
<th>Random undersampling</th>
<th>Training time improvement</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSLR-WEB10K</td>
<td>3</td>
<td>15-19</td>
<td>11 - 8 times</td>
<td></td>
</tr>
<tr>
<td>Yahoo</td>
<td>12</td>
<td>22-100</td>
<td>7 - 1 times</td>
<td></td>
</tr>
</tbody>
</table>

3.6.2.1 Random Undersampling

Figure 3.4 shows plots of random undersampling approach for the eight datasets.

We first discuss results of the six smaller datasets. Along the x-axis of a plot is the percentage of training set used for learning, and along the y-axis is performance. To suppress random fluctuations, each point in a plot corresponds to the average metric of five independent runs. We observe that initially, i.e., with very few non-relevant documents, performance is comparatively poor, as expected; however, the level of degradation depends on the dataset. As the number of non-relevant document increases, so does the performance. In general, after the initial increase, relatively minor improvement in
performance is gained as we increase the number of non-relevant documents (note the restricted interval on the y-axis); and this phenomenon is pronounced in the datasets where the distribution of the relevance labels is highly skewed, namely in TD2004, HP2004 and NP2004.

Table 3.1 shows the percentage of original training set required to reach a given level of performance. We observe that the percentage of training set required to achieve close performance to the baseline varies from task to task. The TD2004, HP2004 and NP2004 datasets benefit most from undersampling – only 10.6%, 7.2% and 4.2% training data are required to reach within 98% of baseline performance respectively. Ohsumed, MQ2007 and MQ2008 datasets are moderately benefitted due to the comparatively less imbalance nature of their data – approximately half, one-third and one-fourth of the original data is needed for these three datasets to reach within 98% of the baseline performance respectively. As for the training time, the TD2004, HP2004 and NP2004 datasets achieve 10-21 times gain over the baseline.

For the two larger datasets, namely MSLR-WEB10K and Yahoo, we report results of one run (instead of five) as these datasets are sufficiently large to suppress the possible fluctuation in the plots (indeed, their plots are found to be comparatively smoother). Also, for better comparison with the baseline, here we focus on the amount of training data required to reach within 98% and 99% of the baseline performance. From the plots we observe that the undersampling approach works better for MSLR-WEB10K dataset than for Yahoo dataset because the relevance label distribution is more skewed in the former case than the latter (cf. Table 2.2). On the MSLR-WEB10K dataset, approximately one-fifth of the data is required to reach within 99% of baseline performance (with 8 times smaller training time), whereas on the Yahoo dataset an increased amount of non-relevant documents is found to be always useful.

This sub-section thus reveals that when RF-point employs the random undersampling approach, both the smaller and larger datasets are candidates for getting improved training time given that their relevance label distributions are sufficiently skewed.

Figure 3.5 shows that for all the datasets the training time increases linearly with increasing training set size.\(^6\)

### 3.6.2.2 Deterministic Undersampling

Our second approach performs a deterministic selection of non-relevant documents. The procedure is as follows. For a query, we, as before, include all the relevant documents

\(^6\)The minor fluctuations are most likely to be due to the fact that the machine running the experiments were being used for running other programs simultaneously.
in the training set. We then sort the non-relevant documents in ascending order of the values of an effective individual ranker, namely the BM25 scorer. In the first configuration, we thus include only the first non-relevant document (i.e., with the minimum feature value) for that query to the training set. For subsequent configurations, additional non-relevant documents are included in the previously sorted order. We call this approach the ascending order approach.

Figure 3.6 shows the plots for the ascending order approach. All the datasets, in contrast to the random undersampling approach, exhibit an increasing trend up to the last point of a curve. A possible explanation for rising performance up to the last configuration is as follows. When we continue to include non-relevant documents having comparatively larger BM25 scores, the performance still continues to slightly increase because even though BM25 is a good ranker, there do exist some non-relevant documents which have high BM25 scores, and including these documents in the training set helps the learning algorithm to learn these patterns in addition to the normal patterns (i.e., low BM25 scores) of the non-relevant documents. Another observation is that for most of the datasets for the very initial points of a curve there seems to be no systematic trend of performance. This is possibly due to the fact that the missing values of the features have been replaced by 0.0 by the dataset providers; so when we include the initial documents in ascending order, the documents with missing values are included which act as “noise” to our intended goal (i.e., to include documents with lower BM25 scores).
Once the documents having genuine BM25 scores start to be included, the systematic trends appear.

### 3.6.2.3 Comparison between Two Approaches

In Figure 3.7 we compare the performance of the two undersampling approaches investigated. We notice that in terms of requirement of percentage of training set to achieve close performance to the baseline, the random undersampling wins almost consistently over the deterministic one (with an exception of HP2004). This behavior is likely to be due to the fact that the random sampling approach preserves the original distribution of non-relevant documents, and as explained during the analysis of ascending order approach that although the importance of non-relevant documents having lower feature values are perceived, the higher feature values are also required to enhance the hypothesis space from which the ranking model is picked.

We reiterate that our results are not directly comparable with that of existing methods (as explained in Section 3.3) since we investigate a previously unexplored area of undersampling only the non-relevant documents, in the post-labelling phase, whereas the existing works focus on sampling both relevant and non-relevant documents, before labelling.
Chapter 3: Handling the Imbalanced Training Data

3.6.3 Using RankSVM Algorithm

In order to claim that the findings from RF-point are generalizable to any rank-learner, we conduct an experiment with the RankSVM algorithm. Figure 3.8 shows the results for random undersampling approach. We see that the trends found in RF-point algorithm are mostly observed in the plots of RankSVM as well. This indicates that the findings emerged from our technique of undersampling are most likely to be invariant to different types of rank-learning algorithms.
Chapter 3: Handling the Imbalanced Training Data

3.7 A More Scalable Undersampling with Random Forest

In the previous part of this chapter we have shown that for imbalanced datasets we can select a random subset of non-relevant documents (per query) without significantly degrading the performance. In this section we show that the architecture of a random forest offers a better way to achieve this goal.

3.7.1 Approach

Since a random forest is an aggregation of many independent base learners, our idea is to exploit undersampling at the level of base learners instead of considering the learner (i.e., the ensemble) as a black-box - this way the effect of information loss due to bootstrapping may be minimized. Recall that our standard approach to undersampling (cf. Figure 3.3) was to undersample a subset of non-relevant documents per query, and thus to generate a new (smaller) training set for an RF (or any other rank-learner). We henceforth call this approach the ensemble-level-undersampling (ELU). In this section, we investigate another approach that performs undersampling for each individual tree of an RF. Elaborately, to learn a tree, the first step is as usual: to generate a bootstrapped sample. After that we impose a second step which retains only a random subset (of pre-defined cardinality) of non-relevant documents (per query) from the bootstrapped sample along with all the relevant documents. Pictorially, Figure 3.9 describes the new approach which we term as the tree-level-undersampling (TLU). Thus unlike the ELU approach, the TLU approach does not need to exclude any non-relevant documents altogether from learning, rather each of them gets a chance to be used in some trees (given
Table 3.2: Comparison between standard random undersampling (i.e., ensemble level) and tree level random undersampling with RF-point: % of training data required (per tree) to achieve a given percent of baseline (i.e., 63% training set) performance (NDCG@10) along with training time improvement. The best value is in *bold and italic* font.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>% of training set required (per tree) for 95%-98%-99% of baseline (i.e., 63% sample) performance</th>
<th>Ensemble level</th>
<th>Tree level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ohsumed</td>
<td>23 - 35 -62</td>
<td>13 - 14 - 19</td>
<td>17.8 - 19.3 - 25.4</td>
</tr>
<tr>
<td>MQ2007</td>
<td>19.3 - 20.9 - 25.4</td>
<td>15.4 - 15.4 - 18.6</td>
<td>2.2 - 4.0 - 7.2</td>
</tr>
<tr>
<td>MQ2008</td>
<td>15.4 - 15.4 - 60.0</td>
<td>2.2 - 4.0 - 7.2</td>
<td>0.19 - 5.7 - 17.6</td>
</tr>
<tr>
<td>TD2004</td>
<td>2.9 - 6.7 - 13.6</td>
<td>0.32 - 3.8 - 30.2</td>
<td>0.32 - 3.8 - 30.2</td>
</tr>
<tr>
<td>HP2004</td>
<td>0.2 - 4.5 - 20.4</td>
<td>0.19 - 5.7 - 17.6</td>
<td>0.32 - 3.8 - 30.2</td>
</tr>
<tr>
<td>NP2004</td>
<td>0.51 - 2.6 - 15.4</td>
<td>0.32 - 3.8 - 30.2</td>
<td>0.32 - 3.8 - 30.2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Dataset</th>
<th>% of training set required (per tree) for 98%-99% of baseline (i.e., 63% sample) performance</th>
<th>Ensemble level</th>
<th>Tree level</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSLR-WEB10K</td>
<td>9 - 12</td>
<td>6 - 9</td>
<td>6 - 9</td>
</tr>
<tr>
<td>Yahoo</td>
<td>14 - 63</td>
<td>11 - 45</td>
<td>11 - 45</td>
</tr>
</tbody>
</table>

The ensemble is sufficiently large. We conjecture that the TLU approach will converge to the baseline (i.e., without using any undersampling) performance more quickly than the ELU approach.

Chen et al. [30] study some modifications of a random forest to tackle the class imbalance problem for classification. Since we are concerned with undersampling, below we summarize one of their techniques which is relevant to ours. For a tree, their method firstly uses a bootstrap sample of only the minority class examples, and after that it randomly samples a subset of majority class examples whose cardinality is equal to the number of minority class instances of the bootstrapped sample. However, this idea is not likely to be effective in the context of LtR because we need to perform sampling on a query-level, and in some datasets there is only one relevant document (i.e., minority class example) per query. Hence we do not consider their approach further.

### 3.7.2 Result Analysis

Table 3.2 shows the comparison between the ELU and TLU approaches. We see that as conjectured, on most of the datasets TLU approach achieves a given level of performance with smaller training data per tree (which in turn further reduces training time) as compared to the ELU approach. Using the data from Table 3.2, Figure 3.10 highlights this comparison for the 98% performance level.

We note that for most of the datasets we found a good number of cases where the TLU approach (with smaller data per tree) performs even better than the baseline (i.e., RF with no undersampling at all). The reason for this phenomenon is, we conjecture, due to the fact that the setting of the TLU approach reduces the correlation among the trees
which is favorable to getting better performance. Inspired by this finding, in the next chapter we investigate the correlation issue of RF in details.

3.8 Discussion

Training time of the LtR algorithms is considered to be an important issue [26], [90, Ch. 7], [94, Ch. 20]. Undersampling techniques reduce the training set size thereby decreasing the training time. Also, smaller training set requires smaller feature computation time, and some features are computationally costly to calculate [26, 100, 118], [94, Ch. 10].

Our investigation is distinct from and complementary to the existing works on using a better initial retrieval approach (i.e., the top \( k \) retrieval as explained in Section 3.1) in that our techniques are applicable after using these methods. The existing works focus on the quality of the training set by treating both relevant and non-relevant documents alike and before labelling the documents, whereas we focus on the training set size (and thus on learning time) in the context of the necessity of including large number of non-relevant documents. Moreover, our methods can also be extended to focus on the quality of the training set by incorporating more advanced undersampling methods in our framework so as to select a more effective subset of non-relevant documents for inclusion in the training set.

The observations of our investigation are summarized below:

- The findings suggest that in many cases much less training data (as little as 19-32% of the training set for TD2004, NP2004, HP2004 and MSLR-WEB10K datasets) can be used without significant degradation of performance, namely retaining at least 99% of the baseline performance. The findings also suggest that our method is more applicable to the tasks where the training data are highly imbalanced.
• The random undersampling approach which preserves the true distribution of non-relevant documents in the undersampled training set appears to be better than a deterministic undersampling which selectively includes the non-relevant document.

• The tree-forest hierarchy of a random forest is more conducive to undersampling. When properly used, this structure further increases scalability (cf. Figure 3.10) in the sense that the same level of performance is achieved by using less amount of training data.

• Applying sampling techniques take only linear time in terms of the size of the training set.

• The datasets for which we have found considerable gain in training time (e.g. TD2004, HP2004 and NP2004) are not in the same scale of size of that used in commercial IR systems. The bigger the datasets (having highly skewed relevance label distribution), the more positive effect is likely to emerge from our technique. Result of one of the two big datasets, namely MSLR-WEB10K corroborates this conjecture – in spite of containing relatively more relevant documents than the abovementioned three datasets, here the undersampling approach worked quite well (cf. Table 3.1).

• Our approach is applicable after the relevance judgements of the documents are labelled. Hence a natural concern is, our approach does not minimize that large cost associated with human labelling. As such, this investigation is more useful where click-through data are available so that methods for automatic labelling (such as [164]) can be used.

In the literature of classification, oversampling of minority class examples is a closely related topic to undersampling. However, this increases the learning time, and the learning time is the main focus of this chapter. Yet for the sake of comprehensiveness of our investigation, we performed oversampling on six datasets (namely MQ2007, MQ2008, Ohsumed, TD2004, HP2004 and NP2004) by randomly duplicating the minority class instances until their presence becomes equal to that of the majority class instances. We also implemented a query-level version of it. After applying RF-point with these two settings, we mostly found slightly poorer performance than the baseline. We show these results in Appendix B.1.

3.9 Conclusion

The core motivation of this research was to investigate a particular characteristic of an LtR environment - namely the relevance label distribution of a training set - given a
limited budget of computational resources, so that an IR system developer can take a more informed decision about the preparation of the training set. We have employed undersampling techniques to reduce training set size in order to achieve better scalability. This investigation reveals that for highly imbalanced datasets the efficacy of our techniques is evident. We have also successfully utilized the hierarchial tree-forest structure of a random forest to perform undersampling that resulted in better even scalability. Our work can be further extended by using more advanced undersampling methods.

Inspired by the findings of TLU approach, in the next chapter we devote more effort to better understand the effect of correlation between the trees of RF based rank-learners.
Chapter 4

Reducing Correlation Between the Trees

In the previous chapter we have investigated the efficacy of undersampling techniques in the LtR domain with an emphasis on a random forest-based rank-learner. The main goal there was to reduce learning time. Also, for the random forest based rank-learners we devised an effective approach that performs undersampling at the tree level instead of at ensemble level. In this setting, however, we have not considered the issue of correlation between the trees which is likely to have an effect in performance of a random forest. This raises a new question: how can we leverage the correlation between the trees in favor of performance and scalability of an RF-based LtR algorithm? This chapter investigates this question.

Traditionally each tree of a random forest is learnt using a bootstrapped copy of the training set, where approximately 63% of the examples are unique (although some studies show that sampling without replacement also works well). The goal of using a bootstrapped copy instead of the original training set is to reduce the correlation between individual trees, thereby making the prediction of the ensemble more accurate. In this chapter, we investigate whether we can further decrease the correlation between the trees while maintaining or possibly even improving accuracy. Among several potential options, we investigate the size of the sub-samples used for learning individual trees. We examine the performance of a random forest based LtR algorithm as we control the correlation using this parameter. Experiments on LtR datasets reveal that for small to moderate sized datasets, substantial reduction in training time can be achieved using only a small amount of training data per tree. Moreover, due to the positive correlation between the variability across the trees and performance of a random forest, we can increase accuracy while maintaining the same level of model stability as the baseline. For
big datasets, although our experiments did not observe increase in accuracy (because with larger datasets the individual tree variance is already comparatively smaller), our technique is still applicable as it allows for greater scalability.

The rest of the chapter is organized as follows. Section 4.1 discusses why it is important to investigate the research problem at hand. Section 4.2 lists the research contributions of this chapter. Existing works related to our topic are discussed in Section 4.3. The methodology is described in Section 4.4, which is followed by result analysis in Section 4.5. Section 4.6 further analyses a few discrete topics related to our approach. Section 4.7 provides a theoretical explanation of our technique. Finally, Section 4.8 summarizes the chapter.

### 4.1 Motivation

Bootstrapping is a well known and effective technique in statistical learning [46], and is an inherent component of a random forest. A bootstrapped sample is generated by randomly choosing an instance with replacement from the original training set, and this selection is repeated $N$ times where $N$ is the size of the original sample. The benefits of bootstrapping include: (i) it has sound mathematical properties [69, Ch. 8], and (ii) for ensemble learning, proper use of it reduces model variance.\(^1\)

A random forest has two components which affect its error rate [15]: (1) the correlation between the trees: the lower, the better, and (2) the strength of individual trees: the higher, the better. Usually there is a trade-off between the two.

The main advantage of aggregating predictions of multiple independent learners (as done in a random forest) is that the model variance is reduced. This means that different learners learn different aspects of the training set, and hence their predictions are aggregated, a more correct decision is expected to emerge. In a random forest, if each base learner, i.e., a tree (which is itself a stochastic learner due to the random selection of features at each node) of an ensemble learns from the entire training set, then there will be high correlation between them. This implies that their predictions will tend to be comparatively similar to each other, thereby reducing the advantage of aggregating the predictions. The goal of using bootstrapping in a random forest is to decrease the correlation (i.e., to increase the variability) between individual trees so that the ensemble is less sensitive to peculiarities of the training set. This is achieved by learning individual trees using a perturbed training sample. At the same time, each tree should be individually effective so that it is able to make good predictions on unseen data. Using

\(^1\)Bootstrapping is more effective in an ensemble of non-linear models rather than in linear models [69, Ch. 15]
a very small training sample for each tree will not yield effective per-tree performance as it will limit the hypothesis space,\(^2\) although it will increase variability. This raises the question: is there any specific benefit of using approximately 63% distinct training examples as is the case for bootstrapping? That is to say, can we decrease correlation by sampling less than 63% data (per tree) in favor of performance? Put differently, what is the optimal sub-sample size in the context of LtR? To the best of our knowledge, in the context of LtR no study has addressed this question, and hence we undertake this investigation.

As such, in this chapter our goal is twofold: (1) to reduce correlation between the trees with a hope that this will increase performance of the ensemble, and (2) to increase scalability while maintaining or improving the level of accuracy.

### 4.2 Contributions

The contributions of this chapter are as follows.

- We modify the traditional practice of sub-sampling in an RF-based rank-learning algorithm in favor of better performance and scalability. We demonstrate that on smaller datasets much smaller sub-samples (for some datasets, as little as 1.6% of the original training set) not only works well, but also increases the accuracy while maintaining similar level of model stability (in terms of variance across multiple runs).

- We examine the effect of ensemble size on our sub-sampling approach, and discuss the relation between strength of individual trees and the sub-sampling method.

- By formalizing and then examining the amount of correlation between the trees as we vary sub-sample size (per tree), we explain some of the theoretical underpinnings of the sub-sampling approach.

### 4.3 Related Work

In the literature a number of options to reduce the correlation have been suggested in the context of classification and regression problems. Below we discuss some of them.

Robnik-Šikonja [131] uses five different gain functions instead of a single one to decrease correlation between the trees for classification and regression tasks. With each of the five

\(^2\)With increasing data, a tree, being a non-parametric method, learns an increasingly complex hypothesis.
functions, his method learns one-fifth of the trees of an ensemble. The author reports minor improvement in accuracy.

Geurts and Louppe [62] inject additional randomness in a standard random forest through the selection procedure of the best split-point. They consider only one randomly chosen value for an attribute, and the entire training set is used to learn each tree (to decrease bias) thereby slightly mitigating the perceived benefit of scalability. In a later study [63], the same authors successfully apply this algorithm to LtR problem.

Another supervised learning framework which sometimes use bootstrapping is gradient boosting [52]. Ganjisaffar et al. [55] show that bootstrapping without replacement works well in their gradient boosting based LtR algorithm. As the for other supervised learning tasks such as classification and regression, Friedman [53] shows in the context of regression that the individual trees of a gradient boosted ensemble can learn from a bootstrapped sample without replacement instead of the usual practice of using the original sample.

With bagging [14] (which can be considered as a predecessor of the random forest), Friedman and Hall [54] demonstrate, both theoretically and empirically, that sub-sampling around 50% of the original training set performs similar to standard bootstrap (63%) sampling in terms of bias-variance considerations for regression problems.

To the best of our knowledge, there is no study on the optimal size of a sub-sample to be used for leaning each tree of a random forest based LtR algorithm. That is why this investigation is undertaken.

4.4 Approach

In the existing random forest based LtR algorithms [63, 109], standard bootstrapping is used where sampling is performed without utilizing the query-document structure of the data. That is, sampling is performed not on a per query basis, but rather on a per example (query-document pair) basis. We argue that this is not the most appropriate approach because it may result in fewer documents (and thus less information in the training set) per query. Hence the sub-sampling procedure in our methods ensures that once a query is chosen for inclusion in a training sample, all of its associated documents are chosen, i.e., we sample the training data on a per-query basis.

The usual practice among the researchers is to perform bootstrapping with replacement [15], but Friedman [53] and Freidman and Hall [54] show that bootstrapping without

---

3A split-point consists of an attribute and a value for that attribute.
**Chapter 4: Reducing Correlation Between the Trees**

**Procedure:** \( \text{BuildForest}(D, E, p) \)

**Data:** Training set \( D \), ensemble size \( E \), number of queries per sub-sample \( p \)

**Result:** Tree ensemble \( Trees \)

\[
\begin{aligned}
    Trees & \leftarrow \emptyset; \\
    \text{for } i \in \{1, \ldots, E\} & \text{ do} \\
    \quad D_i & \leftarrow \emptyset; \\
    \quad \text{while } |Q_{D_i}| < p & \text{ do} \\
    \quad \quad q & \leftarrow \text{chooseRandom}(Q_D \setminus Q_{D_i}); \\
    \quad \quad D_i & .add((q, l_{q,j})_{j=1}^{n_q}); \\
    \quad \text{end} \\
    \quad Trees & .add(\text{BuildTree}(D_i)); \\
    \text{end} \\
    \text{return } Trees;
\end{aligned}
\]

Where the function \( \text{chooseRandom}(A) \) selects an item uniformly at random from the set \( A \).

**Algorithm 2:** Procedure for constructing a random forest with query-level sub-sampling.

![Sub-sampling at tree level of a random forest.](image)

replacement also works well for supervised learning. Moreover, Ganjisaffar et al. [55] show that it works well for the LtR task. Bootstrapping without replacement is lucrative from the perspective of scalability as it reduces learning time.

The query-level sub-sampling procedure used throughout the thesis is shown in Algorithm 2. (The tree building procedure \( \text{BuildTree}(,.) \) was previously shown in Section 3.5.) Thus in our approach, each tree of an ensemble is learnt using a training sample which contains \( p \) randomly chosen queries (and associated documents) from the original training set. Figure 4.1 depicts the procedure. Different values of \( p \) constitute different configurations. We call this algorithm \( \text{RF-point-S}% \) where the symbol \% indicates the percentage of training queries used to learn a tree (\( \% = 100p/|Q_D| \)). As such, the RF-point with the default sampling is denoted by RF-point-S63. When we write bare RF-point, it implies RF-point-S63.
Another approach to reduce correlation using sub-sampling could be to learn a tree using all queries but then to randomly sample a pre-defined number of documents per query. However, this will cause the trees to be relatively more correlated than our proposed approach because of the following reason. Yilmaz and Robertson [169] discover that for the LtR task, given a large pool of queries and documents (from where we need to sample the training examples), sampling more queries with less documents is better than sampling less queries with more documents – better for a rank-learner to learn a good ranking function. This means that in the former case, i.e., sampling more queries with less documents, the characteristics of the original training set (i.e., the collection of documents mentioned above) are better retained in the sampled set. For this reason, if we, as usual, sample all 63% queries for a tree, and then perturb the sampled queries by further (sub-)sampling documents of each of them, then the correlation is not likely to be substantially reduced because the sampled training set for individual trees is, according to the above-mentioned work, likely to contain sufficient information to be representative to the original training set. Hence we do not pursue this direction further.

4.5 Result Analysis

This section describes the experimental settings and analyses the results.

For the first configuration, $p$ is set to 1, i.e., each tree is learnt using a single query. In the subsequent configurations, we gradually increase $p$ by a fixed amount until the sub-sample contains all the queries of the original training set. In order to reduce fluctuation in performance due to the stochastic nature of the algorithm, we perform ten independent runs of each configuration, and report the average value of the evaluation metrics.

We find the best sub-sample size using a validation set (i.e., held-out data), and then apply the learnt model to a separate test set to validate our findings.

The random forest is known to perform well with smaller training sets (e.g., [122]). However, if $p$ is too small, there may be concern regarding the stability of the model. Hence for the experiments we, in addition to measuring performance, analyse the variance of the evaluation metrics across the ten runs. We note, however, that the variance (i.e., instability) is not a major concern for a random forest because it effectively tackles the instability problem of a decision tree by aggregating the predictions of a collage of trees.

To pronounce the effect of using a small sub-sample per tree, an ensemble in this chapter consists of 1000 trees unless otherwise stated.
Figure 4.2: Performance in terms of NDCG@10 (left column) and corresponding standard deviation (right column) for Ohsumed, MQ2008 and MQ2007 datasets (from top to bottom) using validation sets. In the left plots, the points along a vertical line represent NDCG@10 values of 10 independent runs of a configuration (using the % of training queries per tree indicated by the x-axis). A smooth-spline is fitted (with smoothing parameter 0.01). The bar plots in the right indicate the standard deviations of the corresponding NDCG@10 values across the 10 runs.

4.5.1 Relatively Small Datasets

Figures 4.2 and 4.3 show the performance on validation sets in terms of NDCG@10 of the ten independent runs of each configuration. In the left plots, each point is the NDCG@10 of a single run (out of ten) for an experiment where a tree of the ensemble in question is learnt from a certain percentage of randomly chosen queries. There are ten points at each x-value showing the NDCG@10 value for ten independent runs. For example, for the configuration where each tree of the ensemble uses 60% random queries as its training set, the ten NDCG@10 values are indicated along the vertical line corresponding to $x = 60\%$. A smooth-spline is fitted (with smoothing parameter 0.01) in order to capture the trend of the plot (as shown by the black curve). The bar plots in

\[\text{We also plotted MAP for all the datasets, and found similar trend. For brevity we do not show those plots.}\]
Chapter 4: Reducing Correlation Between the Trees

Figure 4.3: Performance in terms of NDCG@10 (left column) and standard deviation (right column) of TD2004, HP2004 and NP2004 (from top to bottom) using validation sets. For description, see caption of Figure 4.2.

The plots show that the performance of the first configuration is low (except in Ohsumed) which is expected as only one query is not likely to yield good accuracy. The performance then increases in the subsequent configurations. After using a certain percentage of the queries (which depends on dataset), performance starts to decrease, and this trend of reduction continues until the end. It is known to the research community that in a random forest sampling 63% data per tree (instead of using the entire data) generally improves performance; our investigation now reveals that for these datasets the existing practice of sampling is not the best choice.

For Ohsumed dataset, using only one query is found to be amongst the best configurations. A possible reason for this is that the Ohsumed dataset has fewer queries and more documents per query than MQ2008 and MQ2007 (cf. Table 2.2). Hence one query is oftentimes sufficient for a tree to learn a good hypothesis, and because of a large number
of trees aggregated in an ensemble, the overall accuracy is high. Another point to note is, there are only 63 queries in the training set of Ohsumed dataset which is much smaller than the ensemble size (i.e., 1000). On average each query is thus used for learning in 16 (∼ 1000/63) trees, whereas this number stands at only 2 for MQ2008 and 1 for MQ2007 (because of having 470 and 1015 queries in their training sets respectively). Hence we conjecture that as we grow the ensemble size, similar trend, i.e., increasing performance with fewer queries will be found in other datasets. We leave this to future work.

The characteristics of TD2004, HP2004 and NP2004 datasets are drastically different from the above-discussed three. These datasets have fewer queries in the training set (i.e., 50), a large number of documents per query (i.e., ∼ 1000), and a heavily skewed relevance label distribution. From Figure 4.3 we see that TD2004 maintains a comparatively smoother trend which is much similar to that of MQ2008 and MQ2007 – this is probably due to the fact that its skewness in relevance label distribution is less than the other two (i.e., HP2004 and NP2004). HP2004 and NP2004, on the other hand, show a slightly less pronounced trend due to greater variability in performance across different runs. However, the trend is still clear: performance initially increases and then degrades with increased sub-sample size.

The variance in performance across different runs of a configuration seems to be largely random. Thus we see that the conjecture that we made earlier that an RF-based rank-learner can successfully suppress variance of the model holds largely true. Another noticeable point is, the numerical values of variances for the HP2004 and NP2004 datasets are comparatively higher than other four datasets which is possibly because the highly skewed relevance label distribution of their training data makes learning difficult.

Figure 4.4 shows that the training time steadily increases with the increasing training data per tree.
Table 4.1: On test sets, comparison between our sub-sampling method and the baseline in terms of training time and performance (using the best values of $p$ found from validation sets). $p$ is the number of queries used to learn a tree. Performance (NDCG@10 and MAP) is computed as the average of 10 independent runs. Standard deviation is also reported.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>$p$ (and % of training set)</th>
<th>Training Time Speed-up</th>
<th>NDCG@10</th>
<th>MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Reduced sample</td>
<td>Baseline</td>
<td>Reduced sample</td>
<td>Baseline</td>
</tr>
<tr>
<td>Ohsumed</td>
<td>1 (1.6%)</td>
<td>40 (63%)</td>
<td>29 times</td>
<td>0.4443 (±0.0022)</td>
</tr>
<tr>
<td>MQ2008</td>
<td>21 (4.5%)</td>
<td>301 (64%)</td>
<td>17 times</td>
<td>0.2292 (±0.0008)</td>
</tr>
<tr>
<td>MQ2007</td>
<td>51 (5%)</td>
<td>651 (64%)</td>
<td>17 times</td>
<td>0.4447 (±0.0013)</td>
</tr>
<tr>
<td>TD2004</td>
<td>3 (6.7%)</td>
<td>28 (62%)</td>
<td>10 times</td>
<td>0.3742 (±0.0035)</td>
</tr>
<tr>
<td>HP2004</td>
<td>4 (9%)</td>
<td>28 (62%)</td>
<td>7 times</td>
<td>0.8236 (±0.0058)</td>
</tr>
<tr>
<td>NP2004</td>
<td>2 (4.4%)</td>
<td>28 (62%)</td>
<td>8 times</td>
<td>0.7994 (±0.0063)</td>
</tr>
</tbody>
</table>

The above-mentioned analysis was conducted on the results from validation sets. In order to corroborate the findings, we now select the best value of $p$ from these experiments, and apply it to a separate test set.

![Figure 4.5: Comparison of performance (in terms of NDCG@10, cf. Table 4.1) between sub-sampling approach and baseline. Numerical values of increase in performance are (from left to right): 2.9%, 1.4%, 1.9%, 5.5%, 1.5%, and 2.5%.

Table 4.1 shows the results for the test sets. In particular, using the data from Table 4.1, Figure 4.5 shows the graphical view of performance improvement in terms of NDCG@10. From the table we see that across the six datasets, only 2-10% of the queries are needed to get better accuracy. The variance is comparable to that of baseline. Furthermore, the learning time is reduced by 7-29 times. It is thus evident that our method achieves twofold benefit over the baseline: (1) increased performance, and (2) better scalability.

4.5.2 Big Datasets

For big datasets, the variance of individual trees are already comparatively low. Hence the ensemble size can be reduced because the goal of using larger ensemble is to reduce
Figure 4.6: Performance (NDCG@10 and MAP) of MSLR-WEB10K (Fold 1) and Yahoo datasets as the % of queries per tree increases.

variance of individual trees.\textsuperscript{5} As such, we go back to our standard setting (for the thesis) of using an ensemble of 500 trees.

Figure 4.6 shows the plots.\textsuperscript{6} The trend of the plots is different from what we observed in the case of smaller datasets. In general, no improvement in performance is noticed with smaller $p$ over the baseline (i.e., 63% sampling). We can thus conclude that for big data our method of using smaller sub-sample per tree may not yield any stark benefit over traditional bootstrapping in terms of performance.\textsuperscript{7} However, since using smaller $p$ reduces training time, this approach can still be preferred over the baseline.

Figure 4.7 shows the training time per tree as we increase sub-sample size per tree.\textsuperscript{8} For MSLR-WEB10K and Yahoo, the relative speed-up of the first configuration with smaller sample that performed similar to the baseline (which is 25% and 35% for the two datasets respectively) is approximately 2 and 4 times over the baseline (i.e., using 63% queries) respectively. As for the model size, the ratio of tree size is approximately

\textsuperscript{5}This setting is also more practical as employing a very large ensemble for big data needs increasing computational resources.

\textsuperscript{6}Here we report result of a single run of a given configuration (instead of multiple runs) as we, due to the large size of these two datasets, expect less instability in performances across different runs of a configuration. Indeed, the plots appear to be sufficiently smooth for analysis.

\textsuperscript{7}We conduct a pairwise significance test by treating the configuration of 63% sampling as the baseline, and find the following: for the Yahoo dataset, the configurations after 35% training data are indistinguishable from the baseline (with $p$-value $\leq 0.01$), and for MSLR-WEB10K this threshold configuration has been found to be 25% training data.

\textsuperscript{8}The fluctuations are most likely to be due to time sharing on the machine running the experiments. However, the trend can be vividly seen from the plots.
Chapter 4: Reducing Correlation Between the Trees

Figure 4.7: Training time (per tree) on Yahoo and MSLR-WEB10K (Fold 1) datasets as sub-sample size per tree increases.

2 for both the datasets. Thus it is clear that using smaller sub-sample to learn a tree reduces both learning and evaluation time.

The cause for not observing performance improvement in big datasets with smaller subsamples could be explained considering the ensemble variance which is as follows. The more the data, the less need for randomization in a random forest [155], because the goal of incorporating randomization in a random forest is to decrease correlation between the trees, which in turn decreases the variance of the ensemble. If the training data are large, then the individual trees have comparatively lower variance, even when smaller sub-samples are used. This means that for big datasets there is less need to reduce the (already lower) ensemble variance. For this reason, on large datasets using smaller sub-sample per tree could not improve performance over the baseline.

4.6 Further Analysis of Sub-sampling Method

In this section we discuss three aspects regarding the sub-sampling method, namely effect of ensemble size, strength of individual trees and disjoint sub-sampling.

4.6.1 Effect of Ensemble Size

Since the model variance depends on the size of the datasets and is influenced by the ensemble size, the ensemble size is likely to play a key role in the efficacy of sub-sampling approach. This sub-section conducts an investigation into the effect of ensemble size.

Figure 4.8 shows the effect of ensemble size on both the RF-point and RF-point-S% on a smaller and a big datasets, namely MQ2007 (one fold) and Yahoo. Since the number of trees in a random forest can be increased without any concern of overfitting [15], both the algorithms were expected to exhibit performance improvement with increasing ensemble size. The minor fluctuations in performance (note the restricted y-axis) with
much higher ensemble sizes are likely to be due to the inherent randomness of the algorithm. Another observation is that the RF-point-S% reaps comparatively more benefit than RF-point with increasing ensemble size, and the rate of this improvement varies across different datasets; the (smaller) MQ2007 dataset gets the most benefit of using RF-point-S2 (because of its higher variance), and for the (larger) Yahoo dataset RF-point-S5 needs almost 4000-5000 trees to yield similar performance to the baseline, i.e., RF-point-S63.9

Thus this investigation suggests that the smaller the dataset, the bigger the positive effect of using larger ensemble sizes for smaller sample sizes in RF-point-S%, and the amount of this benefit varies greatly across different datasets.

4.6.2 Strength of Individual Trees

By reducing the sub-sample size per tree, we decrease the correlation between the trees which is supposed to cause lower error rate of the ensemble. However, as mentioned in the beginning of the chapter, there is another aspect that controls the accuracy of the ensemble which is the strength, i.e., predictive accuracy of individual trees. Since as a result of reducing sub-sample size the hypothesis space is shrunk, the strength is likely to decrease. In spite of this, we still have achieved better performance – this means that the positive effect of decreasing correlation compensates for the negative effect of reducing strength. As such, one direction for further investigation would be to try to retain, or perhaps increase the strength of individual trees while reducing the sub-sample size by contemporaneously increasing the parameter $K$ (i.e., randomly chosen features at each split). However, increasing $K$ will also increase correlation between the trees,

---

9The rationale for using 2% data per tree for the MQ2007 dataset (unlike the 5% used for the Yahoo) is as follows. The MQ2007 dataset, in contrast to the Yahoo dataset, has already exhibited performance improvement with the sub-sampling method. Hence we needed to choose a configuration that, with the smaller ensemble sizes, performs less than the RF-point-63 (note the relative performance of RF-point-2 and RF-point-63 with 500 trees on Figure 4.8) in order to pronounce the effect of ensemble size on performance.
and in addition, will decrease the achieved scalability for RF-point-S%, which is one of our main goals. Despite these concerns, we conduct a pilot experiment with the MQ2007 and MQ2008 datasets where we execute RF-point-S10 and RF-point-S4.5 respectively as we vary $K$. Figure 4.9 shows the plots. Very little, if any improvement in NDCG@10 of MQ2007 is observed as we increase $K$. Since no clear improvement is found from this investigation, and importantly, because of the concern regarding scalability mentioned above, we do not pursue this investigation further.

4.6.3 Further Reduction in Correlation by Disjoint Sub-sampling

We could employ the following approach to reduce correlation even further. For each sub-sample, as before, $p$ queries will be randomly selected, but in addition, it will be ensured that the sub-samples are, if possible, disjoint from each other. As a concrete example, suppose $p = 5$, $|Q| = 100$, and $E = 500$. Then the first 20 trees will learn from disjoint sub-samples (each one having 5% data), and the next 20 trees will also learn from disjoint sub-samples across themselves (but not across the first batch of 20 trees) and so on. This approach can be called disjoint sub-sampling. However, although sounds effective, this approach is less likely to perform better than the standard sub-sampling due to the following reasons. For smaller datasets we have already achieved better performance with quite small sub-samples per tree. Because of their small sizes, the sub-samples are already quite dissimilar to each other. Hence enforcing that the sub-samples be disjoint is unlikely to be able to de-correlate the trees sufficiently to produce a better outcome. Hence we do not explore the disjoint sub-sampling approach further.
4.7 Theoretical Explanation of Sub-sampling Method

So far we have not validated our claim that sampling smaller data per tree indeed reduces correlation between the trees. This section is devoted to this investigation. We measure the numerical amount of correlation between the trees as we vary sub-sample size per tree. In addition, we examine the effect of sub-sampling on variance of a single tree and individual tree strength, and discuss the ensemble variance.

4.7.1 Estimating Correlation, Variance and Strength

Below we design the formulae to be used for empirical estimation of correlation, single tree variance and single tree strength.

**Correlation.** The correlation between the trees in a random forest measures the extent to which the predictions of trees agree with one another. We estimate the correlation of a random forest as the average pairwise correlation between the predictions of individual trees across the examples. As such, given an ensemble of size $E$ where the score predicted for an instance $x_k$ by a tree $T_i$ is denoted by $s_i(x_k)$, we compute it as follows:

$$\hat{\rho}(T_i, T_j) = \frac{1}{E(E-1)} \sum_{i \neq j} r(\{s_i(x_k)\}_{k=1}^N, \{s_j(x_k)\}_{k=1}^N),$$

(4.1)

where $r(a, b)$ is the Pearson’s correlation coefficient of the two vectors $a$ and $b$. Concretely:

$$r(\{s_i(x_k)\}_{k=1}^N, \{s_j(x_k)\}_{k=1}^N) = \frac{\sum_{k=1}^N (s_i(x_k) - \bar{s}_i)(s_j(x_k) - \bar{s}_j)}{\sqrt{\sum_{k=1}^N (s_i(x_k) - \bar{s}_i)^2} \sqrt{\sum_{k=1}^N (s_j(x_k) - \bar{s}_j)^2}},$$

(4.2)

where $\bar{s}_i = \frac{1}{N} \sum_{k=1}^N s_i(x_k)$, is the average score assigned by tree $T_i$ to the examples $\{x_k\}_{k=1}^N$.

**Single Tree Variance.** For a data point $x_k$ with predicted label $f(x_k)$, its variance is given by: $\mathbb{E}[f(x_k) - \mathbb{E}[f(x_k)]]^2$ where the Expectation is taken over different models (learnt from different samples). We estimate the average variance (over the examples) of a single tree in the ensemble by considering the variance in the predictions of trees
within an ensemble on each datapoint as follows\(^{10}\):

\[
\hat{\sigma}^2_T = \frac{1}{N} \sum_{k=1}^{N} \hat{\sigma}^2(x_k); \quad \hat{\sigma}^2(x_k) = \frac{1}{E-1} \sum_{i=1}^{E} (s_i(x_k) - \bar{s}(x_k))^2,
\]

where \(\bar{s}(x_k) = \frac{1}{E} \sum_{i=1}^{E} s_i(x_k)\) \hspace{1cm} (4.3)

**Single Tree Strength.** The strength of an individual tree is defined for classification as the confidence of its predictive accuracy \([15]\). In a pointwise setting of LtR, it is the degree to which the labels of individual instances are correctly predicted. As such, it should measure the absolute difference between a prediction by a tree and the corresponding relevance label.\(^{11}\) We compute the average reciprocal strength \(RS\) of a tree as follows:

\[
\hat{RS} = \frac{1}{E} \sum_{i=1}^{E} \frac{1}{N} \sum_{k=1}^{N} |s_i(x_k) - l_k|.
\]

**Ensemble Variance.** For a random forest consisting of \(E\) trees, the correlation at a datapoint \(\rho(x_k)\), single tree variance at a datapoint \(\sigma^2_T(x_k)\), and ensemble variance at a datapoint \(x_k\) \(\sigma^2_{RF}(x_k)\) are related by \([69, \text{Ch. 15}]\):

\[
\sigma^2_{RF}(x_k) = \rho(x_k)\sigma^2_T(x_k) + \left(1 - \rho(x_k)\right)\sigma^2_T(x_k).
\]

The derivation of the above equation is relatively straightforward and hence may be found in the literature. Appendix C.1 shows one such derivation.

It might be tempting to use the above-mentioned formula to measure the ensemble variance using the correlation computed in Equation 4.1. But it would not be appropriate because the correlation computed in Equation 4.1 is not the same as the correlation mentioned in the above equation. In the above equation the correlation is defined at a particular datapoint \(x_k\) whereas the correlation in Equation 4.1 is calculated based on all the examples across two trees. As noted by Hastie et al. \([69, \text{Ch. 15}]\) that the term \(\rho(x_k)\) denotes a theoretical quantity, which (to the best of our knowledge) cannot be estimated directly, we leave measuring the variance of a random forest to Chapter 7 where we thoroughly analyze bias-variance of rank-learners.

---

\(^{10}\)We note that we are not estimating the variance of the prediction of a single tree in the ensemble directly, but rather estimating the variance in a generic tree.

\(^{11}\)Alternatively, squared difference instead of absolute difference could be used.
4.7.2 Result Analysis

As the theoretical results are usually better reflected asymptotically, we mainly use the two larger datasets, namely MSLR-WEB10K and Yahoo, for our experimentation. Figure 4.10 shows the plots of correlation, single tree variance, ensemble variance, single tree (reciprocal) strength and performance (in terms of NDCG@10).

We see that both of the datasets demonstrate similar trend across all quantities. As conjectured throughout the chapter, the correlation steadily decreases as we decrease the sample size per tree. Single tree variance steadily decreases as we increase its training
data which is also expected. Strength of individual trees steadily increases (since the reciprocal strength decreases) with increasing $p$ which is in congruence with the plot of individual tree variance, i.e., variance and strength move in the opposite direction.

We also used the MQ2007 dataset (which is of moderate size) in our experiment, and found very similar plots to the ones shown here. For brevity we do not show those plots.

### 4.8 Discussion

We now summarize the observations elicited from the investigation of this chapter.

- Our experiments empirically discover that for smaller datasets, performance of an RF-based rank-learning algorithm can be substantially increased (we found 1.4%-5.5% increase) by properly controlling the sub-sample size per tree. In addition, the training time can significantly be reduced (we found 10-29 times reduction) thereby increasing scalability. When using smaller sub-samples, the model variance does not appear to be increased than its usual value. The optimal size of sub-samples depend on the datasets, but it definitely lies far below the 63% (as currently being practiced in the LtR literature). A possible direction for future research is to find out the aspects of a training set which influence the optimum sub-sample size. Some aspects to investigate are: number of queries, average number of documents per query, and relevance label distribution.

- For big datasets, although in our experiments with the sub-sampling method we have not noticed improved performance over the baseline (due to relatively low variance of individual trees as explained in Section 4.5.2), the method can still be prescribed as it reduces both the learning and evaluation times.

- When using smaller sub-samples (per tree), increasing ensemble size is more useful in terms of performance in small to moderate sized datasets than the larger ones. In general, the larger the ensemble size, the more benefit of sub-sampling method will be visible.

- Increasing $K$ with reduced sub-samples is not likely to yield performance improvement over the default value of $K$.

- The findings of this investigation may also be applicable to classification and regression tasks for improving scalability.

- Empirical measurement of different quantities such as correlation, variance and strength confirm our conjectures made on their relationship with the sub-sample size.
4.9 Conclusion

A random forest has two components which control its accuracy, namely (1) the correlation between the trees, and (2) the strength of the individual trees. In this chapter, we have investigated the former of these two in the context of the LtR problem. We have successfully used an approach of reducing the sub-sample per tree to decrease correlation, thereby resulting in improvement of learning and evaluation times and oftentimes performance.

The plot of the Yahoo dataset (Figure 4.6) indicated that the optimal size of sub-samples may be even more than the existing practice of 63% (i.e., bootstrapping). This raises a new question: are the trees sufficiently “strong” for the Yahoo dataset? In other words, are the trees able to capture the possibly complex hypothesis of Yahoo dataset? We address this question in context of parameter $K$ in Chapter 6.

Inspired by the findings of the theoretical investigation of sub-sampling method, a detailed analysis of bias-variance of rank-learners is conducted in Chapter 7.
Chapter 5

Pointwise and Listwise Objective Functions

In the previous chapter we have investigated scalability and performance issues of RF-based LtR algorithms using the correlation between the trees. In this chapter we investigate the performance from the perspective of objective functions.

Current random forest (RF) based learning-to-rank (LtR) algorithms use a classification or regression framework to solve the ranking problem in a pointwise manner. The success of this simple yet effective approach coupled with the inherent parallelizability of the learning algorithm makes it a strong candidate for widespread adoption. In this chapter we aim to better understand the effectiveness of RF-based rank learning algorithms with a focus on the comparison between pointwise and listwise approaches.

We introduce what we believe to be the first listwise version of an RF-based LtR algorithm. The algorithm directly optimises an information retrieval metric of choice (in our case, NDCG) in a greedy manner. Direct optimisation of the listwise objective functions is computationally prohibitive for most learning algorithms, but possible in RF since each tree maximizes the objective in a coordinate-wise fashion. Computational complexity of the listwise approach is higher than the pointwise counterpart, hence for larger datasets, we design a hybrid algorithm which combines listwise objective in the early stages of tree construction and a pointwise objective in the latter stages. We also study the effect of the discount function of NDCG on the listwise algorithm.

Experimental results on several publicly available LtR datasets reveal that the listwise/hybrid algorithm outperforms the pointwise approach on the majority (but not all) of the datasets. We then investigate several aspects of the two algorithms to better
understand the inevitable performance trade-offs. The aspects include examining an 
RF-based unsupervised LtR algorithm, and comparing individual tree strength.

5.1 Motivation

For the LtR problem, listwise algorithms have been shown empirically [90, 94] to outper-
form pointwise algorithms in general. Their generalization ability has also been proven 
theoretically [88]. However, the listwise algorithms are usually computationally more 
demanding, and they oftentimes lack conceptual simplicity.

While it is relatively straightforward to design an RF-based pointwise algorithm, we 
found no existing work on its listwise counterpart. The main barrier to design a listwise 
algorithm is that most of the IR metrics are non-smooth and non-convex with respect to 
model parameters which makes them difficult to directly optimise using many learning 
algorithms. However, decision tree based algorithms optimise an objective function by 
splitting the data in a coordinate-wise fashion (by enumerating all possible cut-points 
along a subset of the dimensions). Thus it is possible to apply these techniques directly 
to the optimisation of non-convex and even non-smooth objective functions (such as 
those that directly optimise IR metrics like NDCG). RF-based pointwise algorithms 
have already been shown to be competitive to the state-of-the-art methods [63, 109]. In 
this chapter we extensively compare the random forest based pointwise and listwise LtR 
algorithms.

5.2 Contributions

The major contributions of this chapter are as follows:

- We design a random forest based listwise LtR algorithm which directly optimises 
  any rank-based metric – we use NDCG and some of its variations.

- We design a random forest based hybrid algorithm which can incorporate different 
  splitting criteria in a single tree. This is especially useful when some splitting 
  criteria have higher computational complexity (e.g. NDCG-based ones) than oth-
  ers (e.g. entropy-based ones) because the amount of computationally demanding 
  splits can be controlled depending on the availability of computational resources. 
  We empirically show that its performance is at least as good as its pointwise coun-
  terpart.

- Our experiments reveal that:
1. For smaller and moderate LtR datasets (in terms of number of queries) with graded relevance labels, the performance of the listwise approach is better than its pointwise counterpart.

2. For large datasets, performances of RF-based pointwise and listwise/hybrid algorithms seem to be dependent on the properties of the dataset.

• We compare the relative performance difference of pointwise and listwise algorithms by investigating the predictive accuracy (also known as strength) of individual trees in terms of ranking performance. This reveals an aspect that is common to both objective functions: both of them tend to isolate similar training instances in the leaves of a tree. Hence when a tree is sufficiently deep, the difference between the effects of the two objective functions diminishes, thereby yielding similar performance in some cases.

• To better understand the role of splitting criterion, we also investigate a purely-random (unsupervised partitioning)-tree based ensemble LtR algorithm. While on big datasets its performance is not on a par with the random forest based pointwise/listwise algorithms, on smaller datasets it performs close to RF-point. We also develop an RF-based pairwise algorithm.

The rest of the chapter is organized as follows. Section 5.3 reviews some related methods. Section 5.4 explains an (existing) random forest based pointwise algorithm (RF-point) that will be used as the baseline, and then designs its listwise counterpart (RF-list). Here we also discuss the time complexities of the pointwise and listwise algorithms which then drives us to develop a hybrid algorithm. Section 5.5 discusses the experimental results of different algorithms. Section 5.6 investigates the performance difference of pointwise and listwise objective functions from several perspectives.

### 5.3 Related Work

Our work is related to two threads of research: rank-learning algorithms based on the random forest, and techniques for direct optimisation of ranking measures. These are described below.

#### 5.3.1 Learning to Rank Using Random Forest

Various machine learning algorithms have been extensively used to solve the LtR problem, yet the application of random forests has thus far not been thoroughly investigated.
Geurts and Louppe [63] use a variation of random forest called Extremely Randomized Trees (ERT) [62] with both classification and regression settings, i.e., using a pointwise approach. Despite the fact that simple settings are used, they report competitive performance on the Yahoo dataset to other state-of-the-art algorithms. Mohan et al. [109] also use a pointwise approach with regression setting. The main difference between these two works is that the former one used a variation of original random forest while the latter one used the original framework.

5.3.2 Direct Optimisation of Ranking Measures

Several methods on direct optimisation of ranking measures for rank-learning have been proposed. Below we describe some of them.

Metzler and Croft [105] use the coordinate ascent method to learn a ranking function which finds a local optimum of an objective function that directly uses an IR metric like NDCG or MAP. The coordinate ascent updates one parameter at a time while keeping other parameters fixed, thereby minimizing the scalability issue of conducting a grid search over a large parameter space. The authors report better results than a standard language model.

Wu et al. [159] propose the LambdaMart algorithm which blends the sophisticated idea of approximated gradient of its predecessor, namely LambdaRank [125] with the gradient boosting framework [52]. The gradient boosting framework can be used to optimise any loss function for which a gradient is defined at each training point, and since LambdaRank uses an approximation to the gradient of the implicit loss function at each training point, these approximated gradients can be used in the gradient boosting framework (instead of the neural network framework which was used in LambdaRank). More specifically, the gradient boosted framework is instantiated using regression trees, and thus LambdaMart iteratively builds a sequence of regression trees, where each subsequent tree fits the approximated gradient of the the listwise-loss at the current “point” in the “model space”.

Taylor et al. [149] address the issue of non-smoothness of the IR metrics, and point out that although the IR metrics like NDCG are either flat or discontinuous everywhere with respect to the model parameters, the scores are smooth with respect to the model parameters. They argue that if the individual ranks of the instances can be mapped into a smooth distribution instead of discrete ranks, then these ranks will be changed smoothly with changes of the model parameters. NDCG changes only if the ranks of the documents change, but if the ranks themselves are allowed to change smoothly with respect to the model parameters, i.e., if non-integer ranks are used, then it is
possible to devise a soft version of NDCG that changes smoothly with respect to the model parameters, thereby solving the fundamental problem of non-smoothness of the IR metrics. The authors use neural network framework to optimise the objective function. The algorithm is called as SoftRank. A limitation of this algorithm is that it takes $O(n_q^3)$ time where $n_q$ is the number of documents per query.

Xu and Li [166] propose a conceptually simple but effective algorithm called AdaRank which uses the AdaBoost framework [50] while maintaining a listwise approach. The AdaBoost framework assigns weights to the individual (weak) learners of an ensemble according to the classification error of the learner on weighted training instances. The authors modify the weighting scheme so that instead of classification error, ranking error of all documents associated with a query is used. Note that the weights are associated with queries rather than single documents thereby retaining the query-document structure of the training data. This ensures that the later learners of the ensemble focus on the hard queries (and associated documents) to achieve an overall good NDCG. This way AdaRank produces weak learners that are specifically built to deliver high NDCG, in contrast to delivering high classification accuracy.

Like Metzler and Croft [105], Tan et al. [148] also use the coordinate ascent method in a listwise manner. The main difference between the two approaches is that the latter locates the optimal point along a particular coordinate (i.e., feature) using a sophisticated technique while the former resort to heuristics to find a good point.

Qin et al. [123] develop a general framework which first converts any rank-based metric into a document-based metric, and then replaces the position function of the differences of scores between the said document and every other document of the list (which is an indicator function and hence non-continuous and non-differentiable) by the (smooth) logistic function. This surrogate objective function can then be optimised using any technique which is able to handle the local optima problem (the authors use the random restart method). A drawback of this framework is that the converted document-based metric needs $O(n_q^2)$ time complexity whereas original sorting could be done in $O(n_q \log n_q)$ time. Another drawback is that they use only three small datasets, namely Ohsumed, TD2003 and TD2004.

### 5.3.3 Motivation for Our Approach

From the previous discussion we see that the objective functions of LtR algorithms which use boosting, neural networks and support vector machines have been well investigated in the literature. For example, with gradient boosting, Li et al. [92] and Quoc and Le [125] use pointwise and listwise algorithms respectively. With AdaBoost, Freund et al.
Chapter 5: Pointwise and Listwise Objective Functions

Table 5.1: LtR algorithms using different approaches and frameworks.

<table>
<thead>
<tr>
<th>Method</th>
<th>Pointwise</th>
<th>Pairwise</th>
<th>Listwise</th>
</tr>
</thead>
<tbody>
<tr>
<td>Coordinate ascent</td>
<td>-</td>
<td>-</td>
<td>[105], [148]</td>
</tr>
<tr>
<td>Adaboost</td>
<td>-</td>
<td>[49]</td>
<td>166</td>
</tr>
<tr>
<td>Gradient boosting</td>
<td>[92]</td>
<td>-</td>
<td>159</td>
</tr>
<tr>
<td>Neural Net</td>
<td>-</td>
<td>[125]</td>
<td>149</td>
</tr>
<tr>
<td>SVM</td>
<td>[110]</td>
<td>[83]</td>
<td>[172]</td>
</tr>
<tr>
<td>Random forest</td>
<td>[63]</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

[49] and Xu and Li [166] are pairwise and listwise algorithms respectively. With neural networks, Cao et al. [22] and Taylor et al. [149] use listwise algorithms. With SVM, Nallapati [110], Joachims [83] and Yue et al. [172] use pointwise, pairwise and listwise algorithms respectively. Table 5.1 summarizes some of the methods under different approaches.

In spite of achieving much success in other disciplines such as bioinformatics [122], image processing [33], spectral data processing [66] etc., there is little work on using random forest for solving the LtR problem, and importantly, to the best of our knowledge, there is no work on using direct optimisation of ranking measures. This chapter aims at filling this gap. In this chapter we specifically investigate its objective function.

Since the NDCG function is both non-smooth and non-convex with respect to the model parameters, it is difficult to directly optimise the loss function of Equation 2.7. Instead, LtR practitioners normally use surrogate loss functions, for example, standard loss functions of classification/regression problems (such as logistic loss, exponential loss etc.) because perfect classification/regression leads to perfect ranking. In this work we address the question: how does a random forest based LtR algorithm perform with different objective functions? As such, we investigate the pointwise and listwise objective functions, and analyse their similarities and differences.

5.4 Approach

In this section we first describe some properties common to all of the algorithms studied in the chapter. We then refer to the existing RF-based pointwise algorithm which will be used as the baseline. After that we develop a listwise algorithm.

The pseudo-code of a generic random forest has already been given in Algorithms 1 (Chapter 3) and 2 (Chapter 4).
5.4.1 Common Properties for All RF-based Algorithms

We discuss three aspects which are as follows: (1) document score during training, (2) sub-sampling per tree, and (3) termination criteria.

5.4.1.1 Document Score During Test Phase

In order to assign a score to a document with respect to a given query, we use the expected relevance (employed by Li et al. [92], Geurts and Louppe [63], Mohan et al. [109] etc.) over the training data instances at that particular node of the tree. Specifically, every leaf of a tree (partition of the data space) is given a score as follows:

$$\text{score}(\text{leaf}) = \sum_{c=0}^{C-1} p(c|\text{leaf}) \times g(c)$$  \hspace{1cm} (5.1)

where $p(c|\text{leaf})$ denotes the relative frequency of relevance level $c \in \{0, 1, ..., C-1\}$ given the leaf of the tree, and $g(c) = c$.\footnote{This equation was originally proposed by Li et al. [92] where $g(.)$ is any monotonically increasing function of $c$. The authors (and later Mohan et al. [108]) empirically examine various $g(.)$ including $c$, $c^2$, $2^c$ on several datasets such as Yahoo, and conclude that $g(c) = c$ works the best. While we utilize their finding in this thesis, we think it is still an open research question as to which instantiation of $g(c)$ is more appropriate in general.}

The score assigned to a test instance (e.g. $j$th document of query $q$) by an ensemble of $E$ trees is calculated as:

$$f(\vec{x}_{q,j}) = \frac{1}{E} \sum_{i=1}^{E} \text{score}_i(\text{leaf})$$  \hspace{1cm} (5.2)

where the $\text{score}_i(\text{leaf})$ is the score of the leaf of $i$th tree where the test instance has landed.

5.4.1.2 Sub-sampling Per Tree

As for choosing between document-level and query-level sampling, traditionally bootstrapped samples are used for learning a tree of a random forest. A bootstrapped sample is formed by randomly choosing instances with replacement from the original sample, and thus it has equal size to the original sample. In pointwise algorithms, sampling can be performed at the document (instance) level because it assigns a loss to individual query-document pairs, although query-level sampling is also applicable. For a listwise algorithm which aims at optimising IR metrics such as NDCG directly, NDCGs of individual queries should be as reliable as possible. Hence we need to sample the training
data on a per-query basis, which means, to sample a query along with all the documents associated with it.

As for choosing between sampling with replacement and without replacement, the usual practice among RF researchers is to perform sampling with replacement. However, Friedman [53], and Friedman and Hall [54] show that sampling without replacement also works well for classification and regression. Also, according to Ganjisaffar et al. [55], it works well for the LtR task. Moreover, it reduces training time significantly, especially for large datasets. Recent works by Scornet et al. [138] and Wager [154] show that sampling without replacement allows for simpler theoretical analysis (which includes variance estimation) as compared to sampling with replacement. Finally, some methods such as Geurts and Louppe [63] do not use any sampling at all, i.e., use the entire data per tree. For these reasons, our setting regarding sub-sampling is to sub-sample (without replacement) 63% of the queries per tree, which means 63% queries (and all of their associated documents) are in a resultant sample. These settings are maintained unless otherwise noted (when dealing with large datasets, we modify the number of queries to sample as will be discussed in Section 5.5.1).

5.4.1.3 Termination criteria

To stop further splitting of a node of a tree, the prevalent practice in RF literature is to grow a tree as deep as possible, thereby deliberately allowing the individual tree to overfit in an attempt to decrease the bias [15]. This is favorable to a random forest because the role of learning each individual tree is to sample from the space of possible predictors. The individual (high) variance of each tree does not deteriorate the performance of the ensemble because the prediction of the ensemble is the average of the predictions of individual trees, thereby reducing the variance of the ensemble. However, later studies such as Segal [141] empirically discover for regression problems that marginal improvement may be found by tuning the maximum node size parameter of a tree, and later works on theoretical analysis of RF such as Lin and Jeon [93], Geurts et al. [62], Scornet et al. [138], Scornet [137] support this claim by arguing that as the data size grows, the maximum node size can also grow. However, for the sake of clarity and better readability, we do not investigate this aspect in this chapter (except a short discussion on overfitting in Section 5.6.2), thereby leaving it to future work. Throughout the chapter we maintain the predominant practice of letting the trees grow as deep as possible. Thus the condition for ceasing to split nodes is: if no gain can be found by considering all the split-points of the randomly chosen $K$ features. Note that

---

$^2$63 is the average cardinality of a bootstrap sample (without replacement).
this condition also encompasses the case when the node contains instances from only one label.

### 5.4.2 Random Forest Based Pointwise Algorithm

It has been shown that both classification [92] and regression errors [31] provide upper bounds on the ranking error. That is why existing pointwise algorithms for RF-based rank-learning yield good performance in general. As a baseline system of this chapter we use the RF-point algorithm introduced in Section 3.5. Recall that this algorithm can be interpreted as a blend of the classification and regression settings – classification setting (with entropy function as the objective) is used to identify the best partitions of the sample space (i.e., during the training phase), while the regression setting is used to assign scores to the final data partitions obtained.

The *Gain* routine was shown in Algorithm 1 (Chapter 3). Throughout the thesis we follow the formulation of gain given in a notable review paper by Criminisi [32].

### 5.4.3 Random Forest Based Listwise Algorithm

A problem of using a surrogate objective function instead of using the true objective (e.g. Equation 2.7) is that the learning algorithm may spend much effort to correctly classify some instances which have lower contribution in a graded IR metric like NDCG, and even worse, perhaps at the cost of wrongly classifying some instances from the top of the current ranked list (which are important from the perspective of NDCG). Therefore, a natural choice would be to choose a split-point which has direct relationship with NDCG. This is the motivation behind developing the algorithm of this sub-section. The goal is to learn individual trees in such a way that NDCG (or any other ranking metric) is maximized. The primary difference between the existing pointwise algorithm and the proposed listwise algorithm is pictorially explained in Figure 5.1. We henceforth call this algorithm *RF-list*. The main novelties of RF-list as compared to RF-point are: (1) using a loss function which directly optimises NDCG (i.e., not an approximation to it as adopted by some existing works), and (2) using it in a random forest framework.

In Figure 5.2 we illustrate the fact that in RF-point (the left figure), the splitting criterion is a local decision because only the training data instances (feature vectors for query-document pairs) which reside at that node are involved in calculating the change in entropy that results from splitting the node into two children. But the RF-list (the right figure) involves data of all leaves of the current structure of a tree, i.e., the entire

---

3 Graded means a decaying function is used as discount of each rank as the rank goes down to a list.
Chapter 5: Pointwise and Listwise Objective Functions

### Training Data

#### Pointwise Training
Learn \( f(x) \) to optimize Entropy-based objective.

#### Listwise Training
Learn \( f(x) \) to optimize NDCG-based objective.

### Evaluation
using rank-based metrics such as NDCG.

### Test Data

Figure 5.1: Relationship between (existing) RF-point and (proposed) RF-list

- **(a) Splitting in a pointwise tree.**
- **(b) Splitting in a listwise tree.**

Figure 5.2: Local decision in a pointwise tree vs global decision in a listwise tree

In a nutshell, the basic idea of RF-list is to recursively partition the training data by choosing split-points such that for a particular split the training instances in the two resulting leaf nodes have such scores that maximize NDCG across all the queries of the training set.

The inevitable question arises here regarding the computational difficulty of directly optimising NDCG or any other rank-based metric. We note, however, that for a decision tree, it is not necessary for the objective function to be convex, because the learning algorithm attempts to optimise the objective in a greedy manner.

The subtleties of RF-list are described below.

#### 5.4.3.1 Document Scores During Training

This issue is not present in RF-point because the entropy-based splitting criterion is invariant to the predicted labels of partitions. However, in order to compute a listwise...
objective, RF-list needs to assign a score to each document during training. To assign score to documents in a given partition during training we use the *expected relevance* introduced in Equation 5.1. The motivation is that the more predictive the individual trees are, the better the NDCG.\(^4\) (We cannot use Equation 5.2 because the trees of a random forest are learnt independently from each other which makes it completely parallelizable.)

5.4.3.2 Computing the Objective Function

We now explain as to how a split-point at a particular node of a tree is determined. Suppose we need to select a split point for a particular node \(j\). A subset of the feature vectors (corresponding to query-document pairs) will be present at that node. It is likely then that those feature vectors do not cover all training queries \(m\), but only a subset of them \(m_j \subseteq m\). For a potential split-point, the scores of the documents directed to left and right children are computed according to the previous discussion (i.e., Section 5.4.3.1), and then the NDCG values of each of the \(m_j\) queries should be updated according to the new scores of the documents. For the rest of the \(m - m_j\) queries, their current NDCGs will not change due to this split because the scores and ranks of their instance documents (sitting in other leaves of the tree) will not change. The split-point which gives the maximum average NDCG across all queries should be selected as the best split-point for the particular (parent) node in question. Thus unlike traditional splitting criteria such as entropy where only the data of a particular node are used to decide the split-point, when optimising a listwise objective function in terms of ranks (such as NDCG), the data at all leaf nodes of the tree (i.e., the entire training data of the tree) must be considered.

Now the key question is, how can we calculate the NDCG of a query when many instances have the same score (since they fall into the same leaf of a particular tree)? While tied document scores rarely occur as the average score in an ensemble due to the fact that different trees split the data differently, they are common for a single tree. Tied scores are especially problematic in early nodes of since initially (at the root node) the tree assigns the same score to all documents. Intuitively, a random order is not a solution. We, therefore, calculate the Expected NDCG over all possible orderings of the tied documents, which can be done efficiently as follows:

\[
\mathbb{E}_{\vec{r} \in \text{rankings}(q; f)}[\text{NDCG}(q; f)] = \frac{\mathbb{E}_r[\text{DCG}(q; f)]}{\max_f \text{DCG}(q; f)} \quad (5.3)
\]

\(^4\)Indeed, the two components control the error rate of a random forest are: (1) the correlation among the trees – the lower the better, and (2) the strengths of individual trees – the higher the better [15]. In our context, the better the NDCG predicted by a single tree, the higher the strength of that tree.
Chapter 5: Pointwise and Listwise Objective Functions

Table 5.2: An example of Expected NDCG computation.

<table>
<thead>
<tr>
<th>Sorted Docs</th>
<th>Assigned Scores</th>
<th>True Label</th>
<th>Gain $(2^l_{q,doc(r',q,f)} - 1)$</th>
<th>Expected Gain $(\text{cf. Equation } 5.5)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$d_3$</td>
<td>1.7</td>
<td>0</td>
<td>0</td>
<td>1.33</td>
</tr>
<tr>
<td>$d_5$</td>
<td>1.7</td>
<td>2</td>
<td>3</td>
<td>1.33</td>
</tr>
<tr>
<td>$d_7$</td>
<td>1.7</td>
<td>1</td>
<td>1</td>
<td>1.33</td>
</tr>
<tr>
<td>$d_9$</td>
<td>0.9</td>
<td>1</td>
<td>1</td>
<td>0.5</td>
</tr>
<tr>
<td>$d_1$</td>
<td>0.4</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
<tr>
<td>$d_4$</td>
<td>0.4</td>
<td>1</td>
<td>1</td>
<td>1.0</td>
</tr>
</tbody>
</table>

Where rankings($q; f$) denotes the set of equivalent rankings and $\vec{r}$ is one such ranking.

Using Equation 2.13 and the linearity of Expectation, we can write the Expected DCG in terms of the Expected gain at each rank position:

$$E_{\vec{r}}[DCG(q; f)] = \sum_{r=1}^{n_q} \mathbb{E}_{\vec{r}' \in \text{ties}(r,q,f)}\left[2^l_{q,doc(r',q,f)} - 1\right] \frac{\log(r + 1)}{\log(r + 1)} (5.4)$$

where ties($r; q, f$) denotes the set of ranks around rank $r$ which have all been assigned the same score by the retrieval function (tree) $f$ for query $q$. The Expected gain at rank $r$ can then be computed as:

$$\mathbb{E}_{r' \in \text{ties}(r)}\left[2^l_{q,doc(r',q,f)} - 1\right] = \frac{1}{\max[\text{ties}(r)] - \min[\text{ties}(r)] + 1} \sum_{j=\min[\text{ties}(r)]}^{\max[\text{ties}(r)]} 2^l_{q,doc(j,q,f)} - 1 (5.5)$$

where $\min[\text{ties}(r)]$ and $\max[\text{ties}(r)]$ are the top and bottom tied ranks around rank $r$.

We now illustrate the above-mentioned concept with an example shown in Table 5.2 which depicts the scenario of seven documents of a query. It is assumed that the seven documents are located in three difference leaves (having scores 1.7, 0.9 and 0.4). Note that there may be documents of other queries in the same leaf, so the assigned scores (2nd column) are not necessarily the average label of the documents of the said query residing in that particular leaf. The values of the last column are calculated simply as the arithmetic average of the values of 4th column for which the score (2nd column) is equal (e.g. 1.33 is the average of 0, 3, and 1).

The Gain routine of Algorithm 1 in the listwise setting is implemented according to Algorithm 3. More details of its implementation is given in Appendix D.3. The algorithm for building a tree and a forest were previously given in Chapters 3 (Algorithm 1) and 4 (Algorithm 2).

Modifying the Listwise Objective Function. We now examine some variations of the listwise objective function. The original NDCG formula (cf. Equation 2.13) is highly
Chapter 5: Pointwise and Listwise Objective Functions

Procedure: \( \text{Gain}_{\text{NDCG}}(\text{tree}, D, D_{\text{left}}, D_{\text{right}}) \)

Data: Current \( \text{tree} \), data at node and left/right child: \( D, D_{\text{left}}, D_{\text{right}} \)

Result: Change in NDCG resulting from split.

begin
leaves ← leavesSortedByAverageRelevance(tree);
leaves.remove(node(D));
leaves.insert(node(D_{\text{left}}), node(D_{\text{right}}));
for \( q \in Q_D \) do
rank[0] ← 0;
i ← 0;
for leaf ∈ leaves do
if \( q \in Q_{\text{leaf}} \) then
i++;
counts[i] ← |D_{\text{leaf}, q}|;
rank[i] ← rank[i - 1] + count[i];
avgGain[i] ← \frac{1}{\text{counts}[i]} \sum_{d \in D_{\text{leaf}, q}} 2^{q_{d}} - 1;
end
expNDCG[q] ← \sum_i \text{counts}[i] \times \text{avgGain}[i] \times \sum_{k=\text{rank}[i-1]}^{\text{rank}[i]} \frac{1}{\log(k+1)};
end
return \left( \frac{1}{|Q|} \sum_q \text{expNDCG}[q] \right) - \text{previousAvgExpNDCG};
end

Note that to reduce the time complexity, the \( \text{avgGain} \) at each leaf for each query is maintained, and the cumulative discount factor \( \sum_{k=\text{rank}[i-1]}^{\text{rank}[i]} \frac{1}{\log(k+1)} \) is precomputed, so that during gain computation they can be executed in constant time.

Algorithm 3: Listwise gain (using Expected NDCG)

influenced by the top few ranks in order to capture user satisfaction [82]. It is likely that this characteristic has an influence in the performance of RF-list in the sense that a more gentle discount function may cause the learning process to generalize better.

It is known to the research community that apart from empirical success at capturing user satisfaction, there is no theoretical justification for the decay function used in Equation 2.13 [34]. Moreover, it has been shown by Wang et al. [157], both theoretically and empirically, that the discount factor of NDCG does have an effect in distinguishing between a good and bad ranked lists. Hence in this section we alter the influence of the discount factor in the following two ways.

1. We modify the discount factor of standard NDCG (Equation 2.12), by raising the log term to a power as shown in Equation 5.6. As \( \alpha \) approaches zero, the influence of top few ranks is gradually reduced, and when \( \alpha = 0 \) there is no discount at all, implying that all the ranks are considered as the same. We also examine values of \( \alpha \) that are greater than 1, implying that the NDCG function is even more sensitive to the top few ranks than the standard version – we conjecture that this will not work well.

\[
\text{discount}_\alpha(r) = [\log(r + 1)]^\alpha; \quad \alpha > 0
\]
2. According to Wang et al. [157], the discount function given in Equation 5.7 is appealing from a theoretical perspective in the sense that it can better distinguish between a good and a bad ranked lists. The influence of top few ranks of standard NDCG formula is less pronounced in this case.

\[ \text{discount}_\beta(r) = r^\beta; \quad \beta \in (0, 1) \]  \hspace{1cm} (5.7)

5.4.3.3 Ordering of Leaf Expansion

A tree in a random forest is usually grown in a recursive (i.e. depth-first) manner. For pointwise loss functions the order of expansion of nodes does not influence the data partitions, since the change in the loss function value that results from the splitting of a node is computed locally, i.e., based only on the data at that node. A listwise loss function, however, cannot be decomposed into losses on individual data points, and instead must be computed over the entire set of documents associated with a particular query. Since the documents of a particular query spread out all over the leaves of a tree, the objective function cannot be calculated locally, i.e., without knowledge of the scores currently assigned to all other documents in the tree. Thus in a listwise approach the node expansion ordering does indeed matter.

Node Expansion Strategies. The recursive implementation, i.e., depth-first exploration is mostly used in the literature due to its ease of implementation. However, it makes the intermediate structures of a tree highly imbalanced which is not favorable for a listwise objective function. Hence we consider the following four exploration strategies: (1) breadth-first: maintains a FIFO queue of leaves, (2) random: selects a random leaf among the available leaves, (3) biggest-first: selects the biggest leaf, the rationale is the biggest node should get priority to be split in order to reduce the amount of sub-optimal predictions, (4) most relevant leaf first: as the name implies, selects the node whose score is the highest (i.e., contains the most relevant documents) because relevant documents are of most interest from the perspective of NDCG. We applied all of these strategies on the MSLR-WEB10K (fold 1) dataset, and found their results mostly similar to each other (the results are given in Appendix D.1). In the breadth-first exploration, data size of a node gradually decreases as the depth of the tree increases, thus it provides a systematic expansion ordering. Hence throughout the rest of the chapter we employ the breadth-first expansion strategy.
5.4.4 A Hybrid Algorithm to Scale-up the Listwise Algorithm

Usually the listwise algorithms have higher computational complexity, and RF-list is expected to be no exception. In this section we first analyse the time complexities of RF-point and RF-list which confirms that RF-list has higher computational complexity than RF-point. Since this may prohibit the use of RF-list for very large datasets when computational resources are limited, we then discuss some heuristics to scale up the algorithm. Relying on heuristics comes at a cost of performance degradation (as compared to original RF-list). We then design a hybrid algorithm which is a better option for scaling up RF-list.

5.4.4.1 Time Complexity

Given \( N \) training instances over \( M \) features and containing \( Q \) queries, the training time complexities RF-point and RF-list are as follows (see Appendix D.2 for their derivations).

**RF-point:** \( O(\log(M)N \log^2(N)) \)

**RF-list:** \( O(\log(M)N(\log^2(N) + QN)) \)

We see that the time complexity of RF-list is greater than quadratic in the number of training examples. For large datasets which is natural for LtR, it may thus be computationally prohibitive. Indeed, as will be shown in Section 5.5, the largest dataset on which we managed to apply RF-list was MQ2007 that contains approximately 1000 queries (45000 query-document) pairs in the training set. For the larger ones, we had to resort to using smaller sub-samples per tree which may result in sub-optimal performance unless the ensemble size is very large – which again may not be supported by the available computational resources. Hence below we discuss some heuristics to reduce the learning time of RF-list. We then design a hybrid algorithm to tackle the computational cost issue.

5.4.4.2 Heuristics to Reduce Time Complexity of RF-list

The following heuristics may be applied to reduce the training time of RF-list.

1. Using a random subset of the available queries at a node for gain computation.

2. Using a smaller fraction of data (instead of traditional 63% data) to learn a tree (cf. the sub-sampling method of Chapter 4).
3. Increasing the minimum number of instances in a leaf (which is tantamount to decreasing the height of a tree).

4. Considering a fraction of possible split-values for a feature instead of enumerating all of them (similar idea is used by Denil et al. [38], Geurts and Louppe [63], Lin and Jeon [93], Ishwaran [80]).

5. Using a larger gain threshold to split a node.

6. Using NDCG@k with a very small k instead of the untruncated one in the gain computation.

We conjecture that each one of the above will increase the bias of individual trees because each of them limits the explorative ability of a tree. Specifically, these heuristics will reduce the tree size – either due to lack of sufficient training data (idea 2), due to explicitly not splitting a node further (ideas 3, 5), due to exploring a limited sample space which results in no further gain (idea 1), or due to limiting the number of potential split-points which may not see further gain (idea 4) – thereby making the individual trees less strong, i.e., with less predictive power. That said, after extensive experiments using the validation set, an effective setting of these parameters may be found which we leave to a future work.\(^5\) Instead, we explore yet another interesting idea as detailed below.

### 5.4.4.3 A Combination of Listwise and Pointwise Splitting

After implementing the algorithms, our experience is that the major portion of learning time of a tree in RF-list is spent when a tree becomes sufficiently deep, i.e., when the number of leaves is large, because for every possible split-point it is computationally expensive to traverse all the leaves. In this section, we aim at minimizing the learning

\(^5\)Except some limited experiments with idea 2 as will be detailed in Section 5.5.
time without using any heuristics mentioned in the previous discussion (which may increase bias). We achieve this by splitting the early nodes of a tree using the listwise objective function, and afterwards resorting to the pointwise objective. That is, up to some pre-defined level of a tree the listwise splitting is used, and for the rest of the nodes the pointwise splitting is used. We call this algorithm \textit{RF-hybrid-Lx} where the parameter \( x \) is the number of levels of a tree up to which the listwise splitting is used. Figure 5.3 depicts the scenario. (Note that breadth-first enumeration is the natural choice here too.)

The idea explained above has an intuitive interpretation. Suppose we use the listwise splitting for the first two levels (i.e., \textit{RF-hybrid-L2} as shown in Figure 5.3). This means that the first three splits (namely the splitting of the root node and then of its two children) are based on listwise objective function. We can view this as if we have four sub-trees which themselves are learnt solely based on the pointwise objective function. We note, however, that the final partitioning produced by the hybrid algorithm is strongly informed by the listwise objective, since that is used to determine the initial and most important cuts in the space. Thus the hybrid algorithm is very different from a pointwise approach. When a test instance traverses the tree, two decisions are made (using the root and either the left or right child of it) as to which pointwise sub-tree should be used to predict the label of this instance. Assuming that the listwise splitting is worthwhile, this hybrid algorithm is thus expected to be able to leverage some of the discriminative power of the listwise objective function depending on the computational resource requirement – the higher the availability of resources, the more listwise splitting will be used.

We note here that it is a well-known practice in classification task to use different types of trees in a random forest (e.g. [131, 163]) in order to get the benefit of various splitting functions, and at the same time to further reduce the correlation among the trees. We are not aware of, however, any research which uses different splitting methods in a single tree.

5.4.5 Random Forest Based Pairwise Algorithm

So far we have discussed RF-based pointwise and listwise objective functions. To complement our investigation, we could think about designing an RF-based pairwise algorithm which can be accomplished as described next. Firstly, we need to create a new training set as follows: for every pair of documents which belong to the same query and which have different relevance judgements, we generate a new feature vector by subtracting one from the other, and assign a new ground truth label, either +1 or -1 depending on
their marginal relevance. After that, we shall fit a standard random forest with classification setting to the modified data. During evaluation, every pair of the documents will have a preference label predicted by the model from which it is, using a topological sort, straightforward to generate a ranking.

Although pairwise algorithms such as RankBoost [49] were popular in early times of LtR research, they essentially operate on quadratic sized training data. Hence for big datasets it is mostly not a feasible solution, and that is mainly why in recent years they have drawn comparatively less interest of research community. However, for the sake of comprehensiveness of our investigation into the splitting criteria (i.e., in addition to pointwise and listwise splitting criteria), we design a greedy but scalable pairwise objective function as detailed in Algorithm 4. The function computes a gain for a split by counting the number of correctly ranked pairs of documents of each query of the data at the parent node. Thus the gain is, like its pointwise counterpart and unlike its listwise counterpart, computed locally, thereby causing the algorithm to be greedy. A weight can be assigned to a correctly ranked pair depending on the labels of the pair – the higher the difference between the two labels, the higher the weight (i.e., reward) for ranking the pair correctly (in fact, any monotonically increasing function of the difference can be used). We call this algorithm RF-pair. Note in our implementation that we avoid exhaustive enumeration (of quadratic order) of all pairs by exploiting the fact that usually the number of distinct relevance labels in LtR data does not exceed five, thereby making it scalable to big datasets.

5.5 Result Analysis

This section analyses experimental results of (1) RF-list with standard NDCG, (2) RF-list with modified NDCG, (3) RF-hybrid, and (4) RF-pair.

5.5.1 RF-list with Standard NDCG

This subsection uses RF-list with Expected NDCG (cf. Equation 5.3).

We first discuss the results of small to moderate sized datasets, namely TD2004, HP2004, NP2004, Ohsumed, MQ2008 and MQ2007. The average results across the five folds are shown in Table 5.3. Here a significance test is performed on all the test queries of all the five folds, not on individual folds.

We see from Table 5.3 that performance of RF-list is significantly better than RF-point in six out of twelve cases (six cases for each of NDCG@10 and MAP), whereas in the
Chapter 5: Pointwise and Listwise Objective Functions

Procedure: \texttt{Gain\_correctlyRankedPairs}(D, D_{left}, D_{right})

Data: Data at node and left/right child: D, D_{left}, D_{right}

Result: A measure of correctly ranked pairs of D resulting from split.

begin
\begin{align*}
\text{leftDist}, \text{rightDist} & \leftarrow \text{getDocDistributionPerQuery}(D_{left}, D_{right}); \\
\text{if } \text{score}(D_{left}) & < \text{score}(D_{right}) \text{ then} \\
\quad \text{smaller} & \leftarrow \text{leftDist}; \\
\quad \text{bigger} & \leftarrow \text{rightDist}; \\
\text{end} \\
\text{else} \\
\quad \text{smaller} & \leftarrow \text{rightDist}; \\
\quad \text{bigger} & \leftarrow \text{leftDist}; \\
\text{end} \\
\text{correctlyRankedPairs} & \leftarrow 0; \\
\text{for } q \in Q \text{ do} \\
\quad \text{distSmaller} & \leftarrow \text{smaller}[q]; \\
\quad \text{distBigger} & \leftarrow \text{bigger}[q]; \\
\text{for } c_1 = 0 \text{ upto } |\text{distSmaller}| \text{ do} \\
\quad \text{for } c_2 = c_1 + 1 \text{ upto } |\text{distBigger}| \text{ do} \\
\quad \quad \text{correctlyRankedPairs} & \leftarrow \text{correctlyRankedPairs} + w(c_1, c_2)\text{distSmaller}[c_1]\text{distBigger}[c_2]; \\
\text{end} \\
\text{end} \\
\text{return correctlyRankedPairs}; \\
\end{align*}
end

Where the function \texttt{getDocDistributionPerQuery}(D_{left}, D_{right}) computes the (absolute, i.e., non-normalized) relevance label distribution of each query of the two data partitions. As such, \text{leftDist} (\text{rightDist}) contains a distribution for each query of the left (right) child, and \( w(c_1, c_2) \) is any monotonically increasing function of \(|c_1 - c_2|\).

Algorithm 4: Pairwise gain

Table 5.3: Performance comparison among pointwise and listwise approaches. RF-p and RF-l stands for RF-point and RF-list respectively. The \textbf{bold} and \textit{italic} and \textbf{bold} figures denote that the best performance is significant with \( p \)-value less than 0.01 and 0.05 respectively.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Metric</th>
<th>RF-p</th>
<th>RF-l</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ohsumed</td>
<td>NDCG@10</td>
<td>0.4187</td>
<td>0.4377</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.4141</td>
<td>0.4326</td>
</tr>
<tr>
<td>MQ2008</td>
<td>NDCG@10</td>
<td>0.2245</td>
<td>0.2326</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.4706</td>
<td>0.4778</td>
</tr>
<tr>
<td>MQ2007</td>
<td>NDCG@10</td>
<td>0.4368</td>
<td>0.4442</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.4523</td>
<td>0.4606</td>
</tr>
<tr>
<td>TD2004</td>
<td>NDCG@10</td>
<td>0.3521</td>
<td>0.3421</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.2551</td>
<td>0.2549</td>
</tr>
<tr>
<td>HP2004</td>
<td>NDCG@10</td>
<td>0.8068</td>
<td>0.8032</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.7042</td>
<td>0.6910</td>
</tr>
<tr>
<td>NP2004</td>
<td>NDCG@10</td>
<td>0.7955</td>
<td>0.7797</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.6749</td>
<td>0.6342</td>
</tr>
</tbody>
</table>

rest of the cases the difference is not significant. Importantly, RF-list wins in all six cases when a dataset has (1) comparatively more queries, and (2) graded relevance labels (for Ohsumed, MQ2008 and MQ2007 - (cf. Table 2.2), and the performance is even better when this type of dataset is comparatively larger (on MQ2007 as will be evident later when we shall report fold-wise \( p \)-values). Thus these results suggest that, for comparatively larger datasets having graded relevance, RF-list tends to outperform RF-point.

Another aspect regarding these results is, TD2004, HP2004 and NP2004 datasets are highly imbalanced in terms of the ratio of relevant to non-relevant documents (cf. Table
Chapter 5: Pointwise and Listwise Objective Functions

2.2). Future research may reveal whether this aspect has any influence on varying performance across the two algorithms.

Now we discuss the results of big data. For big two datasets, namely MSLR-WEB10K and Yahoo, RF-list takes considerable time to execute. This is because to compute the listwise objective of a split-point, RF-list needs to enumerate all of the current leaves of a tree (as opposed to the case of RF-point where only the current (parent) leaf is involved in deciding the split-point), which involves their sorting and traversing a large number of documents. Hence instead of executing the original version of RF-list, we resort to the 2nd idea of the heuristics mentioned in Section 5.4.4.2 to compare RF-point and RF-list.

The idea was prescribed in Chapter 4 which replaces bootstrap samples with much smaller sized (sub-)samples for learning a tree of an ensemble in the context of LtR. We showed that on small to moderate-sized datasets, using smaller sub-samples per tree not only greatly reduces the computational time, but also results in improvement in performance of the ensemble (due to a reduction in correlation between the trees). For big datasets, however, we did not notice performance improvement with sub-sampling method. Nonetheless, here we can fairly compare the relative performance of sub-sampled versions of both RF-point and RF-list.

In this experiment a sub-sample 5% of the training queries are used to learn each tree – this means that 300 and 1000 queries are used for the MSLR-WEB10K and Yahoo datasets respectively (cf. Table 2.2). Following the notation used in Chapter 4, we call this algorithm RF-point(/list)-S\(y\) where \(y\) indicates the percentage of training queries used to learn a tree. As such, the RF-point(/list) with the default sampling is denoted by RF-point(/list)-S63. When we use only RF-point(/list), it implies RF-point(/list)-S63.

Table 5.4: Results of RF-point-S5 and RF-list-S5 on big datasets. The bold and italic and bold figures denote that the best performance is significant with \(p\)-values less than 0.01 and 0.05 respectively.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Metric</th>
<th>RF-point-S5</th>
<th>RF-list-S5</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSLR-WEB10K</td>
<td>NDCG@10</td>
<td>0.4256</td>
<td>0.4344</td>
</tr>
<tr>
<td></td>
<td>ERR</td>
<td>0.3247</td>
<td>0.3421</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.3432</td>
<td>0.3417</td>
</tr>
<tr>
<td>Yahoo</td>
<td>NDCG@10</td>
<td>0.7373</td>
<td>0.7283</td>
</tr>
<tr>
<td></td>
<td>ERR</td>
<td>0.4532</td>
<td>0.4507</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.6121</td>
<td>0.6029</td>
</tr>
</tbody>
</table>

Table 5.4 shows the results. For MSLR-WEB10K dataset, RF-list-S5 wins over its pointwise counterpart in terms of NDCG@10 and ERR (which are heavily influenced by the top few ranks). In contrast to the findings of MSLR-WEB10K, on the Yahoo dataset RF-list-S5 is outperformed by RF-point-S5.
In summary, the findings emerged from this subsection suggest that the relative performance of RF-list and RF-point vary across different datasets.

A concern here could be whether the poor performance of RF-list-S5 on the Yahoo dataset is related to small sub-sample size. That is to say, given the results of RF-point-S5 and RF-list-S5 on the Yahoo data, how likely is it that RF-list-S63 will win over RF-point-S63? We postpone this discussion until we conduct an in-depth analysis of the hybrid algorithm in Section 5.5.3 which will provide us with a reliable indication as to what would be the case if we were to be able to run RF-list with standard setting (i.e., 63% sub-sampling) on Yahoo dataset.

5.5.2 RF-list with Modified NDCG Discount

This subsection uses RF-list with Equations 5.6 and 5.7.

When using Equation 5.6, we choose $\alpha \in \{0.1, 1.0, 4.0\}$. Results are given in Table 5.5. We see that the setting with $\alpha = 0.1$ (when influence of top few ranks are reduced) yields better performance than that of standard NDCG. The poor performance of the setting with $\alpha = 4.0$ indicates that giving (unnecessary) high importance towards the top few ranks does not work well, which was conjectured in Section 5.4.3.2.

As for the second method (i.e., Equation 5.7) we choose $\beta \in \{0.01, 0.1\}$. A similar trend to the previous result is found because $\beta = 0.01$ or 0.1 means that the influence of top few ranks is less than that of standard NDCG.

To summarize, this experiment discovers that (1) as conjectured in Section 5.4.3.2, the amount of influence of top ranks of a ranked list is indeed an important factor for RF-list, and (2) a reducing the influenced in a controlled manner yields slightly better performance because it possibly causes the learning to be stable by reducing the sensitivity of the top few ranks of a list. In general Equation 5.7 has been found to be better than Equation 5.6 which is also justifiable as the utility of Equation 5.7 is established by a theoretical analysis.

We note that for Yahoo dataset, although the listwise objective function with modified discounted NDCG (Equation 5.7) outperforms the listwise objective with standard NDCG, it is still has significantly poorer performance than its pointwise counterpart mentioned in Table 5.4.

5.5.3 RF-hybrid

Table 5.6 shows the results for RF-hybrid. The following observations are made:
Table 5.5: Results of RF-list-S5 with different discount functions. Pairwise significance test is performed treating the standard NDCG (with discount 1.0) as baseline. The **bold and italic** and **bold** figures denote that the performance difference is significant with p-value less than 0.01 and 0.05 respectively. † means significantly poorer than baseline.

<table>
<thead>
<tr>
<th>Metric</th>
<th>0.1</th>
<th>1.0 (Standard NDCG)</th>
<th>4.0</th>
<th>0.01</th>
<th>0.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data: MSLR-WEB10K (Fold 1)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.4333</td>
<td>0.4305</td>
<td>†0.4203</td>
<td>0.4322</td>
<td>0.4319</td>
</tr>
<tr>
<td>ERR</td>
<td>0.3435</td>
<td>0.3423</td>
<td>†0.3351</td>
<td>0.3419</td>
<td>0.3406</td>
</tr>
<tr>
<td>MAP</td>
<td>0.3457</td>
<td>0.3424</td>
<td>†0.3390</td>
<td>0.3478</td>
<td>0.3465</td>
</tr>
</tbody>
</table>

Table 5.6: Results of RF-point and RF-list-S5/RF-hybrid-L6. Significance test is conducted between (1) RF-point-S5 and RF-list-S5, and (2) RF-point and RF-hybrid-L6. The **bold and italic** and **bold** figures denote that the performance difference is significant with p-value less than 0.01 and 0.05 respectively. † means significantly poorer.

<table>
<thead>
<tr>
<th>Metric</th>
<th>RF-point-S5</th>
<th>RF-point</th>
<th>RF-list-S5</th>
<th>RF-hybrid-L6</th>
</tr>
</thead>
<tbody>
<tr>
<td>Data: MSLR-WEB10K</td>
<td>Sub-sample size per tree</td>
<td>Sub-sample size per tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5%</td>
<td>63%</td>
<td>5%</td>
<td>63%</td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.4256</td>
<td>0.4415</td>
<td>†0.4344</td>
<td>0.4502</td>
</tr>
<tr>
<td>ERR</td>
<td>0.3247</td>
<td>0.3421</td>
<td>†0.3421</td>
<td>0.3492</td>
</tr>
<tr>
<td>MAP</td>
<td>0.3432</td>
<td>0.3543</td>
<td>0.3417</td>
<td>0.3570</td>
</tr>
<tr>
<td>Data: Yahoo</td>
<td>Sub-sample size per tree</td>
<td>Sub-sample size per tree</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>5%</td>
<td>63%</td>
<td>5%</td>
<td>63%</td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.7294</td>
<td>0.7290</td>
<td>†0.7257</td>
<td>0.7325</td>
</tr>
<tr>
<td>ERR</td>
<td>0.4511</td>
<td>0.4510</td>
<td>0.4503</td>
<td>0.4520</td>
</tr>
<tr>
<td>MAP</td>
<td>0.6032</td>
<td>0.6028</td>
<td>†0.5989</td>
<td>0.6063</td>
</tr>
</tbody>
</table>

- For MSLR-WEB10K dataset, RF-hybrid-L6 (i.e., having listwise splitting up to level 6 of a tree) wins in all three metrics. (This affirms the efficacy of using RF-point/list-S5 for measuring relative performance.) We can, therefore, conclude that if sufficient computational resources are not available to apply RF-list, RF-hybrid is a viable option to incorporate the benefit of the listwise objective function (if applicable, e.g., in MSLR-WEB10K). The amount of listwise splits will depend on the availability of the resources – the more resources are available, the deeper of the trees can be learnt with listwise splitting. Note that this idea of hybridization is likely to work for any other objective function with higher computationally complexity.

- For the Yahoo data, the RF-hybrid-L6 performs similar to (ERR) or slightly poorer (NDCG@10 and MAP) than its pointwise counterpart. This, again, supports the finding mentioned in Table 5.4 which showed that the listwise objective function is not effective for this dataset, which implies that guiding the data partitions by listwise splits (during the early nodes) does not help here. This further corroborates the hypothesis that RF-hybrid is indeed able to incorporate diverse splitting criteria in a single tree.
In order to be certain as to whether RF-hybrid does indeed incorporate the benefit of higher discriminative power of the listwise objective function into a tree, we now conduct the following experiment. We gradually increase \( L \) (i.e., the level of a tree up to which listwise splitting is performed), and examine the performance trend. If a dataset benefits from listwise objective function (e.g., MSLR-WEB10K), we hope to see a gradually increasing curve for performance.

In Figure 5.4 we show plots of performance and training time for three datasets, namely MSLR-WEB10K, MQ2007 and Yahoo. For the big datasets, since we cannot learn an entire tree using listwise objective function with the computational resources available to us, we stop at level 6 (out of approximately 17 and 16 for MSLR-WEB10K and Yahoo respectively).

From the plots of Figure 5.4 we see that on the MSLR-WEB10K data, performance gradually increases (ignoring the minor fluctuations) with increasing depth of listwise
splitting in a tree. All of these differences between RF-hybrid and RF-point have been found to be statistically significant over RF-point at $p < 0.01$ level. On MQ2007 dataset, although there is more fluctuation because of its comparatively smaller size, the increasing trend is clearly visible (note that in all three cases the ticks of the y-axis are very close to each other). This behavior was expected for these two datasets as both of them are already shown to have benefitted from the listwise objective function (cf. Tables 5.3 and 5.6).

On the Yahoo dataset, the performance of RF-hybrid remains similar to or becomes slightly poorer than RF-point as we increase the amount of the listwise splitting (no RF-hybrid significantly wins over RF-point). This strongly predicts that had we been able to run the RF-list with standard sub-sampling (i.e., RF-list-S63), performance would not be better than RF-point. This analysis diminishes the concern (expressed in Section 5.5.1) about the effectiveness of sub-sampling method while comparing between RF-point-S5 and RF-list-S5, and we can thus conclude that the mediocre performance of RF-list-S5 on Yahoo dataset is less likely to be related to smaller sub-sample size (per tree).

Table 5.7: Training time improvement for RF-hybrid with sub-sampling method.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Level 4 (RF-hybrid-L4)</th>
<th>Level 6 (RF-hybrid-L6)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>30% Queries 63% Queries</td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.4417 0.4421</td>
<td>0.4442 0.4438</td>
</tr>
<tr>
<td>ERR</td>
<td>0.3457 0.3458</td>
<td>0.3489 0.3474</td>
</tr>
<tr>
<td>MAP</td>
<td>0.3565 0.3562</td>
<td>0.3591 0.3562</td>
</tr>
<tr>
<td>Training time per tree (minute)</td>
<td>137 272</td>
<td>403 672</td>
</tr>
</tbody>
</table>

A question may arise here: can we take the advantage of the sub-sampling idea for RF-hybrid? That is to say, can we reduce training time by using smaller data per tree without compromising performance? Table 5.7 and Figure 5.5 show performance...
and training time (per tree) of RF-hybrid-S30 and RF-hybrid-S63 (i.e., standard sub-sampling) for two levels, namely 4 and 6. We see that for both the levels RF-hybrid-S30 produces similar NDCG@10 to that of RF-hybrid-S63 in the sense that the difference is not found to be statistically significant. The learning time is, however, reduced by a factor of 2 and 1.7 respectively. Thus it is evident that the sub-sampling approach indeed helps to reduce learning time for big datasets without compromising performance, especially when a objective function with higher computationally complexity is employed.

**Further Investigation into Performance Trade-off Between Pointwise and Listwise Objective Functions on Yahoo Dataset.** We now extend our analysis on relative performance of RF-point and RF-hybrid on Yahoo dataset as RF-hybrid was unable to outperform RF-point on this dataset. A possibility is that the increased number of features in the Yahoo data enables the pointwise algorithm to learn a complex hypothesis, thereby mitigating the surmised benefit of the listwise algorithm. The setting of a decisive experiment would be to compare performance of Yahoo and MSLR-WEB10K using modified training sets where we retain only the common features of both. This is, however, not possible as the formulas used to compute the features on the Yahoo dataset were not disclosed. In this investigation we reduce the number of features of the Yahoo dataset (as it is our main focus here) from 519 down to 10, 20, 40, 80, 160 and 320. We then train both RF-point and RF-hybrid-L6 using the modified training sets. The goal is to examine if RF-hybrid-L6 performs better with smaller features. Figure 5.6 shows that although RF-hybrid numerically marginally wins over RF-point as the number of features is reduced from 519 down to 10, none of the differences are statistically significant at 0.01 level. This experiment thus indicates that the larger number of features in Yahoo dataset (with respect to MSLR-WEB10K) is probably not the reason for the reduced effectiveness of the RF-hybrid over the pointwise one; but again, we cannot confirm this unless the features of the Yahoo dataset are disclosed. Thus the question as to what is the specific reason for which RF-hybrid could not outperform RF-point on Yahoo dataset still remains open.
### 5.5.4 RF-pair

Table 5.8: Comparison between RF-point, RF-pair, and RF-list/hybrid.

<table>
<thead>
<tr>
<th>Data: MSLR-WEB10K (Fold 1)</th>
<th>Algorithms</th>
<th>RF-point</th>
<th>RF-pair</th>
<th>RF-hybrid-L6</th>
<th>RF-pair-weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.4379</td>
<td>0.4010</td>
<td>0.4442</td>
<td>0.4069</td>
<td></td>
</tr>
<tr>
<td>ERR</td>
<td>0.3405</td>
<td>0.3109</td>
<td>0.3489</td>
<td>0.3173</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.3542</td>
<td>0.3309</td>
<td>0.3591</td>
<td>0.3337</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: Yahoo</th>
<th>Algorithms</th>
<th>RF-point</th>
<th>RF-pair</th>
<th>RF-hybrid-L6</th>
<th>RF-pair-weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.7538</td>
<td>0.7182</td>
<td>0.7525</td>
<td>0.7198</td>
<td></td>
</tr>
<tr>
<td>ERR</td>
<td>0.4594</td>
<td>0.4461</td>
<td>0.4500</td>
<td>0.4468</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.6278</td>
<td>0.5948</td>
<td>0.6263</td>
<td>0.5963</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: MQ2007</th>
<th>Algorithms</th>
<th>RF-point</th>
<th>RF-pair</th>
<th>RF-list</th>
<th>RF-pair-weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.4368</td>
<td>0.4333</td>
<td>0.4442</td>
<td>0.4428</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.4523</td>
<td>0.4632</td>
<td>0.4606</td>
<td>0.4619</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: MQ2008</th>
<th>Algorithms</th>
<th>RF-point</th>
<th>RF-pair</th>
<th>RF-list</th>
<th>RF-pair-weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.2245</td>
<td>0.2285</td>
<td>0.2326</td>
<td>0.2302</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.4706</td>
<td>0.4755</td>
<td>0.4778</td>
<td>0.4777</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: Ohsumed</th>
<th>Algorithms</th>
<th>RF-point</th>
<th>RF-pair</th>
<th>RF-list</th>
<th>RF-pair-weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.4187</td>
<td>0.4477</td>
<td>0.4377</td>
<td>0.4404</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.4141</td>
<td>0.4426</td>
<td>0.4326</td>
<td>0.4408</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: TD2004</th>
<th>Algorithms</th>
<th>RF-point</th>
<th>RF-pair</th>
<th>RF-list</th>
<th>RF-pair-weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.3521</td>
<td>0.3529</td>
<td>0.3421</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.2551</td>
<td>0.2520</td>
<td>0.2549</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: HP2004</th>
<th>Algorithms</th>
<th>RF-point</th>
<th>RF-pair</th>
<th>RF-list</th>
<th>RF-pair-weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.8068</td>
<td>0.7877</td>
<td>0.8032</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.7042</td>
<td>0.7001</td>
<td>0.6910</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: NP2004</th>
<th>Algorithms</th>
<th>RF-point</th>
<th>RF-pair</th>
<th>RF-list</th>
<th>RF-pair-weighted</th>
</tr>
</thead>
<tbody>
<tr>
<td>Algorithms</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.7955</td>
<td>0.7518</td>
<td>0.7797</td>
<td>N/A</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.6754</td>
<td>0.6205</td>
<td>0.6342</td>
<td>N/A</td>
<td></td>
</tr>
</tbody>
</table>

Table 5.8 shows results for RF-pair, along with that of the pointwise and listwise algorithms for ease of comparison. With the un-weighted version, on smaller datasets, in four out of six datasets RF-pair performs slightly better than that of RF-point, and in two of these four cases its performance is less than that of RF-list (cf. Table 5.3). For big datasets, accuracy of RF-point is much better than RF-pair – we hypothesize that the mediocre performance of RF-pair is due to the greedy nature of the algorithm. In the weighted version, we assign the algebraic difference between the relevance labels of the two documents at hand as their weight. We see that in general no significant improvement is found by the weighted version. (Note that for datasets with binary relevance labels, namely TD2004, NP2004 and HP2004, the weighted version is not applicable.)

In general, RF-pair has been found to be (1) not considerably better than RF-list on most of the smaller datasets, and (2) much poorer than RF-point on big datasets. For these reasons we do not further investigate RF-pair.
5.6 Further Analysis of Splitting Criterion

As explained throughout the chapter that the objective function of RF-list is tailored to reduce the ranking error rate as opposed to that of RF-point where the goal is to minimize misclassification rate. However, we see from the experimental results that although in most of the datasets (namely Ohsumed, MQ2008, MQ2007 and MSLR-WEB10K) its performance has been found to be better than its pointwise counterpart, the margin is not that large, and importantly, it did not win in the Yahoo dataset which is a crucial one as it has been a part of a commercial search engine. In this section we further investigate the relative performance of the two objective functions from three perspectives, namely the importance of splitting criterion, overfitting, and individual tree strength.

5.6.1 Importance of Splitting Criteria

We aim to better understand as to how important the splitting criteria is in developing effective partitions of feature space for LtR. That is, does making it more adaptive to the evaluation metric (as done in RF-list) helps?

5.6.1.1 LtR With Completely Randomized Trees

Here we employ the opposite motivation of RF-list. The idea of RF-list was to move from optimising a surrogate metric (miss-classification rate) to optimising the actual metric of interest (NDCG). Now we design an objective function that is independent from any metric.

A random forest is, in fact, not a single algorithm; rather it can be viewed as a general framework where some dimensions can be altered that results in a class of algorithms. A large number of variations of Breiman’s random forest (which we sometimes call the standard random forest) have been investigated in the literature.\(^6\) For our purpose we need to select a version where the splitting criterion is less adaptive to the training data, i.e., where the amount of randomness is high. To this end, we choose an algorithm which has been widely used for classification and regression, both in theoretical (e.g. [59], [137]),

\(^6\)The list of references is too long to mention here, readers can look into [32] and the references therein.
Below we give a brief description of the algorithm which we call RF-rand.

In RF-rand, the splitting criterion is completely non-adaptive to the training data in the sense that the training phase does not depend on the labels of the training instances – the labels are only used to assign scores to the final data partitions. To split a node, a feature is randomly selected. Then to select the cut-off value (along the selected feature) of the available data (at that node), there is a number of choices in the literature from which we choose the following: a random value is chosen between minimum and maximum values of the selected feature [62].

As for the data per tree, most of the variants of RF-rand learn a tree from the entire training set instead of bootstrapped sample (e.g., [62]) in order to reduce bias and variance of individual trees. We, however, think that in our case it is sufficient to use bootstrapped sampling (without replacement) because the datasets in our case are already sufficiently large as opposed to those works. Moreover, we did perform some experiments with full samples, and found the results to be similar. So we report the results of bootstrapping (without replacement, and as before, at query-level).

A point here to note is, although many variations of standard random forest are available in the literature, we have found only one work [63] which uses one of those variations (not our selected one) for LtR task (with much success). As such, the experiment of this section is not a reproduction of an existing technique, rather it does present undiscovered results in the context of LtR as will be evident next.

5.6.1.2 Results and Discussion

Table 5.9: RF-point Vs RF-rand on small to moderate datasets.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Metric</th>
<th>RF-point</th>
<th>RF-rand</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ohsumed</td>
<td>NDCG@10</td>
<td>0.4187</td>
<td>0.4332</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.4141</td>
<td>0.4312</td>
</tr>
<tr>
<td>MQ2008</td>
<td>NDCG@10</td>
<td>0.2245</td>
<td>0.2232</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.4706</td>
<td>0.4742</td>
</tr>
<tr>
<td>MQ2007</td>
<td>NDCG@10</td>
<td>0.4368</td>
<td>0.4293</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.4523</td>
<td>0.4480</td>
</tr>
</tbody>
</table>

Although our main interest here is the big datasets, we first report results of some smaller datasets in Table 5.9. We see that for Ohsumed, MQ2008 and MQ2007 datasets, in general the performances of RF-point and RF-rand are comparable to one another. As

---

7This version of random forest has been widely studied mainly for the following two reasons. (1) Standard random forest, despite its colossal empirical success over a range of tasks, to this date lacks the theoretical evidence of its consistency [38]. (Although very recent works by Wager [154] and Scornet et al. [138] prove consistency of a very close version of it.) It is comparatively easier to theoretically analyse (for example, to prove consistency of) this variation of random forest as done by Scornet [137] and the references therein, Biau [11], Denil et al. [38], Biau et al. [12]. (2) This version is computationally much faster.
Table 5.10: Big datasets with RF-point, RF-rand and a complete random ranking.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Metric</th>
<th>RF-point</th>
<th>RF-rand</th>
<th>Random</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSLR-WEB10K</td>
<td>NDCG@10</td>
<td>0.4445</td>
<td>0.3978</td>
<td>0.1893</td>
</tr>
<tr>
<td></td>
<td>ERR</td>
<td>0.3424</td>
<td>0.2978</td>
<td>0.1588</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.3543</td>
<td>0.3271</td>
<td>0.1796</td>
</tr>
<tr>
<td>Yahoo</td>
<td>NDCG@10</td>
<td>0.7538</td>
<td>0.7389</td>
<td>0.5411</td>
</tr>
<tr>
<td></td>
<td>ERR</td>
<td>0.4594</td>
<td>0.4517</td>
<td>0.2810</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.6278</td>
<td>0.6156</td>
<td>0.4635</td>
</tr>
</tbody>
</table>

for the results of big datasets shown in Table 5.10 (where we also report a random ranking to examine the effectiveness of RF-rand, and later in Section 6.2 we shall compare RF-rand with several state-of-the-art algorithms), we observe that the performance difference between RF-point and RF-rand is stark (with $p < 0.01$).

From this investigation we conclude the following:

- On large LtR datasets, the splitting criterion is important. On smaller datasets, however, the opposite behaviour was observed. This agrees with the findings of the work by Geurts et al. [62] who use classification and regression datasets – which are much smaller than the larger LtR datasets we have used – to show that random splitting method is generally on a par with standard random forests.

- On large datasets, the numerical value of improvement of pointwise objective over random splitting method is much better than the improvement of listwise objective over pointwise objective (if applicable) (cf. Table 5.6). This indicates that the listwise objective function cannot drastically improve (if applicable) the learnt patterns over the pointwise objective. More on this will be discussed in Sections 5.6.3 and 5.6.4.

- Since there is too much randomness in the individual trees of RF-rand, performance is expected to be comparatively more dependent on the ensemble size. The larger the ensemble, the narrower is likely to be the difference between RF-point and RF-rand.

### 5.6.2 Overfitting

When using a listwise algorithm which directly optimises an NDCG-based objective function, a natural concern is whether growing trees with unlimited depth overfits. (Similar concern is expressed by Tan et al. [148].) The core motivation of random forest was that the individual learners need to have low bias, and one way to achieve this is to grow unlimited trees. While it is empirically found that the fully grown trees work very well, studies such as [141] and [93] suggest that the minimum number of instances required
to consider a split \( (n_{\text{min}}) \) can be tuned to gain slight performance improvement in classification and regression, especially for big datasets. Hence in this section we investigate this aspect.

Figure 5.7 shows performance as we increase \( n_{\text{min}} \). In general, reducing \( n_{\text{min}} \) does not improve performance for both pointwise and listwise algorithms for both the datasets. The only exception we found is RF-point-S5 on MSLR-WEB10K where a minor numerical improvement is seen when \( n_{\text{min}} \) is increased from 1 to 5, but this improvement is not statistically significant. The Yahoo data is more sensitive to this change (we performed significance test for every two consecutive \( n_{\text{min}} \) values, and for Yahoo dataset all these differences were found to be significant). Thus this experiments suggest that unlimited trees do not overfit.

5.6.3 Strength of Individual Trees

In order to better understand the generalization performance of RF-point and RF-list, in this section we compare the strength of individual trees grown in a pointwise and listwise manner. The term “strength” was coined by Breiman [15] to indicate the predictive power of the individual classification trees. In our context we measure this predictive power in terms of NDCG.

To measure the strength of a single tree while it is growing, we compute the Expected NDCG of a tree after every split. To get reliable estimate we take the average of a particular value over 20 trees of an ensemble.

Since the sequence of these strength values (while a tree is growing) depends on the order of splits, to ensure fairness in comparison between RF-list (with breadth-first node enumeration) and RF-point, we implement a breadth-first version of RF-point.\(^8\)

\(^8\)As explained in Section 5.4.3.3 that data partitions of RF-point do not depend on the order of node exploration.
Figure 5.8 shows the plots of all of eight datasets using both training and test sets. Along the x-axis is the order of nodes as being expanded in a tree, and the average strength are along the y-axis. For instance, the first point of a curve is the Expected NDCG (averaged over 20 trees) after the root of a tree is split.

It can be seen from the plots of Figure 5.8 that the strength follow similar pattern across
all the datasets which is as follows. In the early splits, Expected NDCGs of a tree of RF-list are higher, but the curve of RF-point catches up that of RF-list after some splits (which varies from dataset to dataset). The reason for this pattern is due to the fact that a tree in RF-list directly maximizes NDCG. However, in RF-point when the nodes are sufficiently pure, i.e., contain instances of mostly one label (which happens after a tree has grown sufficiently deep), the relevant documents are likely to be in such partitions where they are mostly surrounded by other relevant documents, thereby getting higher scores, and in turn, causing the Expected NDCGs to be higher.

Let us not be confused here that the procedure for computation of NDCG values here are not the same as those calculated after building a complete ensemble.

Thus these plots reveal that the individual trees of the two algorithms, although differ in the form of objective functions, essentially learn similar patterns of the training data given that sufficiently deep trees are grown – at that point individual trees of the two algorithms become similarly strong. This explains to some extent as to why the performances of the two algorithms does not differ by a large margin.

As a side note, we have also compared the strength of trees of RF-list with breadth-first and depth-first enumeration, and found, as expected, that the the curves of RF-list-breadth goes up quite early, but as a tree grows deeper, RF-list-depth catches up. This further strengthens our conjecture that breadth-first exploration is a natural choice for RF-list. For brevity, we do not show those plots here.

Before concluding this section, below we briefly discuss two points in support of our conjecture made above that the deeper the trees, the less distinction between the effect of pointwise and listwise objective functions is observed.

1. **Adding a Termination Criterion.** Figure 5.8 suggested that the shorter the trees of an ensemble, the more beneficial the listwise algorithm would be (as compared to the pointwise algorithm). In other words, if the listwise algorithm wins over its pointwise counterpart (e.g. on MSLR-WEB10K), then the performance difference between pointwise and listwise algorithms is expected to increase as shorter trees are learnt. We expected to observe the opposite behavior for the Yahoo dataset where listwise algorithms could not improve performance. To validate our conjecture, we exploit the parameter $n_{\text{min}}$ which is the minimum number of instances to split a node. This parameter has been widely used in the theoretical studies such as [93] as a termination criterion. In Figure 5.9 we plot the performance difference between RF-point-S5 and RF-list-S5, and it shows that our conjecture holds mostly true (note the negative $y$-axis of the left plot).
2. Statistics of Individual Trees. Table 5.11 examines the (1) average number of leaves of a tree, (2) average leaf size, and (3) average leaf score of the MSLR-WEB10K (Fold 1) and Yahoo datasets (the performance of these settings were given in Table 5.4). These values suggest that RF-point needs comparatively deeper trees to compete (be it either when it wins (Yahoo) or loses (MSLR-WEB10K)) with RF-list.

Table 5.11: Tree size, leaf size and leaf score of the trees of RF-point-S5 (p-S5) and RF-list-S5 (l-S5), averaged over the trees of the ensemble. The performance of these settings were given in Table 5.4.

<table>
<thead>
<tr>
<th>Dataset</th>
<th># Leaves</th>
<th># Instances in a Leaf</th>
<th>Leaf Score</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p-S5</td>
<td>l-S5</td>
<td>p-S5</td>
</tr>
<tr>
<td>MSLR-WEB10K</td>
<td>11770</td>
<td>6190</td>
<td>3.09</td>
</tr>
<tr>
<td>Yahoo</td>
<td>8597</td>
<td>6366</td>
<td>2.76</td>
</tr>
</tbody>
</table>

Local Optimality During Splitting. A further question may arise from the statistics of the trees of the two ensembles (cf. Table 5.11): since the deeper size of the trees of RF-point as compared to the trees of RF-list is likely to be a cause for the betterment of its performance, do the shallower trees in RF-list harness their explorative power? To answer to this question, we conduct a pilot experiment on Yahoo dataset as in this dataset RF-list-S5 could not improve performance over RF-point-S5. A tree growing process may get stuck in a local optima because if there is no gain for splitting a node, it is not split further. In our experiment, we relax this terminating condition as follows: if no gain is found for splitting a node, we still give it a chance to split (provided it contains sufficient instances) based on a pre-defined probability which we call branch probability. We choose 0.3 and 0.5 for our experiment. Table 5.12 shows the results. We observe two aspects: (1) the performance of RF-list-S5 slightly yet steadily increases with increasing branch probability, while RF-point-S5 is largely invariant to this change of the splitting criterion, and (2) the (average) tree size of RF-list-S5 steadily increases (which means that breaking the local optimality eventually causes more meaningful splits down a tree), while RF-point-S5 is, again, is not affected. Thus this investigation further supports our
conjecture that the relatively deeper trees of RF-point indeed help its performance to improve.\footnote{Since the performance improvement of the listwise algorithm found from this investigation is very slight, we do not practice this approach further.}

**Table 5.12:** Performance comparison between RF-point-S5 (p-S5) and RF-list-S5 (l-S5) with different branching probabilities on Yahoo dataset. Pairwise significance test is conducted by treating the setting with branch probability = 0.0 as the baseline (within each of the two groups).

<table>
<thead>
<tr>
<th>Metric</th>
<th>Branch probability: 0.0</th>
<th>Branch probability: 0.3</th>
<th>Branch probability: 0.5</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>p-S5 l-S5</td>
<td>p-S5 l-S5</td>
<td>p-S5 l-S5</td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.7373 0.7283</td>
<td>0.7361 0.7300</td>
<td>0.7366 0.7308</td>
</tr>
<tr>
<td>ERR</td>
<td>0.4532 0.4507</td>
<td>0.4523 0.4515</td>
<td>0.4527 0.4513</td>
</tr>
<tr>
<td>MAP</td>
<td>0.6121 0.6027</td>
<td>0.6140 0.6040</td>
<td>0.6116 <strong>0.6045</strong></td>
</tr>
<tr>
<td>Tree size</td>
<td>8597 6366</td>
<td>8574 7830</td>
<td>8589 9241</td>
</tr>
</tbody>
</table>

### 5.6.4 Remarks

The entropy-based objective function tries to isolate the similar class instances (i.e., of the same relevance label) in the leaf nodes of a tree. As for the NDCG-based objective, it also achieves a better gain if the leaf nodes of a tree which have predominantly relevant documents also have less number of non-relevant documents (and vice versa), because, in such cases, the scores of these nodes will be higher, and hence the resultant NDCG will be higher. In addition, using the same reasoning, the listwise objective function prefers to create a child node containing, for example, a lot of instances of label 2 and a few instances of label 1 to a lot of instances of label 2 and a few instances of label 0 – simply because the latter partition drags the score of the relevant leaf to be lower than it should be.

In essence, the listwise splitting criterion, like its pointwise counterpart, attempts to improve the purity of the nodes. In addition, it attempts to capture the ordinal relationship between the relevance labels. Hence the following conjecture emerges from this research: any nearest-neighbor-based algorithm, if the average of the relevance labels of the instances of a region (i.e., data partition) is assigned as the score of that region, will work well for LTR task. This may be an interesting direction for future work. That is why, we believe, RF-rand yielded reasonable performance despite its random splitting (cf. Section 5.6.1.1). In fact, a random forest has already been interpreted by Lin and Jeon \cite{93} as a special case of \(k\)-nearest neighbor method.

The above discussion may explain the effect of deep trees. When a tree is sufficiently deep, the entropy-based objective function then manages to create such small partitions that the relevant documents are isolated from the non-relevant ones to a reasonable extent. As compared to entropy-based splitting, an NDCG-based splitting probably
takes, in some sense, a harsher decision in the early partitions with an aim to quickly make the relevant nodes more pure, and moreover, with an emphasis on their ordinal relationship as explained above.

Finally, we quote a comment made by Chepelle and Cheng [25] after their experience in the Yahoo LtR Challenge:

*Most learning to rank papers consider a linear function space for the sake of simplicity. This space of functions is probably too limited and the above reasoning explains that substantial gains can be obtained by designing a loss function specifically tuned for ranking. But with ensemble of decision trees, the modeling complexity is large enough and squared loss optimisation is sufficient.*

This comment agrees to our explanations as to why a simple ensemble of randomized trees such as RF-point works quite well in practice for learning a good ranking function. Hence for randomized tree ensemble based algorithms the improvement of the splitting criteria may not have a very large impact on performance. Another explanation is that for the big datasets, we do not yet have enough data to be able to distinguish well between the better learner amongst the pointwise and listwise algorithms.

### 5.7 Discussion

We have developed a random forest based listwise algorithm which directly optimises an IR evaluation metric of choice. For large datasets, a listwise algorithm can be computationally prohibitive, so we developed a hybrid algorithm which uses listwise splitting in earlier stages of tree construction and pointwise splitting in the latter stages. In addition, we investigated the effect of modifying the discount factor used in NDCG when employing it as a listwise objective function. We also devised an RF-based pairwise algorithm which greedily maximizes correctly ranked pairs of documents.

We further investigated the relative generalization performance of the listwise and pointwise approaches. Firstly, we examined the efficacy of an unsupervised partitioning strategy (using random splitting) in order to quantify the importance of splitting criterion. Secondly, we have investigated the overfitting issue in both the pointwise and listwise algorithms. Thirdly, we analysed the predictive accuracy (strength) of individual trees learnt by the pointwise and listwise algorithms.

The key findings of this chapter are summarized below:
• Experimental results have shown the listwise objective-based LtR algorithm to outperform its pointwise counterpart across several datasets.

• The hybrid approach resulted in significant performance improvement in one of the two large datasets investigated. Also, we have shown on multiple datasets that the hybrid trees can successfully blend different splitting criteria.

• By reducing the amount of influence of top ranked documents in NDCG formulation, we have been able to slightly improve the generalization performance of the listwise algorithm.

• The pairwise algorithm resulted in small improvement over the pointwise algorithm on most of the smaller datasets. However, its performance has been found to be poorer than the listwise algorithm. Moreover, on the big datasets the pointwise algorithm performed much better than its pairwise counterpart.

• As for the random splitting criterion, results on the smaller datasets indicated that the splitting criteria may not be most important aspect for controlling the error rate, while for the larger datasets the splitting criteria did appear to have a significant effect on performance. On smaller datasets the random splitting-based algorithm performed better than anticipated.

• When using an unlimited tree, overfitting has not been found to be a serious concern in both the pointwise and listwise algorithms.

• Our experiments have revealed that the predictive power of individual trees of pointwise algorithms becomes more similar to that of the listwise algorithms as the trees are grown deeper.

5.8 Conclusion

Random forest based algorithms provide an inherently parallelizable solution to the learning-to-rank problem. While these algorithms have demonstrated competitive performance in comparison with other (oftentimes more complicated) techniques, their performance trade-offs were still largely unknown. An attempt has been made in this chapter to rectify this, by examining the effect of pointwise and listwise splitting criteria in the context of learning-to-rank.

Several interesting directions for future research have emerged out of this work:
• Analysing different characteristics of the datasets such as relevance label distribution, sparsity of the data, to understand their impact on the performance difference of pointwise and listwise algorithms.

• Investigating nearest-neighbor based classification/regression algorithms for LtR as discussed in Section 5.6.4.

• Investigating alternative methods for building a hybrid pointwise-listwise objective function.

• Improving the treatment of missing values in a training set.

• Tuning the parameter for controlling the number of candidate features considered at each node of a tree.

In the next chapter we focus on comparison between RF-based algorithms and other state-of-the-art LtR algorithms.
Chapter 6

Detailed Comparison with State-of-the-Art Rank-Learners

In this thesis our primary goal is to investigate RF-based LtR algorithms. However, comparing performance of these algorithms with other state-of-the-art algorithms would help us better understand the practical applicability of the former group of systems. This is the main focus of this chapter. Before that, for the sake of comprehensiveness of the thesis, we investigate a few additional topics regarding the performance of RF-based algorithms.

The chapter is organized as follows. In Section 6.1 we investigate four dimensions of RF-based pointwise algorithms. Section 6.2 conducts the performance comparison between the investigated RF-based LtR algorithms and other relevant state-of-the-art algorithms.

6.1 Determining the Right Settings of RF-based Algorithms

This section conducts four distinct investigations.

6.1.1 Motivation

As discussed in Section 2.7, a random forest has a number of appealing characteristics that makes it popular among the researchers of versatile fields. A notable such characteristic is that it usually performs quite well with the default parameter settings. That being said, since in LtR the random forest has not been thoroughly investigated in the literature, it is tempting to check if this conjecture holds true. Hence in this section
we conduct four different investigations regarding the settings of an RF-based LtR algorithm. Due to computational constraints, we conduct these experiments with the RF-based pointwise algorithms. In the comparison that follows this section, however, we do evaluate both the pointwise and listwise systems.

6.1.2 Contributions

The following contributions are made in the first half, namely Section 6.1 of this chapter:

1. We compare the efficacy of classification and regression settings with an RF-based pointwise algorithm. The existing literature does not offer an explicit guideline as to which one is preferable (if at all).

2. We examine the effect of using the absolute relevance labels versus a mapped relevance labels. Again, the current literature does not explore this area extensively.

3. We control the model complexity using the number of features considered at each node of a tree (denoted by $K$). In chapter 4 an indication was found that $K$ could have an influence in performance, thereby warranting further investigation.

4. Instead of using the average of the predictions of the trees of a random forest, we investigate the performance of a weighted random forest where a weight is assigned to a tree according to its individual ranking performance.

6.1.3 Classification or Regression?

This section investigates as to whether RF-based pointwise algorithms should be used with a classification or a regression setting.

6.1.3.1 Motivation and Methodology

Mohan et al. [109] use a random forest consisting of regression trees in a pointwise fashion to evaluate the MSLR-WEB10K and Yahoo datasets. Geurts and Louppe [63] employ a variant of random forest, namely, Extremely Randomized Trees [62], to evaluate the Yahoo dataset, and their findings do not reveal any significant performance difference between the classification and regression settings. None of these works examines performance on other (smaller) LtR datasets. For the sake of thoroughness in this thesis, we examine performance of a random forest with both the classification and regression settings, and evaluate on eight datasets. The classification model was introduced in Section 3.5. The regression setting treats the relevance judgements (e.g. 0 - 4)
as the target variable, and then minimizes the sum of squared error of the data at a node during a split. The objective function of this algorithm is described in Algorithm 5.

**Algorithm 5:** Gain in the regression setting.

Table 6.1: RF-point with classification (RF-p-cla) versus regression (RF-p-regr) settings. For two larger datasets (MSLR-WEB10K and Yahoo), the *bold* and italic and *bold* figures denote that the best performance is significant with $p$-value less than 0.01 and 0.05 respectively. For smaller datasets, an average over 5 independent runs is reported (and each run is the result of 5-fold cross-validation), and the winning value is given in *italic* font.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Metric</th>
<th>RF-p-regr</th>
<th>RF-p-cla</th>
<th>RF-p-regr</th>
<th>RF-p-cla</th>
</tr>
</thead>
<tbody>
<tr>
<td>MSLR-WEB10K</td>
<td>NDCG@10</td>
<td>0.4512</td>
<td>0.4451</td>
<td>0.7554</td>
<td>0.7538</td>
</tr>
<tr>
<td></td>
<td>ERR</td>
<td>0.3505</td>
<td>0.3434</td>
<td>0.4603</td>
<td>0.4594</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.3276</td>
<td>0.3544</td>
<td>0.6290</td>
<td>0.6278</td>
</tr>
<tr>
<td>Yahoo</td>
<td>NDCG@10</td>
<td>0.4345</td>
<td>0.4360</td>
<td>0.7227</td>
<td>0.7234</td>
</tr>
<tr>
<td></td>
<td>ERR</td>
<td>0.4515</td>
<td>0.4524</td>
<td>0.4674</td>
<td>0.4693</td>
</tr>
<tr>
<td>MQ2007</td>
<td>NDCG@10</td>
<td>0.4168</td>
<td>0.4396</td>
<td>0.3509</td>
<td>0.3513</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.4265</td>
<td>0.4213</td>
<td>0.2559</td>
<td>0.2574</td>
</tr>
<tr>
<td>Ohsumed</td>
<td>NDCG@10</td>
<td>0.7624</td>
<td>0.7860</td>
<td>0.7578</td>
<td>0.8082</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.6307</td>
<td>0.6647</td>
<td>0.6730</td>
<td>0.7174</td>
</tr>
</tbody>
</table>

### 6.1.3.2 Result Analysis

Table 6.1 shows the results of the eight datasets.\(^1\) As before, the ensemble size is 500, and unpruned trees are learnt. We observe two different outcomes, and the difference can be attributed to the cardinality of the datasets.\(^2\) On the two larger datasets, namely MSLR-WEB10K and Yahoo statistically significant differences are observed with the regression setting being the winner, and the MSLR-WEB10K dataset benefits from using regression more than the Yahoo. For smaller datasets, however, the classification setting almost always wins over the regression setting. Thus this investigation suggests that for

---

\(^1\) Since we shall include both the classification and regression settings in the comparison of different algorithms in Section 6.2, we do not use the validation sets here.

\(^2\) For all of the experiments of this section, for two larger datasets (MSLR-WEB10K and Yahoo), the *bold* and italic and *bold* figures denote that the best performance is significant with $p$-value less than 0.01 and 0.05 respectively. For the smaller datasets, an average over five independent runs is reported (and each run is the result of 5-fold cross-validation), and the winning value is given in *italic* font.
big datasets the regression setting performs better than its classification counterpart, whereas for smaller dataset the opposite behavior is expected.

Guided by the findings of this investigation, for the subsequent experiments of this section we use the regression setting for the two larger datasets, and the classification setting for the remaining six datasets.

6.1.4 Ground Truth Labels: Absolute Relevance or Scaled?

This section examines as to whether better performance could be achieved using the mapped relevance labels instead of absolute relevance.

6.1.4.1 Motivation and Methodology

In a pointwise setting - be it a classification or a regression - the relevance labels assigned by an oracle to the query-document pairs are treated as the target variable, and the learning algorithm tries to predict exactly the target variable. However, since from the perspective of IR metrics the relevant documents are of more importance (to be placed in the top part of the returned ranked list) than the non-relevant ones, a natural idea is to stretch out the differences between the highly relevant documents more than in between the less relevant ones. One such approach is to map the relevance labels $l_{q,i}$ to $2^{l_{q,i}} - 1$. For instance, the labels \{0, 1, 2, 3, 4\} are mapped onto \{0, 1, 3, 7, 15\}.

The data partitions produced by the sum-of-square based objective function (Algorithm 5) are affected if the ground truth labels are modified. As for the classification setting (i.e., when using an entropy-based objective function), although there is no effect of stretching out the labels on the data partitions, the assignment of scores to the final data partitions (i.e., to the leaf nodes of a tree) are affected. That is, in the classification setting the mapped relevance acts as a different output encoding scheme. Hence we experiment with both the classification and regression settings. We henceforth call the standard approach as standard relevance (SR), and the approach of stretching out labels as mapped relevance (MR).

To this end, we briefly discuss two related works. Li et al. [92], while using a gradient boosted framework with both the classification and regression settings on proprietary datasets, study the effectiveness of the SR and MR approaches. They conclude that on their datasets the SR approach performs slightly better than the MR approach with the classification setting. The other study we find has already been discussed earlier, \footnote{Recall that the usual practice has been to use the average relevance of the instances as the score.}
which is by Geurts and Louppe \cite{63}; the authors employ a variant of random forest in both the classification and regression settings on the Yahoo dataset, and examine the effect of using the SR and MR approaches on evaluation phase only (i.e., as different output encoding schemes). Thus the novelty of our investigation here is twofold: (1) investigating a standard random forest framework, and (2) evaluating a range of datasets.

### 6.1.4.2 Result Analysis

Table 6.2: For regression setting, RF-point with standard relevance (RF-p-SR) versus mapped relevance (RF-p-MR) approaches.

<table>
<thead>
<tr>
<th>Metric</th>
<th>MSLR-WEB10K</th>
<th>Yahoo</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.4512 0.4495</td>
<td>0.7554 0.7522</td>
</tr>
<tr>
<td>ERR</td>
<td>0.3505 0.3596</td>
<td>0.4605 0.4608</td>
</tr>
<tr>
<td>MAP</td>
<td>0.3576 0.3508</td>
<td>0.6290 0.6226</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>MQ2007</th>
<th>MQ2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.3574 0.3516</td>
<td>0.3574 0.3516</td>
</tr>
<tr>
<td>ERR</td>
<td>0.4515 0.4514</td>
<td>0.4674 0.4659</td>
</tr>
<tr>
<td>MAP</td>
<td>0.4515 0.4514</td>
<td>0.4674 0.4659</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>MQ2007</th>
<th>MQ2008</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.4515 0.4514</td>
<td>0.4674 0.4659</td>
</tr>
<tr>
<td>ERR</td>
<td>0.4515 0.4514</td>
<td>0.4674 0.4659</td>
</tr>
<tr>
<td>MAP</td>
<td>0.4515 0.4514</td>
<td>0.4674 0.4659</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Ohsumed</th>
<th>HP2004</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.7578 0.7534</td>
<td>0.7624 0.7592</td>
</tr>
<tr>
<td>ERR</td>
<td>0.6730 0.6676</td>
<td>0.6307 0.6321</td>
</tr>
<tr>
<td>MAP</td>
<td>0.6730 0.6676</td>
<td>0.6307 0.6321</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>HP2004</th>
<th>NP2004</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.3513 0.3522</td>
<td>0.7860 0.7686</td>
</tr>
<tr>
<td>ERR</td>
<td>0.4213 0.4186</td>
<td>0.7686 0.7686</td>
</tr>
<tr>
<td>MAP</td>
<td>0.4213 0.4186</td>
<td>0.7686 0.7686</td>
</tr>
</tbody>
</table>

The results of the SR and MR approaches with both the regression and classification settings are given in Tables 6.2 and 6.3 respectively. On the two larger datasets, the MR approach demonstrates a tendency to win over SR approach in terms of ERR, both for classification and regression, whereas in terms of NDCG@10 the opposite behavior is observed. In terms of MAP, SR approach wins in most of the datasets for both classification and regression. On smaller datasets, the SR approach mostly wins over MR approach – both for classification and regression, and in both NDCG@10 and MAP.
Since from this experiment we have not observed any clear benefit of using the mapped relevance approach, for the subsequent experiments we stick to the conventional setting, i.e., using the original relevance labels.

6.1.5 Controlling Individual Tree Complexity

This section investigates the optimal number of candidate features at each node of a tree.

6.1.5.1 Motivation and Methodology

In a random forest, a crucial source of randomization in a tree is achieved by reducing the number of candidate features at each node (denoted by $K$), which has the default value of $\log(M) + 1$ \cite{15}. Wager et al. \cite{155} point out that the more the training data, the less need for randomization, because the goal of randomization is to decrease correlation, which in turn decreases the variance of the ensemble. If the training data are large, then the individual trees have already relatively low variance, thereby in need of less randomization.\footnote{That is why in Chapter 4 we observed that for large datasets the benefit of using smaller sub-sample per tree was less visible than that for smaller datasets – we found that bagging \cite{14} that does not conduct any feature randomization performs well for these the large datasets because individual trees are already less prone to overfitting.}

In this section, we investigate as to how performance is affected while we vary individual tree complexity by controlling $K$. On the one hand, the configuration of $K = 1$ produces highly stochastic trees, which greatly de-correlates the trees but causes a high individual tree variance. On the other hand, the configuration of $K = M$, i.e., using all the features produces comparatively low variance trees, but results in high correlation between the trees. By carefully controlling $K$ we aim to learn optimal amount of individual tree complexity.

6.1.5.2 Result Analysis

We analyze results of the two types of datasets separately.

**Relatively Small Datasets.** In Figure 6.1 we plot the performance as $K$ is varied. We observe that for all the four datasets, in general there is no improvement as $K$ is increased from the default value (i.e., $\log(M) + 1$), rather in some cases there is a tendency of slight degradation of performance. This behavior was expected from our previous discussion that the smaller datasets are mostly benefited from using smaller
K. Hence we conclude that for the smaller datasets the default value of K is among the best choices.\(^5\)

Table 6.4: Varying K with for regression setting on validation set, K ∈ {\(\log(M) + 1\), \(\sqrt{M}\), 1/4\(M\), 1/2\(M\), 3/4\(M\), \(M\)}. Significance test result is performed between every two consecutive settings of K.

Data: MSLR-WEB10K (Fold1)

<table>
<thead>
<tr>
<th>Metric</th>
<th>K=8</th>
<th>K=12</th>
<th>K=34</th>
<th>K=68</th>
<th>K=102</th>
<th>K=136</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.4579</td>
<td>0.4637</td>
<td>0.4695</td>
<td>0.4677</td>
<td>0.4668</td>
<td>✦0.4618</td>
</tr>
<tr>
<td>ERR</td>
<td>0.3694</td>
<td>0.3660</td>
<td>0.3708</td>
<td>0.3707</td>
<td>0.3711</td>
<td>✦0.3670</td>
</tr>
<tr>
<td>MAP</td>
<td>0.3605</td>
<td>0.3695</td>
<td>0.3735</td>
<td>0.3731</td>
<td>0.3715</td>
<td>✦0.3683</td>
</tr>
</tbody>
</table>

Data: Yahoo

<table>
<thead>
<tr>
<th>Metric</th>
<th>K=10</th>
<th>K=23</th>
<th>K=130</th>
<th>K=260</th>
<th>K=390</th>
<th>K=519</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.7464</td>
<td>0.7496</td>
<td>0.7527</td>
<td>✦0.7505</td>
<td>0.7506</td>
<td>0.7496</td>
</tr>
<tr>
<td>ERR</td>
<td>0.4549</td>
<td>0.4559</td>
<td>0.4573</td>
<td>0.4573</td>
<td>0.4565</td>
<td>0.4561</td>
</tr>
<tr>
<td>MAP</td>
<td>0.6199</td>
<td>0.6224</td>
<td>0.6246</td>
<td>0.6255</td>
<td>0.6228</td>
<td>✦0.6211</td>
</tr>
</tbody>
</table>

Table 6.5: With test sets, the best values of K (= 1/4\(M\)) found from validation sets.

Big Datasets. Table 6.4 shows the result of six different assignments of K, namely \(\log(M) + 1\), \(\sqrt{M}\) (this is also practised by some researchers such as Gislason et al. [64]), 1/4\(M\), 1/2\(M\), 3/4\(M\), and \(M\) (this is equivalent to bagging). The results show that among the six settings, for both the datasets the setting of K = 1/4\(M\) yields

\(^5\)Since properties of HP2004 and NP2004 datasets are similar to that of TD2004, we do not conduct further experiments.
the best performance. Also, the performance improvement is more vivid in the MSLR-WEB10K dataset than in the Yahoo. We conjecture that the cause of performance improvement is that the LtR data contains many individually weak features that, when taken together, help in effective decision making (Nayyar et al. [173] also make a similar comment). Hence comparatively large number of features are required to generate useful data partitions.

Finally, Table 6.5 validates the findings emerged from the experiments of validations sets. As expected, the characteristics revealed from the validation set are also observed in test sets.

This investigation thus reveals that while for smaller datasets the default value of $K$ is among the best choices, for larger datasets it would be wise to use a higher $K$ – instead of the usual practice of $K = \log(M) + 1$ or $K = \sqrt{M}$, our experiment finds that $K \approx 1/4M$ gives significantly better performance. In general, the weaker the individual features, the more the possibility that a higher $K$ will work better.

6.1.6 Weighting the Trees

This section examines the effect of assigning weights to individual trees.

6.1.6.1 Motivation and Methodology

The prevalent approach to using a random forest is to compute an un-weighted average of the predictions of individual trees. It is, however, conceivable that the quality of the predictions of all the trees are not the same. Hence we could assign weights to the predictions of the individual trees according to their individual accuracy. Assigning weights to the trees, however, could result in overfitting (by increasing the variance of the ensemble prediction). In this section we investigate a weighted version of the pointwise algorithm that assigns a weight to a tree based on its individual ranking accuracy.

We predict the individual performance of a tree by using the Expected NDCG (introduced in Chapter 5) per tree which is evaluated using the held-out data (i.e., validation set). As such, the weight of the $i$th tree is given by:

$$w_i = \frac{1}{|D_{\text{heldout}}|} \sum_{q \in D_{\text{heldout}}} \mathbb{E}[NDCG(q; s_i(.))],$$

(6.1)

where $\mathbb{E}[NDCG(.)]$ is the Expected NDCG of query $q$ (cf. Equation 5.3) based on the scores $\{s_i(x_q,j)\}_{j=1}^{n_q}$ produced by $i$th tree. The prediction of a test instance is then the
Chapter 6: Detailed Comparison with State-of-the-Art Rank-Learners

weighted average of predictions of individual trees. We call this algorithm RF-point-weighted.

 Regarding the existing works related to our idea, Marko Robnik-Šikonja [131], Winham et al. [158], among others, examine techniques for improving classification/regression accuracy by using various weighting schemes. Another related line of research is to search for a subset of the trees that performs better than the entire ensemble [9, 91, 114]. The novelty of our investigation as compared to these existing works is, we assign weights to the individual trees based on their individual performance on a rank-based evaluation metric, namely NDCG (instead of the classification/regression accuracy).

Table 6.6: Comparison between RF-point and RF-point-weighted.

<table>
<thead>
<tr>
<th>Metric</th>
<th>MSLR-WEB10K</th>
<th>Yahoo</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RF-point</td>
<td>RF-point-weighted</td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.4512</td>
<td>0.4513</td>
</tr>
<tr>
<td>ERR</td>
<td>0.3505</td>
<td>0.3508</td>
</tr>
<tr>
<td>MAP</td>
<td>0.3576</td>
<td>0.3576</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>MQ2007</th>
<th>MQ2008</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RF-point-weighted</td>
<td>RF-point-weighted</td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.4360</td>
<td>0.4360</td>
</tr>
<tr>
<td>MAP</td>
<td>0.4524</td>
<td>0.4524</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>Ohsumed</th>
<th>TD2004</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RF-point-weighted</td>
<td>RF-point-weighted</td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.4306</td>
<td>0.4305</td>
</tr>
<tr>
<td>MAP</td>
<td>0.4213</td>
<td>0.4212</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Metric</th>
<th>NP2004</th>
<th>HP2004</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RF-point-weighted</td>
<td>RF-point-weighted</td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.7860</td>
<td>0.7861</td>
</tr>
<tr>
<td>MAP</td>
<td>0.6647</td>
<td>0.6648</td>
</tr>
</tbody>
</table>

### 6.1.6.2 Result Analysis

Table 6.6 shows the results for the eight datasets. We observe that the accuracy of the two systems (RF-point and RF-point-weighted) are strikingly similar. This is, we conjecture, due to the fact that an ensemble is sufficiently large to offset the negative effect of the relatively poor trees.

Our experiments thus suggest that the standard un-weighted RF-based pointwise approach performs similar to its weighted counterpart.

### 6.1.7 Discussion

The findings of this section are summarized below:

- Using the eight datasets we compare between RF-point with classification and RF-point with regression. We find that for smaller datasets the classification setting
is more robust across different datasets than the regression setting. However, for big datasets the opposite behavior is observed.

- We compare between the use of absolute relevance (i.e. \( l_{q,i} \)) and mapped relevance (i.e., \( 2^{q,i} - 1 \)) – for both classification and regression settings. We find that (1) on the larger datasets there is no clear winning approach, and (2) on the smaller datasets the conventional practice of using the absolute relevance labels marginally wins over the mapped relevance approach.

- We examine the effect of varying tree complexity by controlling \( K \). For smaller datasets we do not find any reliable change in performance. For big datasets, however, \( K \approx \frac{1}{4M} \) yields significantly better performance than the default value (i.e., \( \log(M) + 1 \)). As such, even a non-extensive tuning of \( K \) has been found to improve the performance significantly.

- We investigate a weighted RF-based pointwise algorithm where each tree is assigned a weight based on its individual ranking performance. The investigation suggests that RF-point performs similar to its weighted counterpart.

### 6.2 Comparison with Other Algorithms

In this thesis our primary goal is to investigate the random forest based LtR algorithms. That said, it is tempting to compare these algorithms with other state-of-the-art algorithms, especially the tree ensemble based ones. This section is devoted to such a comparison.

#### 6.2.1 Baselines

We use the following six baselines:

1. Mart [92]: a pointwise algorithm based on gradient boosting which is a tree ensemble method.
2. RankSVM [83]: a pairwise algorithm based on SVM. This has been a popular baseline for many years.
3. AdaRank [166]: a listwise algorithm based on AdaBoost framework which is a tree ensemble based method, although the prevalent implementation uses a single feature instead of a tree.
4. CoorAsc [105]: a listwise algorithm based on coordinate ascent method. We choose this one because a random forest can also be interpreted as an optimization of the objective function in a coordinate-wise fashion.

5. RankBoost [49]: a pairwise algorithm based on AdaBoost framework (i.e., tree ensemble based).

6. LambdaMart [159]: a listwise algorithm based on gradient boosting (i.e., tree ensemble based).

The parameter settings of these algorithms (and a brief description of the ones which were not discussed earlier in the thesis) are given in Appendix E.1. In addition to the results of these algorithms, we also show results of BM25 scorer which is very popular in the IR community as a single feature. 6

Before delving into the analysis of the experimental results, the following three points need to be discussed:

- **Selecting the right number of documents per query (k) in the top-k retrieval phase.**

As discussed in Section 3.3, the choice of k in the top-k retrieval phase - the phase when k documents are retrieved (per query) using a base ranker to prepare the training set (cf. Section 3.1) - is likely to vary the effectiveness of the learning phase [100]. Now the question for us is, are the values for k values for each of the datasets, as decided by the dataset creators (and shown in Table 2.2) appropriate? In other words, will the chosen values for k provide for reliable performance? One of the major findings of the said work of Macdonald et al. [100] is that the navigational queries need a comparatively larger k to allow us to draw a reliable conclusion (from the experimental results) about the LtR algorithms. 7 Amongst our eight datasets HP2004 and NP2004 contain navigational queries, and from Table 2.2 we see that around 1000 documents per query were retrieved, which is termed as “sufficient” by the said work [100]. For informational queries, the same work of Macdonald et al. finds that much smaller sized (as small as 10-20) k works well. So we can conclude that the datasets used in this thesis fulfill the condition for being eligible for LtR experimentation. We note that these datasets have been used for LtR experimentations for a long time.

---

6While the researchers who released the Yahoo dataset did not disclose information regarding the formulas used to compute the feature values, they did mention that a particular feature index corresponded to BM25.

7This is intuitive since navigational queries have very few relevant documents, so setting a higher value for k facilitates the base ranker(s) put those few relevant documents into the training set of rank-learning phase.
• Choosing the right metric for evaluation given a particular user information need. MAP may not be considered as a very effective choice for evaluation for navigational information need, i.e., the queries oftentimes having only one relevant document [101, Sec. 8.4]. NDCG is, however, considered to be a reasonable choice in different scenarios. We still report MAP (along with NDCG@10) on HP2004 and NP2004 datasets (which contain navigational queries) mainly for the sake of comparison with the existing works since a large number of existing works report MAP on these datasets.

• Optimizing the right metric during learning in listwise algorithms. In general the idea of choosing the training metric carefully in a listwise algorithm sounds appealing. He et al. [71] and Robertson [128] speculate that the training metric that is optimized during learning need not to be the same as the evaluation metric. While Donmez et al. [44] contradict this speculation by showing that the training metric should be the same as the test metric given “sufficient” training data are available, Yilmaz and Robertson [170] later put their claim into question by arguing that sufficient training data are not always available in practise. More recently, Macdonald et al. [100] provide further insight into this ongoing academic debate by conducting extensive experiments, and reveal that optimizing the very test metric during training do not necessarily translate into a better effectiveness during evaluation. In particular, they (Macdonald et al.) discover that optimizing ERR cannot improve the effectiveness over the setting of optimizing NDCG. From this discussion we conclude that NDCG has been rightfully a standard choice for optimization among the researchers for a range of user information needs. Hence the fact that RF-list optimizes NDCG should not be a problem for the scenarios where NDCG may not, if any, be considered as the best evaluation metric.

6.2.2 Smaller Datasets

We begin our analysis with the smaller datasets. Results of RF-rand (cf. Section 5.6.1.1), RF-point (with both classification and regression) (cf. Sections 3.5 and 6.1.3), RF-point-S%, i.e., RF-point with reduced sub-sample (cf. Chapter 4), RF-list (cf. Chapter 5), and RF-list-S% are given in Table 6.7. For RF-point(/list)-S%, the sizes of sub-samples are taken from the best values found in Chapter 4. Since the RF-based algorithms are stochastic, we report the average metrics of five independent runs, and recall that each run is an average of five folds. Performances of the six state-of-the-art algorithms are listed in Table 6.8.

The following observations are made regarding the RF-based algorithms:
Table 6.7: Performance of various RF-based algorithms on smaller datasets. Each metric is the average of five independent runs, and each run is the average of five folds. For RF-point(/list)-S%, the sub-sample (per tree) percentage (and size in terms of queries) are (for the six datasets, in the order of top to bottom): 10, 10, 6, 7, 9, and 4. The algorithms are: RF-rand (rfr), RF-point-classification (rfpc), RF-point-classification-S% (rfpcs), RF-point-regression (rfpr), RF-point-regression-S% (rfprs), RF-list (rfl), RF-list-S% (rls).

### Data: MQ2007

<table>
<thead>
<tr>
<th>Metric</th>
<th>RFR</th>
<th>RFPC</th>
<th>RFPCS</th>
<th>RFPR</th>
<th>RFPRS</th>
<th>RFL</th>
<th>RFLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.4314</td>
<td>0.4360</td>
<td>0.4439</td>
<td>0.4345</td>
<td>0.4423</td>
<td>0.4433</td>
<td>0.4442</td>
</tr>
<tr>
<td>MAP</td>
<td>0.4493</td>
<td>0.4524</td>
<td>0.4604</td>
<td>0.4515</td>
<td>0.4588</td>
<td>0.4613</td>
<td>0.4611</td>
</tr>
</tbody>
</table>

### Data: MQ2008

<table>
<thead>
<tr>
<th>Metric</th>
<th>RFR</th>
<th>RFPC</th>
<th>RFPCS</th>
<th>RFPR</th>
<th>RFPRS</th>
<th>RFL</th>
<th>RFLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.2228</td>
<td>0.2234</td>
<td>0.2286</td>
<td>0.2227</td>
<td>0.2269</td>
<td>0.2313</td>
<td>0.2320</td>
</tr>
<tr>
<td>MAP</td>
<td>0.4700</td>
<td>0.4693</td>
<td>0.4735</td>
<td>0.4674</td>
<td>0.4719</td>
<td>0.4774</td>
<td>0.4777</td>
</tr>
</tbody>
</table>

### Data: Ohsumed

<table>
<thead>
<tr>
<th>Metric</th>
<th>RFR</th>
<th>RFPC</th>
<th>RFPCS</th>
<th>RFPR</th>
<th>RFPRS</th>
<th>RFL</th>
<th>RFLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.4351</td>
<td>0.4306</td>
<td>0.4499</td>
<td>0.4168</td>
<td>0.4468</td>
<td>0.4443</td>
<td>0.4490</td>
</tr>
<tr>
<td>MAP</td>
<td>0.4311</td>
<td>0.4213</td>
<td>0.4450</td>
<td>0.4265</td>
<td>0.4421</td>
<td>0.4332</td>
<td>0.4430</td>
</tr>
</tbody>
</table>

### Data: TD2004

<table>
<thead>
<tr>
<th>Metric</th>
<th>RFR</th>
<th>RFPC</th>
<th>RFPCS</th>
<th>RFPR</th>
<th>RFPRS</th>
<th>RFL</th>
<th>RFLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.3299</td>
<td>0.3513</td>
<td>0.3692</td>
<td>0.3509</td>
<td>0.3627</td>
<td>0.3558</td>
<td>0.3622</td>
</tr>
<tr>
<td>MAP</td>
<td>0.2275</td>
<td>0.2574</td>
<td>0.2675</td>
<td>0.2550</td>
<td>0.2629</td>
<td>0.2586</td>
<td>0.2627</td>
</tr>
</tbody>
</table>

### Data: HP2004

<table>
<thead>
<tr>
<th>Metric</th>
<th>RFR</th>
<th>RFPC</th>
<th>RFPCS</th>
<th>RFPR</th>
<th>RFPRS</th>
<th>RFL</th>
<th>RFLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.8048</td>
<td>0.8082</td>
<td>0.8203</td>
<td>0.7578</td>
<td>0.7957</td>
<td>0.7935</td>
<td>0.8035</td>
</tr>
<tr>
<td>MAP</td>
<td>0.7040</td>
<td>0.7174</td>
<td>0.7196</td>
<td>0.6730</td>
<td>0.7117</td>
<td>0.7031</td>
<td>0.7159</td>
</tr>
</tbody>
</table>

### Data: NP2004

<table>
<thead>
<tr>
<th>Metric</th>
<th>RFR</th>
<th>RFPC</th>
<th>RFPCS</th>
<th>RFPR</th>
<th>RFPRS</th>
<th>RFL</th>
<th>RFLS</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.7480</td>
<td>0.7860</td>
<td>0.7963</td>
<td>0.7624</td>
<td>0.7993</td>
<td>0.7675</td>
<td>0.7991</td>
</tr>
<tr>
<td>MAP</td>
<td>0.6309</td>
<td>0.6647</td>
<td>0.6602</td>
<td>0.6307</td>
<td>0.6625</td>
<td>0.6317</td>
<td>0.6673</td>
</tr>
</tbody>
</table>

• In general, RF-rand performs better than anticipated which is interesting because completely random splitting is used here. In Ohsumed dataset it performs even better than RF-point, while in MQ2007, MQ2008 and HP2004 its performance is close to that of RF-point.

• RF-list wins over RF-point in MQ2007, MQ2008 and Ohsumed, whereas in TD2004 it marginally wins, and in HP2004 and NP2004 it loses.

• Both RF-point and RF-list leverages from using the smaller sub-sample approach (with an exception of HP2004). However, RF-point benefits more than RF-list – the increase in performance from RF-point to RF-point-S% is in general more than that from RF-list to RF-list-S%.

The following points are observed from comparison between RF-based algorithms and other state-of-the-art algorithms:

• There is no consistent winner across different datasets. (This underlines the importance of investigating new algorithms despite the large number of algorithms in the literature.)
### Table 6.8: Lambdamart (LMart), Coordinate Ascent (CoorAsc), AdaRank (AdaR), Mart, RankBoost (RankB), RankSVM (rSVM) on small to moderate-sized datasets.

<table>
<thead>
<tr>
<th>Data: MQ2007</th>
<th>Metric</th>
<th>LMart</th>
<th>CoorAsc</th>
<th>AdaR</th>
<th>Mart</th>
<th>RankB</th>
<th>rSVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.4487</td>
<td>0.4408</td>
<td>0.4324</td>
<td>0.4422</td>
<td>0.4330</td>
<td>0.4436</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.4678</td>
<td>0.4614</td>
<td>0.4530</td>
<td>0.4608</td>
<td>0.4573</td>
<td>0.4659</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: MQ2008</th>
<th>Metric</th>
<th>LMart</th>
<th>CoorAsc</th>
<th>AdaR</th>
<th>Mart</th>
<th>RankB</th>
<th>rSVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.2302</td>
<td>0.2292</td>
<td>0.2220</td>
<td>0.2267</td>
<td>0.2263</td>
<td>0.2309</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.4751</td>
<td>0.4788</td>
<td>0.4700</td>
<td>0.4730</td>
<td>0.4767</td>
<td>0.4744</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: Ohsumed</th>
<th>Metric</th>
<th>LMart</th>
<th>CoorAsc</th>
<th>AdaR</th>
<th>Mart</th>
<th>RankB</th>
<th>rSVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.4367</td>
<td>0.4433</td>
<td>0.4476</td>
<td>0.4217</td>
<td>0.4362</td>
<td>0.4504</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.4173</td>
<td>0.4461</td>
<td>0.4458</td>
<td>0.4322</td>
<td>0.4452</td>
<td>0.4447</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: TD2004</th>
<th>Metric</th>
<th>LMart</th>
<th>CoorAsc</th>
<th>AdaR</th>
<th>Mart</th>
<th>RankB</th>
<th>rSVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.3292</td>
<td>0.3074</td>
<td>0.2812</td>
<td>0.2592</td>
<td>0.3076</td>
<td>0.2913</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.2378</td>
<td>0.2159</td>
<td>0.1899</td>
<td>0.1877</td>
<td>0.2235</td>
<td>0.2061</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: HP2004</th>
<th>Metric</th>
<th>LMart</th>
<th>CoorAsc</th>
<th>AdaR</th>
<th>Mart</th>
<th>RankB</th>
<th>rSVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.6398</td>
<td>0.7492</td>
<td>0.7682</td>
<td>0.6012</td>
<td>0.7160</td>
<td>0.7720</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.5177</td>
<td>0.6433</td>
<td>0.6626</td>
<td>0.4771</td>
<td>0.6257</td>
<td>0.6720</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Data: NP2004</th>
<th>Metric</th>
<th>LMart</th>
<th>CoorAsc</th>
<th>AdaR</th>
<th>Mart</th>
<th>RankB</th>
<th>rSVM</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.6221</td>
<td>0.7976</td>
<td>0.6974</td>
<td>0.5866</td>
<td>0.7004</td>
<td>0.7950</td>
<td></td>
</tr>
<tr>
<td>MAP</td>
<td>0.5006</td>
<td>0.6529</td>
<td>0.5830</td>
<td>0.4433</td>
<td>0.5591</td>
<td>0.6755</td>
<td></td>
</tr>
</tbody>
</table>

### Table 6.9: Using Tables 6.7 and 6.8, the ranks and aggregate ranks of all algorithms (in terms of NDCG@10) across the six datasets. The lower the rank, the better.

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MQ7</th>
<th>MQ8</th>
<th>Ohsu</th>
<th>TD</th>
<th>HP</th>
<th>NP</th>
<th>Aggregate Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF-rand</td>
<td>9</td>
<td>8</td>
<td>9</td>
<td>6</td>
<td>3</td>
<td>8</td>
<td>42</td>
</tr>
<tr>
<td>RF-point-cla</td>
<td>6</td>
<td>7</td>
<td>10</td>
<td>4</td>
<td>1</td>
<td>5</td>
<td>33</td>
</tr>
<tr>
<td>RF-point-cla-S%</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>RF-point-regr</td>
<td>7</td>
<td>7</td>
<td>12</td>
<td>5</td>
<td>8</td>
<td>7</td>
<td>46</td>
</tr>
<tr>
<td>RF-point-regr-S%</td>
<td>0</td>
<td>1</td>
<td>4</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>RF-list</td>
<td>3</td>
<td>0</td>
<td>5</td>
<td>4</td>
<td>5</td>
<td>6</td>
<td>22</td>
</tr>
<tr>
<td>RF-list-S%</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>3</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>Lambdamart</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>7</td>
<td>11</td>
<td>11</td>
<td>37</td>
</tr>
<tr>
<td>CoordAscent</td>
<td>1</td>
<td>1</td>
<td>6</td>
<td>9</td>
<td>9</td>
<td>2</td>
<td>28</td>
</tr>
<tr>
<td>AdaRank</td>
<td>8</td>
<td>7</td>
<td>3</td>
<td>11</td>
<td>7</td>
<td>10</td>
<td>47</td>
</tr>
<tr>
<td>Mart</td>
<td>1</td>
<td>2</td>
<td>11</td>
<td>12</td>
<td>12</td>
<td>12</td>
<td>50</td>
</tr>
<tr>
<td>RankBoost</td>
<td>7</td>
<td>3</td>
<td>8</td>
<td>8</td>
<td>10</td>
<td>11</td>
<td>47</td>
</tr>
<tr>
<td>RankSVM</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>6</td>
<td>4</td>
<td>4</td>
<td>20</td>
</tr>
</tbody>
</table>


- Although Lambdamart is considered by the LtR community to be one of the top algorithms, in our experiments it suffers from robustness across different datasets. Its performance in HP2004 and NP2004 are not near to the best. (However, on big datasets it is indeed one of the top performers as will be discussed shortly.)
Given that no consistent winning algorithm has been found across different datasets, it is difficult to draw an overall conclusion for all the algorithms on all six datasets. Hence in Table 6.9 we show the aggregate rank of each algorithm across all six datasets which is calculated by counting the number of times an algorithm is beaten by some others. Concretely, assuming $A$ is the set of algorithms and $D$ is set of datasets, and $m(A_i, D_j)$ is the score of a metric of choice of $i$th algorithm on $j$th dataset, 

$$\text{aggregated rank}(A_i) = \sum_{j=1}^{|D|} \sum_{k=1}^{|A|} 1(m(A_i, D_j) < m(A_k, D_j))$$

where $1(.)$ is an indicator function; the lower the aggregated rank of an algorithm, the better. This table demonstrates that RF-point-cla-S% is the most accurate (and also robust) algorithm across different datasets, closely followed by RF-list-S%. This investigation thus reveals that on the smaller datasets the state-of-the-art algorithms perform differently on different datasets, and that the RF-based algorithms are among the top candidates – both in terms of accuracy and robustness (across different datasets). Also, on the highly imbalanced datasets where many otherwise top performers find it difficult to perform well (e.g. LambdaMart), RF-based algorithms are the best.

The most unbalanced datasets, namely, HP2004 (home-page finding task) and NP2004 (named-page finding task) correspond to the search task of navigational queries, whereas TD2004, MQ2007 and MQ2008 datasets correspond to the search task of informational queries, and the Ohsumed dataset corresponds to a domain-specific search task. As discussed in Section 1.1 that the search tasks of domain-specific search and enterprise search usually differ from that of conventional general-purpose search engines. Our findings of this subsection reveal that RF-based algorithms are likely to perform very well in these domains where getting a large amount of labelled data is comparatively difficult.

Before concluding the discussion, we show the efficacy of LtR methods as compared to two individually highly informative features, namely BM25 and Language Model in Table 6.10. This table shows that LtR approaches are indeed effective as compared to a conventional ranker.

### 6.2.3 Big Datasets

This section analyzes results of two larger datasets from two perspectives: absolute performance and learning curves.

---

8 We perform a pairwise significant test on the comparatively larger (in terms of number of queries) MQ2007 and MQ2008 datasets. Since for the rest of the datasets the number of queries is small, the significance test results may not be reliable.

9 As explained earlier, since HP2004 and NP2004 datasets contain navigational queries, MAP may not be considered to be a very effective choice for evaluation for this type of information need [101, Sec. 8.4]. That is why in Table 6.9 we chose NDCG@10 for overall comparison.
Table 6.10: BM25 and Language Model (LM) performance on the six datasets along with the best and worst LtR performers among the algorithms we have investigated (cf. Tables 6.7 and 6.8). The percentage increase (or decrease) of performance of the best LtR method over BM25 is also reported in bracket.

<table>
<thead>
<tr>
<th>Data</th>
<th>Metric</th>
<th>BM25</th>
<th>LM</th>
<th>Best LtR</th>
<th>Worst LtR</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQ2007</td>
<td>NDCG@10</td>
<td>0.2956</td>
<td>0.2723</td>
<td>0.4487 (+152%)</td>
<td>0.4324</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.3384</td>
<td>0.3274</td>
<td>0.4678 (+138%)</td>
<td>0.4530</td>
</tr>
<tr>
<td>MQ2008</td>
<td>NDCG@10</td>
<td>0.1623</td>
<td>0.1307</td>
<td>0.2320 (+143%)</td>
<td>0.2228</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.3588</td>
<td>0.3137</td>
<td>0.4777 (+133%)</td>
<td>0.4700</td>
</tr>
<tr>
<td>Ohsumed</td>
<td>NDCG@10</td>
<td>0.3992</td>
<td>0.3902</td>
<td>0.4504 (+113%)</td>
<td>0.4217</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.4274</td>
<td>0.4260</td>
<td>0.4447 (+104%)</td>
<td>0.4322</td>
</tr>
<tr>
<td>TD2004</td>
<td>NDCG@10</td>
<td>0.1976</td>
<td>0.0854</td>
<td>0.3692 (+187%)</td>
<td>0.2592</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.1502</td>
<td>0.1076</td>
<td>0.2675 (+179%)</td>
<td>0.1877</td>
</tr>
<tr>
<td>HP2004</td>
<td>NDCG@10</td>
<td>0.6069</td>
<td>0.3570</td>
<td>0.8203 (+135%)</td>
<td>0.6012</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.4984</td>
<td>0.2629</td>
<td>0.7196 (+144%)</td>
<td>0.4771</td>
</tr>
<tr>
<td>NP2004</td>
<td>NDCG@10</td>
<td>0.5977</td>
<td>0.5654</td>
<td>0.7991 (+134%)</td>
<td>0.5866</td>
</tr>
<tr>
<td></td>
<td>MAP</td>
<td>0.5140</td>
<td>0.4741</td>
<td>0.6673 (+130%)</td>
<td>0.4433</td>
</tr>
</tbody>
</table>

Table 6.11: Performance of various RF-based algorithms on big datasets. Abbreviations are as follows: P for RF-point, H for RF-hybrid, cla for classification, reg for regression, rand for RF-rand.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.3978</td>
<td>0.4445</td>
</tr>
<tr>
<td>ERR</td>
<td>0.2978</td>
<td>0.3424</td>
</tr>
<tr>
<td>MAP</td>
<td>0.3271</td>
<td>0.3543</td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.7389</td>
<td>0.7538</td>
</tr>
<tr>
<td>ERR</td>
<td>0.4517</td>
<td>0.4594</td>
</tr>
<tr>
<td>MAP</td>
<td>0.6156</td>
<td>0.6278</td>
</tr>
</tbody>
</table>

Table 6.12: Performance of various algorithms on big datasets. The algorithms are: Mart, RankSVM (rSVM), AdaRank (AdaRa), CoorAsc, RankBoost (RBoost), LambdaMart (LmMart), and BM25 score.

<table>
<thead>
<tr>
<th>Data: MSLR-WEB10K</th>
<th>Metrics</th>
<th>Mart rSVM AdaRa CoorAsc RBoost LmMart BM25</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDCG@10</td>
<td>0.4416</td>
<td>0.3041 0.4169 0.3360 0.4869 0.2831</td>
</tr>
<tr>
<td>ERR</td>
<td>0.3491</td>
<td>0.2134 0.2746 0.3365 0.2460 0.3695 0.1910</td>
</tr>
<tr>
<td>MAP</td>
<td>0.3541</td>
<td>0.2657 0.2814 0.3197 0.2853 0.3640 0.2562</td>
</tr>
<tr>
<td>Data: Yahoo</td>
<td>Metrics</td>
<td>Mart rSVM AdaRa CoorAsc RBoost LmMart BM25</td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.7453</td>
<td>0.7177 0.7083 0.7177 0.7168 0.7520 0.6966</td>
</tr>
<tr>
<td>ERR</td>
<td>0.4575</td>
<td>0.4316 0.4441 0.4400 0.4315 0.4615 0.4285</td>
</tr>
<tr>
<td>MAP</td>
<td>0.6163</td>
<td>0.5983 0.5839 0.5906 0.5985 0.6208 0.5741</td>
</tr>
</tbody>
</table>

6.2.3.1 Absolute Performance

Tables 6.11 and 6.12 show the results of RF-based algorithms and other algorithms respectively on the MSLR-WEB10K and Yahoo datasets. As for the statistical significance among the differences in performance, Table 6.13 shows the pairwise significance test results conducted on several pairs of algorithms from Tables 6.11 and 6.12 — we avoid exhaustive pairwise comparison since most of the algorithms perform relatively poorly as compared to the top performers. Using the data from Tables 6.11 and 6.12, Figure 6.2 graphically shows performance of various algorithms.
Table 6.13: Significance test results for comparison of different algorithms from Tables 6.11 and 6.12. A pairwise test is performed, and the winning algorithm (among the two given in the heading of a column) is mentioned in a cell (along with the corresponding p-value). The examined algorithms are: RF-point-cla (cl), RF-point-reg (rg), RF-point-reg-K (rk), RF-hybrid-L6-cla (hc), RF-hybrid-L6-reg (hr), LambdaMart (lm), and RF-hybrid-L4-reg-K (hrk).

<table>
<thead>
<tr>
<th>p-value</th>
<th>cl, rg</th>
<th>cl, hc</th>
<th>rg, rk</th>
<th>rg, hr</th>
<th>rk, hrk</th>
<th>rk, lm</th>
<th>hrk, lm</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0.01</td>
<td>rg</td>
<td>hc</td>
<td>rk</td>
<td>rg</td>
<td>hrk</td>
<td>lm</td>
<td>lm</td>
</tr>
<tr>
<td>&lt; 0.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data: MSLR-WEB10K, Metric: NDCG@10

<table>
<thead>
<tr>
<th>p-value</th>
<th>cl, rg</th>
<th>cl, hc</th>
<th>rg, rk</th>
<th>rg, hr</th>
<th>lm, cl</th>
<th>lm, rg</th>
<th>lm, rk</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt; 0.01</td>
<td>rg</td>
<td>cl</td>
<td>rk</td>
<td>rg</td>
<td>lm</td>
<td>cl</td>
<td>rg</td>
</tr>
<tr>
<td>&lt; 0.05</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Data: Yahoo, Metric: NDCG@10

In both the datasets, RF-based algorithms and LambdaMart dominate the winning list, followed by Mart; the rest of the algorithms algorithms are comparatively poorer. Efficacy of RF-hybrid algorithm is observed in both classification (RF-point-classification) and regression (RF-point-regression) settings (on MSLR-WEB10K dataset), but the classification setting gets comparatively more benefit from it. With $K = 1/4M$, application of RF-hybrid becomes restrained by the computational resources available. On the Yahoo dataset, since the listwise objective was not found to be effective as compared...
to its pointwise counterpart, we do not report performance of RF-hybrid-K130 on it.\textsuperscript{10}
For the MSLR-WEB10K dataset, LambdaMart marginally wins over its nearest competitor which is RF-hybrid-regression-L4-K34. RF-point performs similar to or better than Mart. Interestingly, in spite of partitioning the data space randomly, RF-rand performs much better than RankSVM, RankBoost and AdaRank in both the datasets, and in addition, is better than CoorAsc on Yahoo data. However, its performance is significantly poorer than RF-point on both datasets, indicating that an objective-based splitting criterion (in this case the pointwise objectives) does help in an RF framework. Given AdaRank is a listwise algorithm, its mediocre performance is somewhat surprising. This is probably due to the fact that it combines the features linearly, and we have already discussed in the last segment of Section 5.6.4 that linear models may not capture the patterns of LtR data very well, especially if the dataset is large. CoorAsc is, in spite of being a listwise algorithm, outperformed by RF-point by large margin. In the Yahoo dataset, the RF-based pointwise and listwise algorithms dominate the winning list that also includes LambdaMart and Mart. Interestingly, here RF-point-K130 with regression setting wins over LambdaMart.

Both the RF-hybrid and LambdaMart perform relatively better on MSLR-WEB10K data than on Yahoo data – this is revealed when we compare RF-point with these two systems across the two datasets. This raises a question as to whether the characteristics of the Yahoo dataset cause the tree ensemble based listwise algorithms to perform comparatively less effectively (than the MSLR-WEB10K dataset). Below we give some discrepancies (cf. Table 2.2) between the two datasets:

- Number of features ($M$): MSLR-WEB10K: 136, Yahoo: 519.
- Sparsity in data (i.e., number of zeros which include missing values – the missing values are replaced by zero as per the standard practice [94]): MSLR-WEB10K: 37\%, Yahoo: 57\%.
- Number of documents per query: MSLR-WEB10K: 120, Yahoo: 23.
- Number of relevant documents per query: MSLR-WEB10K has much less relevant documents per query than that of Yahoo.

Investigation into the aspects outlined above could be an interesting direction for future research to find out if these aspects have any role to play in the performance difference between the pointwise and listwise algorithms.

\textsuperscript{10}We, however, did run a pilot experiment on comparison between RF-point and RF-hybrid with $K \in \{23, 50, 80, 130\}$ (the value 23 is used as $\sqrt{M}$), and did not observe improvement in performance of RF-hybrid over RF-point for any of the settings. We thus conclude that the $K$ is not likely to play a significant role in the relative performance of these two systems for the Yahoo dataset.
### Chapter 6: Detailed Comparison with State-of-the-Art Rank-Learners

#### 6.2.3.2 Learning Curve

Now we examine the learning curves for the two algorithms of our main interest, namely RF-point and RF-list, and the top performer of Table 6.12, namely LambdaMart. Figure 6.3 shows the plots for the MSLR-WEB10K and Yahoo datasets. Ignoring some minor fluctuations, the trend in performance across the different algorithms is clear. We see that with very small training sets, both the random forest based algorithms perform better than LambdaMart. In general, LambdaMart appears to be learning slightly faster with increasing amount of training data.

The learning curves further solidify the findings of the experiments with smaller datasets that RF-based algorithms perform comparatively better on smaller datasets.
6.2.4 Discussion

Some key findings of experiments of this section regarding the RF-based algorithms are summarized below:

On smaller datasets:

- RF based LtR algorithms with reduced sampling per tree helps to improve performance in both the pointwise and listwise algorithms.

- The random splitting-based algorithm performs slightly poorer or as good as the pointwise splitting-based algorithm.

- On the datasets with graded relevance, the listwise algorithm performs better than its pointwise counterpart.

On larger datasets:

- Random splitting is not as effective as it was found in smaller datasets.

- Performance of a listwise splitting criterion is likely to be dependent on the properties of the dataset in that it may or may not improve performance over its pointwise counterpart.

- Hybrid counterparts of the both classification and regression algorithms improves performance (when applicable, for example, in MSLR-WEB10K dataset).

In general, the following findings emerge from the comparison between RF-based rank-learners and other state-of-the-art systems:

- Performances of various algorithms are heavily dependent on the properties of the datasets. Key such properties include: data size and relevance label distribution.

- For the Yahoo dataset, adapting the objective function to a rank-based metric appears to help comparatively less than it did in the case of MSLR-WEB10K (considering the relative performance degradation of RF-hybrid and LambdaMart on the Yahoo dataset as compared to the MSLR-WEB10K).

- In terms of data-dependency, for big datasets LambdaMart is a relatively robust algorithm.
• RF-point/list consistently win over to RankSVM (pairwise), RankBoost (pairwise), Coordinate Ascent (listwise), AdaRank (listwise), and marginally better than Mart (pointwise) on big datasets. This shows that RF-based LtR algorithms are competitive with the state-of-the-art methods, and hence these algorithms need further attention from the research community.

• BM25 on its own performs poorer than all LtR systems in all cases, demonstrating the efficacy of the rank learners.

• The random splitting-based algorithm performed better than some state-of-the-art algorithms, namely AdaRank [166], RankSVM [83], RankBoost [49], and sometimes Coordinate Ascent [105]. Given the non-adaptiveness of its objective function, this finding is interesting. This indicates that the nearest-neighbor based methods may be effective in LtR. Moreover, a completely random tree based ensemble has huge computational advantage over many other LtR algorithms.

• Experiments on the smaller datasets (of diverse properties) and on the learning curves show that with smaller training sets discover that the RF-based LtR algorithms are robust and perform quite well, which makes them strong candidates for the domains where getting large amount of labelled training data is difficult (e.g. domain-specific search and enterprise search [147]).

• Irrespective of the approaches (i.e., pointwise etc.), non-linear models tend to perform better for large scale LtR. This raises a question: should more focus be devoted on understanding the model complexity (e.g., bias-variance tradeoff) rather than designing sophisticated LtR algorithms? This is an interesting research direction that we address in the next chapter.

Before concluding this section, we highlight the fact that the random forest based LtR algorithms are inherently completely parallelizable, and require very little tuning of their parameters. We note that different algorithms such as boosting have recently been shown to be eligible for some form of parallelization [151]. However, these techniques are not as obvious as the parallelism of random forests is, and moreover, these techniques are still subject to scrutiny in terms of performance\textsuperscript{11}. Nonetheless, objectively comparing the fastest LtR algorithms is still an interesting direction for future work.

\textsuperscript{11}The inventors of the said technique [151] admit that there is a concern of degradation of accuracy, slightly though.
6.3 Conclusion

In this chapter, firstly, we have investigated a few dimensions of random forest based algorithms. Secondly, we have extensively compared the algorithms investigated in the previous chapters with other relevant state-of-the-art algorithms.
Chapter 7

Understanding Bias and Variance of Rank-Learners

The bias-variance profiles of LtR algorithms have not been, to the best of our knowledge, explored in the literature. In this chapter we aim to better understand the bias-variance profiles of RF-based rank-learning algorithms, although our developed frameworks are applicable to other LtR algorithms as well. Firstly, we formalise the bias and variance from a pointwise perspective. We then develop a framework to analyse the variability and systematic error of a rank-learner in terms of its ranking error, i.e., we analyse the bias-variance from a listwise perspective. After that we study the effect of varying an important parameter, namely the sub-sample size used to learn each tree, on bias and variance. Our hypothesis is that as the sub-sample size (per tree) increases, classical bias-variance tradeoff should be observed. On the two large LtR datasets, experimental results show that our hypothesis holds true. Finally we explain the relative performance of two of the top performing algorithms using the bias and variance.

7.1 Motivation

In our investigation on sub-sample size per tree (denoted by $p$) in Chapter 4 we found that as we vary $p$, the performance of the two big datasets looked like Figure 7.1 (copied from Chapter 4). We also speculated a possible cause for this behavior which is the comparatively low variance of individual trees (due to inherent large size of the data). However, the investigation into the behavior from the classical bias-variance perspective was out of scope of that chapter. In this chapter, we aim at better understanding the bias-variance profiles of RF-based rank-learners in the context of the sub-sample size (per tree) parameter, although the developed formulations are equally applicable
to a generic rank-learner. Briefly, the bias of a learner is the systematic error of a model across different samples of a fixed size, and the variance is the error due to the fluctuation in predictions of the model across different samples of a fixed size. From a practical perspective, our intended investigation would help us in the following two ways: (1) explaining the cause of performance trade-offs between different rank-learners, and (2) knowing whether bias or variance contributes more to the error rate of RF-based rank-learners would allow us to pay special attention to that aspect.

7.2 Contributions

The major contributions of this chapter are as follows:

- From the perspective of bias and variance, we provide an intuitive explanation of evolution of a random forest from a single tree.

- We formalise bias and variance of rank-learners from a pointwise perspective.

- From a listwise perspective, we design a framework to empirically measure both the variability of ranking error across different samples and the systematic ranking error of a generic rank-learner. We also thoroughly discuss some alternative frameworks.

- Experimental results using an RF-based rank-learner demonstrate that modifying the parameter for the sub-sample size per tree allows one to control the trade-off between bias and variance.

- We explain the relative performance of an RF-based rank-learner and the LambdaMart algorithm using the estimations of listwise bias and variance.
7.3 Background

The bias of a learner is the systematic error of the learner less the Bayes’ (aka irreducible) error rate. For ease of exposition, in the following we shall assume that the Bayes’ error rate is zero.

For parametric methods, where the number of model parameters is fixed, bias can be understood as the error that is achieved by a classifier trained on infinite quantities of data.\footnote{Providing the learning algorithm is consistent this will correspond to the point of minimum loss in the parameter space.} For non-parametric models such as decision trees, where the number of parameters grows with the amount of training data available, the bias of the learner is not constant but reduces with the quantity of training data. In the limit of large data the bias reduces to zero.

Thus the bias of a non-parametric learner depends on the amount of data available and can be defined as the error rate of the expected prediction over all models learnt from all possible training data sets of a given size. In other words, for a regression tree learner the (squared) bias is simply the error rate of an ensemble of all possible regression trees learnt from different training sets of that size.

The variance of a learner is the expected value (over all such trees) of the additional (over and above the bias) prediction error of a particular tree learnt from a given dataset. The additional error is due to the fact that on the particular training set, the learner learns a suboptimal hypothesis due to overfitting the peculiarities of this particular sample.

From the theory of learning algorithms [57], we know that the following equation holds for squared loss function (i.e., regression problem):

\[
\text{Generalization Error} = \text{bias}^2 + \text{variance} + \text{Irreducible Error}.
\] (7.1)

In the rest of the chapter, after numerous works such as [41, 86, 87], we assume the irreducible error (cf. Equation 7.1) to be zero.\footnote{In the context of LtR, this assumption implies that we are considering the relevance judgements of documents to be deterministic.}
Mathematically, Equation 7.1 can be written as:

\[
Err_{\text{mse}}(x_k) = \mathbb{E}[(f(x_k) - l_k)^2] \tag{7.2}
\]

\[
= \mathbb{E}[(f(x_k))^2] + \mathbb{E}[(l_k)^2] - 2\mathbb{E}f(x_k)l_k
\]

\[
= \text{Var}(f(x_k)) + (\mathbb{E}[f(x_k)])^2 + l_k^2 - 2l_k\mathbb{E}f(x_k)
\]

\[
= [\mathbb{E}f(x_k) - l_k]^2 + \mathbb{E}[(f(x_k) - \mathbb{E}f(x_k))^2]
\]

\[
= b^2(x_k) + \sigma^2(x_k)
\]

where \(b^2(x_k)\) denotes the squared bias at the point \(x_k\) and \(\sigma^2(x_k)\) denotes the variance at point \(x_k\).

Some useful reading materials to understand bias and variance of a generic loss function have been written by Domingos [41], Geurts [61] and James [81].

### 7.4 Understanding Random Forests from a Bias-Variance Perspective

Let \(T, B,\) and \(R\) denote a single regression tree, a bagged ensemble [14], and a random forest [15] respectively; and \(b^2\) and \(\sigma^2\) denote the squared bias and variance respectively.\(^3\)

We use \(T_s\) to denote that a tree is learnt using \(s\) percentage of training set; thus: \(T_{63}\) and \(T_{100}\) denote that the tree learnt from a bootstrap (without replacement) sample and the full sample respectively. \(B_s\) denotes an ensemble of bagged trees where each tree is learnt using \(s\) percentage of training set, and similar interpretation is used for \(R_s\). If the meaning is obvious, we omit some subscripts; for example, sometimes we use bare \(B\) and \(R\) to denote the default case, i.e., \(B_{63}\) and \(R_{63}\) respectively.

As mentioned earlier in the thesis, a number of works such as [53, 54, 138, 154] demonstrated that bootstrapping without replacement works as well as bootstrapping with replacement, so maintaining our setting of the previous chapters, we use the former setting.

Ignoring the irreducible term of Equation 7.1 [41, 86, 87], we can write:

\[
Err_T = b^2_T + \sigma^2_T \tag{7.3}
\]

\[
Err_B = b^2_B + \sigma^2_B
\]

\[
Err_R = b^2_R + \sigma^2_R
\]

\(^3\)While values of squared bias and variance are understood to be calculated at a point denoted by \(b^2(x_k)\) and \(\sigma^2(x_k)\) respectively, for the sake of simplicity the parentheses with the data location will be dropped in many cases.
There are several useful reading materials for understanding bias-variance decomposition of randomized tree ensembles such as [15] and [69, Ch. 15]. However, their explanations of the bias and variance of RF are not particularly thorough. Since in our investigations we shall concentrate on RF-based rank-learners, it is imperative to have an in-depth understanding of RF from a bias-variance perspective. Hence below we explain the evolution of an RF from a single tree from a bias-variance perspective.

7.4.1 Bagging for Variance Reduction

We begin with the analysis of a bagged ensemble (also known as bagger) because it can be considered as a predecessor to a random forest. It is known from the literature (see, for example, [69, Ch. 15]) that the bias of a bagged ensemble is equal to that of a single tree from the ensemble. Therefore:

\[ b^2_B = b^2_{T_{63}} \]  

(7.4)

and variance:

\[ \sigma^2_B = \rho \sigma^2_{T_{63}} + \frac{(1 - \rho) \sigma^2_{T_{100}}}{E} \]  

(7.5)

where \( \rho \) is the correlation between two trees (at a datapoint) and \( E \) is size of the ensemble respectively. Our derivation of this formula is given in Appendix C.1.

If \( E \) is large enough, the 2nd term of Equation 7.5 tends to vanish irrespective of the values of \( \rho \) and \( \sigma^2_{T_{63}} \). For an ensemble of identical trees, \( \rho = 1 \). The idea of bagging is to decrease \( \rho \) by using a different training set to learn each tree so as to reduce \( \rho \). However, in practice only one training set is available. To overcome this problem, each tree is learnt from a bootstrap sample thereby making each tree different from one another. However, bootstrapping gives rise to two additional concerns: (1) it increases \( \sigma^2_{T_{63}} \) (i.e., \( \sigma^2_{T_{63}} > \sigma^2_{T_{100}} \)), because individual trees are now learnt using less information than before, and hence the overfitting tendency of a tree learnt from a bootstrapped sample is more than a tree learnt from the original sample, and (2) it increases bias of individual trees (i.e., \( b^2_{T_{63}} (= b^2_B) > b^2_{T_{100}} \)), because, again, now the sample space is smaller which shrinks

---

4This can be explained as follows. The (squared) bias at a data point \( x_k \) is \((l_k - E[f(x_k)])^2\). Given the model complexity is fixed (a tree with fixed parameters is being used in all cases, i.e., trees are identically distributed [69, Ch. 15]), increasing the number of trees cannot improve the bias component of error. This is because increasing the number of trees to compute their average simply brings the average closer to the true average, it does not systematically increase/decrease the average.

5As explained in Section 4.7.1 that it is important to distinguish between the correlation between two trees of a random forest and the correlation \( \rho(x_k) \) in Equation 7.5.
the hypothesis space. But empirically it has been observed the positive effect of reducing $\rho$ is much greater than the negative effect of increasing $\sigma_{T_{63}}^2$ (cf. Equation 7.5), thereby causing:

$$\sigma_B^2 << \sigma_{T_{100}}^2$$

Thus we can summarize the above discussion as follows:

$$b_B^2 > b_{T_{100}}^2$$

$$\sigma_B^2 << \sigma_{T_{100}}^2$$

Empirically it has been found that:

$$b_B^2 + \sigma_B^2 < b_{T_{100}}^2 + \sigma_{T_{100}}^2$$

$$\Rightarrow Err_B < Err_{T_{100}}$$ (using Equation 7.3).

Therefore, the bottom line regarding bagging is the following: the benefit of aggregating predictions of many trees learnt from comparatively smaller training sets (i.e., bootstrapped training set) outweighs the benefit of using a single prediction from a tree learnt from a comparatively larger training set.\(^6\)

### 7.4.2 Adding Randomness to Make a Random Forest

The idea of a random forest is to reduce $\rho$ even more, without substantially increasing per-tree-variance, $\sigma_{T_{63}}^2$ and bias, $b_{T_{63}}^2$ ($= b_{R_{63}}^2$). The way a random forest achieves this is by modifying the tree building procedure as follows. At each node a small number (denoted by $K$) of features are randomly chosen, then the optimal split is determined over only those $K$ chosen features.\(^7\) (We note that there are other possible ways to achieve the same correlation reduction effect, for example, our approach of using smaller sub-sample per tree as discussed in Chapter 4.) The formulation of variance of a random forest is the same as that of bagging (cf. Equation 7.5) because a random forest simply adds an extra source of randomness to a tree (by selecting a random subset of features

\(^6\)Another way to explain the benefit of diversity in an ensemble is that by increasing diversity across trees we enable the ensemble to capture increasing amount of non-linear (complex) relationship between labels (by making the ensemble decision boundary smoother when averaging many different axis-parallel decision boundaries), given ensemble size is sufficiently large.

\(^7\)An additional benefit of this scheme is (as indicated by the advocates of a random forest), features which did not get “chance to speak” due to the influence of some other stronger features now get a chance to be selected as a split-point, thereby improving over the greedy nature of a decision tree.
at a node). Again, empirically it has been found that the (positive) effect of reduction of $\rho$ on $\sigma^2_R$ is greater than the (negative) effect of rise of $\sigma_T$ (cf. Equation 7.5), thereby causing:

$$\sigma^2_R < \sigma^2_B << \sigma^2_{T100}$$

As for the bias of a random forest, it is, like bagging, equal to that of a single tree from the ensemble. As long as $K$ is not too small and the ensemble size is sufficiently large, the additional randomness does not harm the systematic error of the ensemble (i.e., expected error across multiple ensembles learnt from different samples), thereby causing $b_R^2$ to be roughly of similar level to $b_B^2$. Thus we can write:

$$b_R^2 \geq b_B > b_{T100}^2$$

Now using the same reasoning of the discussion of bagging, we can write:

$$Err_R < Err_B < Err_{T100}$$

We highlight the following point again: reducing correlation helps only if the per-tree variance, $\sigma^2_{T_{k3}}$ and per-tree bias, $b_{T_{k3}}$ are not greatly increased; that is why using a very small number of random features at each node of a random forest does not yield good performance [69, Ch. 15]. For example, in Chapter 5 we showed that an ensemble of completely randomized trees that has very low correlation does not perform well on big data; and our experimental analyses later in the chapter will reveal that if we reduce the correlation drastically, the individual tree variance and bias heavily increase.

Thus the bottom line about a random forest is the following: as we reduce $\rho$ (by selecting a small subset of random features and by bootstrapping), we increase both the variance $\sigma^2_{T_{k3}}$ and squared bias $b^2_{T_{k3}}$ of the random forest as compared to $\sigma^2_{T_{100}}$ and $b^2_{T_{100}}$. However, $\sigma^2_R$ (cf. Equation 7.5) continues to decrease due to the reduction of $\rho$, and as a result of this, if we continue to increase randomness (by decreasing the number of randomly chosen features used to determine each split), then up to some point the ensemble error $Err_R$ continues to decrease. After this point, $Err_R$ starts to increase again; and the question is, is the higher variance $\sigma^2_{T_{k3}}$ or higher squared bias $b^2_{T_{k3}}$ or both responsible for this rise in $Err_R$? This study is scant in the literature of classification and regression; in fact we found only one such work by Hastie et al. [69, Ch. 15] who conducted a pilot experiment on a synthetic regression dataset which shows that if $K$ is reduced greatly,

---

8We note that due to the greedy nature in which each tree is built, it is likely but not necessarily the case that trees in a random forest have higher bias than do trees in bagged ensemble.

9Also, $\sigma^2_{T_{k3}}$ is increased as compared to bagger’s variance.
both the single tree variance and bias increase which in turn increase the error rate. In the ranking domain, we have not found any such investigation. Therefore, this chapter is devoted to the bias-variance analysis of rank-learners.

7.5 Related Work

On correlation and strength of a random forest. Bernard et al. [10] plot strength (i.e., predictive accuracy of individual trees) and correlation between the trees along with error rate of random forests for classification. They also examine the effect of ensemble size. Their findings include: the relationship between strength, correlation and error rate formulated in Theorem 2.3 of [15] largely holds in practice.

On parameter tuning of a random forest. Segal [141] tunes the minimum node size parameter, $n_{\text{min}}$, of a tree for the regression setting. Lin and Jeon [93] also work with $n_{\text{min}}$ and, in addition, with the number of candidate features at each node, $K$ (usually set to $\log(\#\text{features}) + 1$). Their findings indicate that slight improvement in performance may be found by tuning these parameters, although default values work well in practice. Hastie et al. [69, Ch. 15] comment that while for regression tuning $n_{\text{min}}$ helps slightly, for classification it rarely makes any difference.

On theoretical analyses of random forests. The theory behind random forest was not given much attention until recently. Some notables works are: Wager [154], Scornet et al. [137] and the references therein. Most of these works analyse simplified versions of RF because the standard random forest where the best split is found amongst a random subset of the variables is difficult to theoretically analyse.

On theoretical underpinnings of LtR problem. The theoretical background on which the LtR problem is based has been analysed in a few papers [88, 89, 160, 161]. However, none of these works are bias-variance oriented, rather they focus on generalization performance and consistency of listwise algorithms.

The only work we found that even hints at the trade-off between bias and variance for online LtR is conducted by Hofmann et al. [75]. The authors, however, preferred the notions “exploration” and “exploitation” instead of bias and variance respectively. As such, our work is, to the best of our knowledge, the first endeavor in the bias-variance realm of rank-learning algorithms.

Thus we see that although there have been some work on analysing the correlation, strength and generalization error of a random forest in the context of classification and regression, we did not find any substantial work on the bias-variance interaction.
Chapter 7: Understanding Bias and Variance of Rank-Learners

of random forest based rank-learners. This chapter attempts to fill this gap in the literature: it helps us better understand the theory of rank-learners in terms of their bias-variance characteristics, and applies the theory to practice by investigating into an important parameter of the RF-based LtR algorithms, namely the sub-sample size per tree.

7.6 Methodology

We first define the bias and variance. We then select a dimension, namely the percentage of training data used to learn each tree (denoted by $p$) of an RF in order to examine the trend of bias and variance.

In the rest of the chapter we aim to achieve three goals in particular: (1) to design a framework to analyse bias-variance of a generic rank-learner, (2) to examine if classical bias-variance trade-off is observed while varying the parameter sub-sample size per tree of an RF, and (3) to explain performance trade-off between two top performing algorithms on large LtR datasets, namely LambdaMart and an RF-based algorithm.

7.6.1 Two Types of Analyses

In this chapter we differentiate between pointwise/listwise learning algorithms/models and pointwise/listwise analysis of bias-variance profiles. Recall that in a pointwise algorithm, a query-document pair (i.e., a feature vector) is treated independently from one another, whereas in a listwise algorithm, a query along with its associated documents is considered as a single instance. Both the categories of the algorithms predict a real score for each query-document pair, but they differ in the way they optimise their loss functions. This common characteristic motivates us to analyse two frameworks for the bias-variance analysis of a generic rank-learner as explained below.

- **Pointwise analysis.** This setting analyses the variance and systematic error of the predicted scores (of feature vectors) themselves, i.e., in terms of surrogate loss. It assumes that the prediction of a model is a series of scores (without any notion of queries) whose target labels are the relevance judgements of individual documents. This analysis would help us understand how the scores (of the documents) themselves are affected in terms of the bias-variance of a model.

- **Listwise analysis.** This setting analyses the variance and systematic error of the ranked lists induced by the predicted scores, i.e., in terms of ranking loss. It assumes that the prediction of a model is a ranked list of documents associated
with a query. This analysis would help us understand how the IR evaluation metrics are affected in terms of the bias-variance of the model.\footnote{We note here that there is a discussion in the literature regarding casting the classification problem (with class posterior probabilities) onto regression and then using the formulations and bias-variance decomposition of regression problem such as by Manning et al. \cite[Sec. 14.6]{Manning} and Hastie et al. \cite[Sec. 15.4]{Hastie} – we quote the comment of the latter work: \textit{“...Furthermore, even in the case of a classification problem, we can consider the random-forest average as an estimate of the class posterior probabilities, for which bias and variance are appropriate descriptors.”} However, Friedman \cite{Friedman} explains that a better calibration of the class posterior probabilities of the correct class does not necessarily translate into a higher accuracy of a classifier (in terms of 0/1 loss). A similar concern is true for ranking loss functions, hence the investigation of listwise rather than pointwise bias-variance analysis is important.}

Note that the listwise analysis is also applicable to the pointwise algorithms since applying this would explain the ranking loss of the latter group of algorithms. However, it would not make sense to apply the pointwise analysis to listwise algorithms as these algorithms do not target to predict the relevance labels themselves, but rather attempt to predict entire ranked lists.\footnote{Hence the scores predicted by these algorithms (e.g., LambdaMart) can even be negative.}

### 7.6.2 Model

We mainly use an RF based pointwise rank-learner with regression setting (denoted by simply RF-point). However, our formulations of pointwise analysis are applicable to any pointwise rank-learner, and the formulations of listwise analysis are applicable to any rank-learner. We apply both the pointwise and listwise analyses to RF-point.

### 7.7 Pointwise Analysis

We have already shown the relationship between the error rate, bias and variance for regression problems in Section 7.3.

For the classification setting (i.e., 0/1 loss), a number of different (and often somewhat conflicting) definitions are proposed in the existing literature, see \cite{Manning, Hastie} for detailed comparison. Also, in Chapter 6 we found that for big datasets the regression setting oftentimes works better than the classification. For these reasons, in our analyses we use the mean-squared error loss function.

A question may arise as to what is the relationship between bias-variance of regressors and that of rank-learners. Since the regression error provides an upper bound to ranking error \cite{Lee}, the analysis of bias-variance of a regression-based rank-learner would, to some extent, be helpful to understand its ranking error.
Below we develop the framework to be used for empirical estimation of variance and bias from a pointwise perspective. To generate multiple training sets from a single one, we employ two different methods that enjoy widespread practice within the research community (as detailed below).

### 7.7.1 Estimating Bias-Variance Using Multiple Bootstrap Samples

We employ a well-known practice among researchers which is to sample a bootstrapped copy of the training set for learning each estimator (in our context, each estimator is an entire ensemble). The advocates of this approach include Domingos [41], Geurts [61], James [81] and Sexton and Laake [142]. We note, however, that there is a trade-off in choosing the amount of data per ensemble. On the one hand, we want to learn many ensembles using as different training samples as possible. On the other hand, we have only limited amount of training data, so if we instead use many small (and disjoint) subsamples, we can build more ensembles without sharing any information across the ensembles, but this will increase the variance of the model. Considering this trade-off, we stick to the standard practice of the thesis, i.e., sampling 63% of the data which is tantamount to bootstrapping without replacement. We call this method the method of bootstrapping.

In this method the ensemble variance is likely to be underestimated because the training sets of the ensembles share much information as each ensemble is learnt on 63% percent of the original training data. This underestimation is, however, not a major stumbling block for us since our main goal is to examine if we can capture the trend of variance and bias, and not necessarily their absolute values. That is, our goal is to examine whether bias and variance are changed according to our hypothesis as we vary a parameter of the model. Similar reasoning is expressed by Domingos [41] and Geurts [61], among others.

Let $B$ be the number of ensembles/models from which empirical variance and empirical bias are to be computed. Each ensemble/model is learnt using approximately 63% data (i.e., using $D_i; 1 \leq i \leq B$) of the training set. The different quantities are then estimated using the test instances.

**Bias:** Squared bias is the average over the data instances of the squared bias of the ensemble predictions. As such, it is defined as:

$$
\hat{b}_{\text{bootstrap}}^2 = \frac{1}{N} \sum_{k=1}^{N} \hat{b}_{\text{bootstrap}}^2(x_k), \quad (7.6)
$$
where $\hat{b}^2_{\text{bootstrap}}(x_k)$ is the estimate of squared bias at $x_k$. Thus:

$$\hat{b}^2_{\text{bootstrap}}(x_k) = (l_k - \bar{f}(x_k))^2, \quad (7.7)$$

where $\bar{f}(x_k) = \frac{1}{B} \sum_{i=1}^{B} f_{D_i}(x_k)$, and $f_{D_i}(x_k)$ is the prediction of $i$th ensemble/model learnt from $D_i$ on $k$th feature vector. In the case of RF, $f_{D_i}(x_k) = \bar{s}_i(x_k) = \frac{1}{E} \sum_{j=1}^{E} s_{j,i}(x_k);\ s_{j,i}(x_k)$ is the prediction of $j$th tree in $i$th ensemble on $x_k$.

**Variance:** The model variance is calculated by taking the average over the data instances of the variances of the model predictions. As such, it is given by:

$$\hat{\sigma}^2_{\text{bootstrap}} = \frac{1}{N} \sum_{k=1}^{N} \hat{\sigma}^2_{\text{bootstrap}}(x_k) \quad (7.8)$$

$$\hat{\sigma}^2_{\text{bootstrap}}(x_k) = \frac{1}{B-1} \sum_{i=1}^{B} (f_{D_i}(x_k) - \bar{f}(x_k))^2. \quad (7.9)$$

where $\hat{\sigma}^2_{\text{bootstrap}}(x_k)$ is the estimate of variance at $x_k$. Thus:

**7.7.2 Estimating Bias-Variance Using Two-Fold Cross-Validation**

Here we employ another standard practice (e.g. by Kohavi and Wolpert [86]) which is to divide the dataset into two disjoint sets, and learn two models using them. We then calculate the variance and bias using these two ensembles which gives us an estimate about how much the prediction on a single datapoint varies if it is predicted using an ensemble learnt from a completely different training set. We repeat the whole process $J$ times, and finally average the estimates of these “two-fold” experiments. We call this method the *method of repeated twofold CV*. This variance, however, is likely to be overestimated because of using only half of of the training data to train a learner. Thus we see that both the methods have limitations.

**Bias:** As explained above that we generate two disjoint training sets $D_1$ and $D_2$ from the original set $D$, and we repeat this process $J$ times that results in samples ${\{D_{11}, D_{12}, D_{21}, D_{22}, \ldots, D_{1J}, D_{2J}\}}$ where $D_{1j} \cap D_{2j} = \emptyset$. With $1 \leq j \leq J$, the $j$th estimate of squared bias is defined as:

$$\hat{b}^2_{\text{twofold}} = \frac{1}{N} \sum_{k=1}^{N} \hat{b}^2_{\text{twofold}}(x_k); \ \hat{b}^2_{\text{twofold}}(x_k) = (l_k - \bar{f}_j(x_k))^2, \quad (7.11)$$

where $\bar{f}_j(x_k) = \frac{1}{2} \sum_{i=1}^{2} f_{D_{ij}}(x_k)$. 


Finally, the squared bias estimate for the model is:

\[ \hat{b}_{\text{twofold}}^2 = \frac{1}{J} \sum_{j=1}^{J} \hat{b}_{\text{twofold},j}^2. \]  

(7.12)

**Variance:** With \( 1 \leq j \leq J \), the \( j \)th estimate of variance is given by:

\[ \hat{\sigma}_{\text{twofold},j}^2 = \frac{1}{N} \sum_{k=1}^{N} \hat{\sigma}_{\text{twofold},j}(x_k): \quad \hat{\sigma}_{\text{twofold},j}(x_k) = \frac{1}{2-1} \sum_{i=1}^{2} (f_{D,j}(x_k) - \bar{f}_j(x_k))^2, \]  

(7.13)

where \( f_{D,j}(x_k) \) is, as before, the prediction of \( j \)th ensemble/model.

Finally, the variance estimate for the model is:

\[ \hat{\sigma}_{\text{twofold}}^2 = \frac{1}{J} \sum_{j=1}^{J} \hat{\sigma}_{\text{twofold},j}^2. \]  

(7.14)

### 7.8 Listwise Analysis

While the pointwise analysis of bias-variance gives us an idea of bias and variance in terms of ranking error, it cannot precisely tell us about the systematic ranking error and fluctuation in predictions of a model in terms of ranking error. For example, high variance in the predicted scores does not necessarily translate into high variance in terms of NDCG. Moreover, this analysis is not suitable for pairwise and listwise algorithms as these algorithms do not estimate the relevance labels, but rather the entire ranked lists for a query. For these reasons, in this section we develop a framework for analysing the bias and variance of a generic rank-learning algorithm in terms of ranking error.

**Notations.** Recall that \( f(\bar{x}_{q,i}) \) is the predicted score for the feature vector \( \bar{x}_{q,i} \) corresponding to the query \( q \) and \( i \)th document of \( q \). For the sake of better readability, here we slightly abuse our standard notations. We introduce a simpler notation for the sequence of feature vectors for a particular query \( q \) as follows: let \( x_q = \{\bar{x}_{q,i}\}_{i=1}^{n_q} \), and \( \tilde{l}_q = \{l_{q,i}\}_{i=1}^{n_q} \). Also, let \( \tilde{f}(\{\bar{x}_{q,i}\}_{i=1}^{n_q}) = \tilde{f}(x_q) = (f(\bar{x}_{q,1}), f(\bar{x}_{q,2}), \ldots, f(\bar{x}_{q,n_q})) \), i.e., \( \tilde{f}(x_q) \) (or \( \tilde{f}(q) \)) is a series of values, and \( \tilde{f}(x_q) \) is the average of \( \tilde{f}(x_q) \) across models learnt from different samples. Finally, the NDCG of a query \( q \) induced by the ranking generated from the scores predicted by a model \( \tilde{f} \) is denoted by \( NDCG(\tilde{l}_q; \tilde{f}) \). Sometimes we, depending on the context, alternatively use \( NDCG(q; \tilde{f}) \) to represent \( NDCG(\tilde{l}_q; \tilde{f}) \).
7.8.1 Preliminaries

Let $y$ be the true label of an instance $x$, $\hat{y}(x)$ is the prediction at $x$. The loss at $x$ is expressed $\text{loss}(\hat{y}(x), y)$. To begin with, the systematic prediction at $x$ is defined as the prediction that minimizes the expected loss between it and the predictions using other samples. Concretely, $\hat{y}^*(x) = \arg\min_{\hat{y}'}(\text{loss}(\hat{y}'(x), \hat{y}(x)))$. Assuming that the irreducible error (aka Bayes’ error rate) is absent, this loss has two components: (1) the systematic error which is denoted by $\text{loss}(\hat{y}^*(x), y)$ (i.e., bias of the learner), and (2) the error due to sampling variation which is denoted by $\mathbb{E}_D[\text{loss}(\hat{y}(x), \hat{y}^*(x))]$ (i.e., the variance of the learner). Figure 7.2 depicts the scenario. In order to compute the bias and variance for the ranking problem, we need to instantiate these two losses in the context of ranking problem, and the forms of these loss functions depend on the assumption on the form of $x$ which is described below.

Two Representations of Queries. In a listwise setting a query is considered to be an instance (as opposed to individual query-document pairs). There are two ways to represent a query:

1. Permutations of documents.
2. Labels of documents.

These two representations lead to different instantiations of various quantities as explained next.

7.8.2 Bias-Variance Using Permutations of Documents

Suppose $\pi(l_q)$ is a true ranking of the feature vectors corresponding to the documents associated with query $q$, and $\pi(f(x_q))$ is the ranking of those documents induced by

---

Footnotes:

12 Here $x$ is a generic instance, not necessarily the feature vector corresponding to a query-document pair.
13 The work by Domingos [41] is useful to further understand this concept.
the scores predicted by the ranking model \( f(\cdot) \). Within this setting, the true label is denoted by \( \pi(l_q) \), the predicted label is denoted by \( \pi(f_D(x_q)) \), and the systematic prediction for an instance is denoted by \( \mathbb{E}_D[\pi(f_D(x_q))] \) where \( \pi(f_D(x_q)) \) is the predicted ranked list of query \( q \) by model \( f \) learnt from dataset \( D \).

As such, the ranking loss at a query \( q \) is written as:

\[
\text{Err}_{\text{ranking}}(q) = \text{loss}(\pi(f(x_q)), \pi(l_q)).
\] (7.15)

In order to define the bias and the variance for the ranking loss, we now need to (1) define the systematic prediction, i.e., a method for aggregating multiple ranked lists, and (2) measure the deviation between two ranked lists of documents. To achieve the former of these two tasks, the rank-aggregation methods such as Borda’s method can be used [45, 139], whereas to accomplish the latter, the well-known rank-distance methods such as Kendall’s Tau or Spearman’s Footrule can be used. Table 7.1 summarizes various quantities within this setting. However, in what follows, we argue that using these methods, i.e., the rank-distance and rank aggregation methods may or may not serve our purpose of measuring the variance and bias of a model in terms of ranking error.

**Table 7.1: Using permutations of documents, instantiations of different quantities.**

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Instantiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>True label, ( y )</td>
<td>( \pi(l_q) )</td>
</tr>
<tr>
<td>Predicted label of a model, ( \hat{y}(x) )</td>
<td>( \pi(f(x_q)) )</td>
</tr>
<tr>
<td>Systematic prediction, ( \hat{y}^*(x) )</td>
<td>( \mathbb{E}_D[\pi(f_D(x_q))] )</td>
</tr>
<tr>
<td>loss(( \hat{y}^*(x), y ))</td>
<td>rank_distance(( \pi(l_q), \mathbb{E}_D[\pi(f_D(x_q))] ))</td>
</tr>
<tr>
<td>loss(( \hat{y}(x), \hat{y}^*(x) ))</td>
<td>rank_distance(( \mathbb{E}_D[\pi(f_D(x_q))], \pi(f(x_q)) ))</td>
</tr>
</tbody>
</table>

The bias (aka systematic error) and variance should reflect the error induced by an evaluation metric that befits the problem at hand (which is, in our case, the IR ranking problem). In IR community, the nearly-ubiquitous practice regarding evaluation of a candidate ranked list of documents produced by an IR system against a gold-standard ranked list is to use a metric that incorporates per-document relevance judgements. For example, Moffat [107] studies thirteen IR metrics that have been heavily used within the IR community (later Jones et al. [84], among others, also conduct such a study), and all of these metrics invariably exploit the relevance judgements of individual documents in their definitions. In contrast, Kendall’s tau and other similar rank-distance metrics are popular choices in other domains of IR such as measuring the effectiveness of IR metrics [23, 168], and deciding whether two test collections are equivalent [153].

14The symbol \( \pi \) is a bijection on the set of items \( \{1, 2, \ldots, n_q\} \), i.e., \( \pi : \{1, 2, \ldots, n_q\} \rightarrow \{1, 2, \ldots, n_q\} \).
Chapter 7: Understanding Bias and Variance of Rank-Learners

148

Candidate ranked list
Many equally correct forms of true ranked list
IR metric calculator
IR metric calculator
Compare (with emphasis on top docs)
Quality of the candidate ranked list
IR metric
IR metric
Select any of the ranked lists
Aggregate different lists
IR metric calculator
IR metric calculator
Compare (with no emphasis on top docs)
Quality of the candidate ranked list
Aggregate different lists
IR metric
IR metric

Figure 7.3: Evaluating a candidate ranked list: comparison between the conventional IR metrics (left figure) and the rank-aggregation metrics (right figure).

Now the question is, why not use the rank-distance and aggregation metrics instead of IR metrics? We argue that the IR researchers have not adapted rank-distance and rank-aggregation measures for IR evaluation due to the reasons as described next. The rank-distance and rank-aggregation metrics usually assume that every pair of items in a ranked list has a preference relationship. This, however, does not reflect the practice of IR ranking domain because conventionally only a few distinct labels are assigned to a large number of documents; thus in IR ranking there is no preference relationship in a large number of document pairs (pertaining to a query). Regarding the rank-aggregation methods, due to the same reason, to make a gold standard list of documents, an exponentially large number of candidate lists need to be aggregated, which is less likely to be computationally feasible. In contrast, the conventional IR metrics (e.g., DCG) represent many equivalent ranked lists by a single value, whereas the rank-aggregation metrics find it difficult (due to an exponentially large number of candidate lists and/or the heuristic nature of the algorithm) to combine many equivalent lists. Figure 7.3 illustrates the simplicity of IR metrics over the rank-distance and rank-aggregation metrics in evaluating the quality of a candidate ranked list of documents produced by an IR system.

In addition, neither the rank-distance methods nor the rank aggregation methods usually emphasize the top portion of a ranked list. While some rank-distance metrics such as Kendall’s tau can incorporate positional weight [168], it is not immediately clear as to how these metrics relate to the conventional IR metrics such as NDCG, and we have not found any such study in the literature. In contrast, IR metrics such as DCG has

---

15It is relatively straightforward to define an “IR evaluation metric” based on the rank-distance metrics by negating (or inverting) the value of the distance between the candidate list and the gold-standard list. A single gold-standard list may be produced from many equivalent (gold-standard) lists by applying a rank-aggregation method on the equivalent lists.

16Cf. the 5th and 9th rows of Table 2.2.

17It is known that producing an optimal aggregation, i.e., finding an aggregate list which minimises the number of miss-ranked pairs across all candidate lists is NP-hard problem for even only four lists [45]. (Although the limited number of distinct labels in IR domain may reduce this time complexity, the solution is not immediately clear.) Eventually heuristic algorithms are used in the literature to solve the problem which come with their own limitations.
a plethora of such studies such as [2, 135] which validate the efficacy of these metrics based on real user satisfaction.

In essence, due to the mismatch between the IR evaluation approach and the rank-distance and rank-aggregation based (hypothetical) evaluation approach, we believe that using the rank-distance and aggregation metrics to define bias-variance (in terms of ranking error) may not reflect the true characteristics of bias and variance in the IR ranking domain. We, therefore, believe that using the IR evaluation metrics which exploit relevance judgements of individual documents for bias and variance formulations of ranking problem would be more appropriate. That said, analyzing bias and variance using the rank-distance and rank-aggregation methods is an interesting research direction, that we leave to future work.

A question that may be asked is as to why in the regression domain the above-discussed problem is not present. We note here that in the regression problem, the deviation of a predicted label from the ground-truth label is well-defined (i.e., the algebraic difference), and so is the method of aggregating multiple predictions (i.e., the arithmetic mean). That is why the bias-variance decomposition of squared loss function is comparatively easier. In ranking, however, as explained above that both of these two procedures are complicated.

### 7.8.3 Bias-Variance Using the Dissimilarity between Score Lists

We now introduce the estimations of bias and variance that make direct use of the document labels instead of using the permutations of documents. We consider two variations in this setting: (a) using the score lists, and (b) directly using IR metrics. These two settings are described in this and next subsection respectively.

In IR ranking, the training set consists of a set of queries $Q$. Hence for a query (and associated documents), the loss can be defined as the deviation between the list of scores predicted by a model and the corresponding list of ground truth labels of the said documents. Within this setting, the true label, the predicted label and the systematic prediction for an instance is denoted by $\vec{l}_q$, $\vec{f}(x_q)$, and $\mathbb{E}_D[\vec{f}_D(x_q)]$ respectively, where $\vec{f}_D(x_q)$ is the predicted score list of query $q$ by model $f$ learnt from dataset $D$.

As such, the ranking loss at a query $q$ can be written as:

$$Err_{\text{ranking}}(q) = \text{loss}(\vec{f}(x_q), \vec{l}_q).$$  \hfill (7.16)
Table 7.2: Using score lists of documents, instantiations of different quantities.

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Instantiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>True label, (y)</td>
<td>(l_q)</td>
</tr>
<tr>
<td>Predicted label of a model, (\hat{y}(x))</td>
<td>(\hat{f}(x_q))</td>
</tr>
<tr>
<td>Systematic prediction, (\hat{y}^*(x))</td>
<td>(\mathbb{E}_D[\hat{f}_D(x_q)])</td>
</tr>
<tr>
<td>loss((\hat{y}^*(x), y))</td>
<td>score_distance((\mathbb{E}_D[\hat{f}_D(x_q)], \overline{l_q}))</td>
</tr>
<tr>
<td>loss((\hat{y}(x), \hat{y}^*(x)))</td>
<td>score_distance((\mathbb{E}_D[\hat{f}_D(x_q)], \hat{f}(x_q)))</td>
</tr>
</tbody>
</table>

To define the systematic prediction, the scores \(\hat{f}(x_q)\) across different samples could be averaged. To measure the \(loss(.)\) between two score lists, reciprocal (or inverted) values of the standard correlation methods such as Spearman’s correlation coefficient can then be used. Table 7.2 summarizes various quantities within this setting. This could be called as *hybrid* analysis as it incorporates both pointwise and listwise information. However, this approach has a few problems which are as follows. (1) This approach is confined to only pointwise rank-learners as the listwise algorithms do not estimate the individual relevance judgments, but rather they estimate entire ranked lists. (2) Even when using pointwise algorithms, the limitations of pointwise analysis also apply here (of course, to a lesser degree than the pointwise analysis) because of using (limited) listwise information. For example, here the variance measurement measures the fluctuation in individual scores of documents (of a particular query), but we are interested in measuring the variability in predictions in terms of IR evaluation metrics. For these reasons we do not consider this setting further.

### 7.8.4 Bias-Variance Using an IR Metric-based Loss

Like the previous setting here the true label, the predicted label and the systematic prediction for an instance is denoted by \(l_q\), \(\hat{f}(x_q)\), and \(\mathbb{E}_D[\hat{f}_D(x_q)]\) respectively.

In the previous two subsections we have argued that the very (IR) metrics used in evaluating a ranked list of documents should be taken into account when measuring the variability in predictions in terms of ranking error and the systematic ranking error of an estimator. To this end, we propose that in order to measure the deviation of one ranked list from another, we make direct use of an IR metric such as NDCG as detailed below.

Within this setting, since the maximum possible NDCG for a query is 1, the ranking loss at query \(q\) takes the following form:

\[
Err_{\text{ranking}}(q) = \text{loss}(1, \text{NDCG}(\overline{l_q}; \overline{f})) = 1 - \text{NDCG}(\overline{l_q}; \overline{f}).
\]
We now define the estimator of the systematic ranking error at a query \( q \) as the difference between 1 (i.e., the maximum possible NDCG) and the NDCG of the document list ordered by the scores of the systematic prediction (i.e., \( \hat{f}(x_q) = \mathbb{E}_D[\hat{f}_D(x_q)] \)). As such, the systematic ranking error (SRE) or the bias of a rank-learner at a query \( q \) becomes:

\[
SRE(q) = 1 - \text{NDCG}(\hat{l}_q; \hat{f}).
\] (7.18)

We could then define the estimator of the error due to variability in rank-predictions (i.e., the expected value of the “remaining error” over all possible models built on different training samples of a fixed size) at a query \( q \) as: \( VRE(q) = \mathbb{E}_D[\text{NDCG}(\hat{l}_q; \hat{f}_D) - \text{NDCG}(\hat{l}_q; \hat{f})] \), where VRE denotes the variability in predictions in terms of ranking error. The assumption behind this variance definition is that the ranking error for a query \( 1 - \text{NDCG}(\hat{l}_q; \hat{f}_D) \) is decomposed into bias plus variance, i.e., adding the terms SRE(\( q \)) and VRE(\( q \)) gives the expected error at a query which is \( \mathbb{E}_D[1 - \text{NDCG}(\hat{l}_q; \hat{f}_D)] \).\(^{18}\)

However, a drawback of this variance definition is that the variance for a query may be systematically underestimated since for some datasets (over which the Expectation is being performed) this difference could be negative; because it may happen that the \( \text{NDCG}(\hat{l}_q; \hat{f}_D) \) value for a particular training dataset \( D_i \) (out of the many which are used in the procedure of estimating bias and variance) is higher than \( \text{NDCG}(\hat{l}_q; \hat{f}) \).

It would be tempting to define the variance as the squared (or absolute) difference between the NDCG of the systematic prediction and the prediction of an individual model, expected over the samples, i.e., as: \( VRE(q) = \mathbb{E}_D[(\text{NDCG}(\hat{l}_q; \hat{f}_D) - \text{NDCG}(\hat{l}_q; \hat{f}))^2] \), or, alternatively, \( VRE(q) = \mathbb{E}_D[|\text{NDCG}(\hat{l}_q; \hat{f}_D) - \text{NDCG}(\hat{l}_q; \hat{f})|] \). Doing so, however, would exacerbate the problem since it would turn any negative cost into a positive one. Elaborately, as mentioned in the previous paragraph that it may happen that the \( \text{NDCG}(\hat{l}_q; \hat{f}_D) \) of a particular sample is higher than \( \text{NDCG}(\hat{l}_q; \hat{f}) \). This means that this incident of observing a higher \( \text{NDCG}(\hat{l}_q; \hat{f}_D) \) is a positive outcome, but the above-mentioned formulae consider it as an error component by adding the (squared or absolute) difference between \( \text{NDCG}(\hat{l}_q; \hat{f}_D) \) and \( \text{NDCG}(\hat{l}_q; \hat{f}) \) in the variance estimator.\(^{19}\) Considering this problem, we propose our first definition for the error due to

\(^{18}\)A pitfall here would be to define the expected ranking loss: \( \text{Err}_{\text{ranking}}(q) = (1 - \text{NDCG}(q))^2 \). At a first glance this definition may look appealing because we could use the similar derivation of the regression problem (cf. Equation 7.2) to decompose the squared bias as \( (1 - \mathbb{E}_D[\text{NDCG}(\hat{l}_q; \hat{f}_D)])^2 \) and variance as \( \mathbb{E}_D[(1 - \mathbb{E}_D[\text{NDCG}(\hat{l}_q; \hat{f}_D)])^2] \). However, this setting has a major problem: the systematic prediction, i.e., the quantity \( \mathbb{E}_D[\text{NDCG}(\hat{l}_q; \hat{f}_D)] \) is in fact not of our interest. The reason is, \( \mathbb{E}_D[\text{NDCG}(\hat{l}_q; \hat{f}_D)] \) (which is estimated by \( NDCG(\hat{l}_q; \hat{f}_D) = \frac{1}{B} \sum_{b=1}^{B} \text{NDCG}(q) \) where \( B \) is the number of training samples) does not measure the average predictions of the models, but rather it simply measures the average quality of ranking performance after applying an IR evaluation measure on predictions of individual models.

\(^{19}\)A natural question to ask is why in the regression setting this phenomenon is not a problem? We note that in the regression setting, predicting both the greater or less than the target value is an error. But in the ranking problem, observing a greater NDCG is always a positive outcome. Elaborately, in the regression problem, the variance estimator tells us how much variation we expect to observe when the model is learnt from a different sample. Here both the positive and negative fluctuations, i.e., both the
variability in rank-prediction at a query \( q \) as follows:

\[
VRE(q) = \mathbb{E}_D[|NDCG(\vec{l}_q; \hat{f}) - NDCG_{f_D}(\vec{l}_q; \hat{f}_D)|_+],
\]

(7.19)

where \(|a|_+\) denotes the positive only, i.e., \(|a|_+ = \max(0, a)\).

It is evident from this definition that the problems of the previous (potential) definitions are not present here. However, the ranking loss is no longer assumed to be decomposable into the bias and variance (because of the \(|\cdot|_+\) operator). We henceforth call this definition as \( VRE_{additive} \) since this definition is nonetheless close to the additive decomposition of ranking error.

We now propose a second variance definition. Recall that a variance estimator is defined as \( \mathbb{E}_D[loss(\hat{y}(x), \hat{y}^*(x))] \) where \( \hat{y}(x) \) is the prediction of a generic instance \( x \) and \( \hat{y}^*(x) \) is the systematic prediction of \( x \). We now directly use NDCG to calculate an estimate of the variance as follows. We assume that the systematic prediction \( \bar{f} \) produces the best ranking, so we assign new “ground truth” labels to the documents for each query according to the ranking induced by the scores \( \bar{f} \). Mathematically:

\[
VRE(q) = \mathbb{E}_D[1 - NDCG(\vec{l}_q; \bar{f})]
\]

(7.20)

\[
\vec{l}_q; \bar{f}) = \arg \max_{\vec{l}_q \in \Pi(\vec{l}_q)} NDCG(\vec{l}_q; \bar{f}),
\]

where \( \Pi(\vec{l}_q) \) denotes the set of all reordering (permutations) on the vector of labels \( \vec{l}_q \).

The documents are first sorted in descending order of the scores given by \( \bar{f} \), and then new “ground truth” labels (taken from the original set of labels \( \vec{l}_q \)) are assigned to these sorted documents starting from the top. We henceforth call this definition as \( VRE_{direct} \) since this definition directly measures the error due to the variability in systematic rank-prediction.

**Table 7.3:** Using IR metrics on score lists of documents, instantiations of different quantities.

<table>
<thead>
<tr>
<th>Quantity</th>
<th>Instantiation</th>
</tr>
</thead>
<tbody>
<tr>
<td>True label, ( y )</td>
<td>( \vec{l}_q )</td>
</tr>
<tr>
<td>Predicted label of a model, ( \hat{y}(x) )</td>
<td>( \hat{f}(x) )</td>
</tr>
<tr>
<td>Systematic prediction, ( \hat{y}^*(x) )</td>
<td>( \mathbb{E}_D[\hat{f}_D(x)] )</td>
</tr>
<tr>
<td>( loss(\hat{y}(x), y) )</td>
<td>( 1 - NDCG(\vec{l}_q; \hat{f}_D) - NDCG(\vec{l}_q; \hat{f}_D) )</td>
</tr>
<tr>
<td>( loss(\hat{y}(x), \hat{y}^*(x)) )</td>
<td>( a. \</td>
</tr>
<tr>
<td></td>
<td>( b. \ 1 - NDCG(\vec{l}_q; \hat{f}_D) )</td>
</tr>
</tbody>
</table>

higher and lower predictions than the true label are condemnable because the goal is to predict the exact target value. In the ranking problem, it is true that the definition \( \mathbb{E}_D[(NDCG(\vec{l}_q; f_D) - NDCG(\vec{l}_q; \hat{f})|^2] \) of the variance estimator tells us as to how much fluctuation in predictions in terms of NDCG we expect to observe across different samples. But unlike regression, here if a particular model yields a higher NDCG than the “systematic NDCG” (i.e., \( NDCG(\vec{l}_q; \bar{f}) \)), this is in fact a positive outcome, so we should not consider it as an error to be contributed to the variance term.
Table 7.3 summarizes the various quantities defined according to the above-mentioned discussion. Figure 7.3 illustrates the instantiations of two types of losses in ranking domain. Figure 7.5 pictorially describes the procedure for computing SRE and VRE for a query.

![Figure 7.4: Bias and variance estimation for ranking error.](image)

It is plausible that the approach of estimating the bias using SRE and the variance using VRE outlined above is able to analyse the systematic error and the error due to variability in prediction of a model in terms of ranking error. However, this setting has a problem from theoretical perspective: according to the desiderata listed by Kohavi and Wolpert [86], for a variance definition for a generic loss function, the variance must be independent of the ground truth labels (later on, James [81] suggested similar properties). In contrast, our variance definitions (Equations 7.19 and 7.20) make use of ground truth labels. This problem is, however, difficult to avoid as it has been discussed earlier that almost all of the IR metrics make direct use of relevance labels for documents. Apart from this caveat, this framework essentially avoids the problems of the two previously discussed approaches outlined in Sections 7.8.2 and 7.8.3. Using this framework we are able to measure the systematic ranking error of a rank-learner across the different training sets and the ranking error that results from the variation in predictions across the different training sets (conceptually similar to the variance).

An important merit of our framework developed in this section is that it is applicable to all types of LtR algorithms (pointwise, listwise etc.).

We discussed in Section 7.5 that to the best of our knowledge there is no investigation in the literature into decomposition of bias-variance for ranking error. The loss function of squared error has nice mathematical properties (which are mostly absent in zero-one loss of classification) that make it easy to decompose. For the classification problem which is a mature field of research, there is a multitude of decompositions, and some of them take the form of bias-variance. However, this is not the case for ranking. We note that the definition of $VRE_{direct}$ only uses the count of the different relevance labels (for each query), not the labels themselves. So $VRE_{direct}$ almost fits the requirement of a standard variance definition.
them are somewhat contradictory to one another [41]. We have argued that the true
loss function of ranking problem is even more complicated since it involves a group of
instances (documents) as opposed to a single instance as in the case for zero-one loss,
and also the scores themselves are not the main targets for predictions. Besides our
formulations developed above, there might be other options for defining the bias and
variance such as the one discussed in Section 7.8.2. Also, it still remains an open question
as to how exactly the bias and variance decompose the ranking error.

7.8.5 Formulae for Estimating Listwise Bias and Variance

In this section we describe the exact formulae that we use for estimating VRE and SRE.

7.8.5.1 Estimating Quantities Using Bootstrap Samples

As was the case for the pointwise analysis, we estimate SRE and VRE using two methods:
(1) bootstrap samples (this subsection), and (2) repeated twofold cross-validation (the
next subsection). In this subsection, we assume that $B$ bootstrap samples (without
replacement) $\{D_1, D_2, \ldots, D_B\}$ are generated from the original training set $D$.

**Systematic Ranking Error (SRE):** $SRE(q)$ is measured as the difference between 1
and the ranking performance of the systematic predictions. Concretely:

$$SRE_{\text{bootstrap}} = \frac{1}{|Q|} \sum_{l=1}^{|Q|} S\hat{R}E_{\text{bootstrap}}(q_l)$$  \hspace{1cm} (7.21)

where $S\hat{R}E_{\text{bootstrap}}(q_l) = (1 - NDCG(q_l; \bar{f}))$,

where $NDCG(q_l; \bar{f})$ is calculated on the list ranked by the average scores across all $B$
learners (built from samples $D_i, 1 \leq i \leq B$), i.e., by $\bar{f}(q_l) = \frac{1}{B} \sum_{i=1}^B \hat{f}_{D_i}(q_l)$.
Chapter 7: Understanding Bias and Variance of Rank-Learners

Variability in Rank-prediction Error (VRE): VRE of a random forest is estimated by averaging $VRE(q)$ values over all queries.

$VRE_{\text{additive}}$ for a query is estimated as the positive differences between the ranking performance of systematic prediction and the ranking performance of the individual models, averaged over multiple models. Concretely:

$$\hat{VRE}_{\text{additive}}^{\text{bootstrap}}(q_l) = \frac{1}{B-1} \sum_{i=1}^{B} |\text{NDCG}(q_l; \bar{f}) - \text{NDCG}(q_l; \vec{f}_{D_i})|_+, \quad (7.22)$$

where $|a|_+ = \max(0, a)$, and $\text{NDCG}(q_l; \vec{f}_{D_i})$ is value for query $q_l$ computed from the prediction of $i$th model.

$VRE_{\text{direct}}$ for a query is estimated using the following equation:

$$\hat{VRE}_{\text{direct}}^{\text{bootstrap}}(q_l) = \frac{1}{B} \sum_{i=1}^{B} \text{NDCG}(\vec{\tilde{l}}_q; \vec{f}_{D_i}) \quad (7.23)$$

$$\vec{\tilde{l}}_q = \arg \max_{\vec{\tilde{l}}_q \in \Pi(\vec{l}_q)} \text{NDCG}(\vec{\tilde{l}}_q; \bar{f}(q_l)),$$

where $\Pi(\vec{l}_q)$ denotes the set of all reorderings (permutations) on the vector of labels $\vec{l}_q$.

Finally, $\hat{VRE}_{\text{bootstrap}} = \frac{1}{|Q|} \sum_{l=1}^{|Q|} \hat{VRE}_{\text{bootstrap}}(q_l)$.

7.8.5.2 Estimating Quantities Using Repeated Two-Fold Cross-Validation

Recall that within this setting we generate two disjoint training sets $D_1$ and $D_2$ from the original set $D$, and we repeat this process $J$ times that results in samples $\{D_{11}, D_{12}, D_{21}, D_{22}, \ldots, D_{1J}, D_{2J}\}$ where $D_{1j} \cap D_{2j} = \emptyset$.

Systematic Ranking Error (SRE): With $1 \leq j \leq J$, the $j$th estimate of SRE of the model is calculated as:

$$\hat{SRE}_{\text{twofold}}(q_l) = \frac{1}{|Q|} \sum_{i=1}^{|Q|} \hat{SRE}_{\text{twofold}}(q_l) \quad (7.24)$$

where $\hat{SRE}_{\text{twofold}}(q_l) = (1 - \text{NDCG}(q_l; \bar{f}_j))$, $\bar{f}_j = \frac{1}{2} \sum_{i=1}^{2} f_{D_{ij}}(q_l)$.

Finally, $\hat{SRE}_{\text{twofold}} = \frac{1}{J} \sum_{j=1}^{J} \hat{SRE}_{\text{twofold}}$.

Note that in this equation we have divided the total variance by $B$ instead of $B - 1$ because here we are computing a simple average of individual estimates of errors, i.e., we do not need to make the variance estimator “unbiased”.

21
Variability in Ranking Error (VRE):

The $j$th estimate for $VRE_{\text{additive}}$ of the model is given by:

$$
\hat{VRE}_{\text{additive\ twofold}}^{j} = \frac{1}{|Q|} \sum_{l=1}^{|Q|} \hat{VRE}_{\text{additive\ twofold}}^{j}(q_l),
$$

(7.25)

where,

$$
\hat{VRE}_{\text{additive\ twofold}}^{j}(q_l) = \frac{1}{2} \left( 2 - \sum_{i=1}^{2} |\text{NDCG}(q_l; \vec{f}_j) - \text{NDCG}(q_l; \vec{f}_{D_{ij}})|_+ \right).
$$

(7.26)

Likewise, the $j$th estimate for $VRE_{\text{direct}}$ of the model is given by:

$$
\hat{VRE}_{\text{direct\ twofold}}^{j} = \frac{1}{|Q|} \sum_{l=1}^{|Q|} \hat{VRE}_{\text{direct\ twofold}}^{j}(q_l),
$$

(7.27)

where,

$$
\hat{VRE}_{\text{direct\ twofold}}^{j}(q_l) = \frac{1}{2} \sum_{i=1}^{2} \text{NDCG}((\vec{l}^{*}, q_l); \vec{f}_{D_{ij}}).
$$

(7.28)

Finally, $\hat{VRE}_{\text{twofold}} = \frac{1}{J} \sum_{j=1}^{J} VRE_{\text{twofold}}^{j}$.

### 7.9 Effect of Sub-sample Size (Per-Tree) on Bias-Variance

In this section we examine how the bias and variance react while varying a parameter of the learning algorithm of our interest, namely RF-point. Chapter 4 revealed that proper tuning of the sub-sample size per tree in the context of LtR may yield better accuracy, and at the same time allows for scaling up the algorithm for large datasets. We take this parameter for our analysis.

In bagging [14] (which is known as a predecessor of a random forest), the tendency was to reduce training set similarity across different trees by using a bootstrapped sample (i.e., using around 63% data) to learn each tree. A random forest keeps the default setting of bagging with regard to sampling per tree, but exploits another dimension in its favor which is the number of candidate features at each node. At an intuitive level, one might be tempted to think that a bagged ensemble, by reducing the training sample per tree further below 63%, could have achieved the same reduction of $\rho$ as achieved by a random forest. However, Friedman and Hall [54] show that if less than 50% data are used, it does not decrease the error rate of bagging, even though it does reduce $\rho$ (we note that reducing correlation generally increases bias and variance of individual base
learners, i.e., trees). However, the effect of a random forest’s reduction of $\rho$ (by using another dimension which is the number of candidate features at each node) empirically turns out to be favorable to reducing generalization error.

Then comes our idea of Chapter 4: can we achieve better performance by holding the best configuration of a random forest in terms of number of candidate features, and then by going backward to the initial motivation of bagging (which was to decrease the commonalities amongst sub-samples used to learn individual trees)? Once again, the main goal is to reduce $\rho$ (and thereby variance of ensemble of sub-sampled trees) but in a different way from both the bagging and random forest. In doing so, however, it gives rise to the risk of increasing individual tree variance and bias, thereby warranting proper tuning of the parameter.

In the results that follow, for the method of bootstrapping we use $B = 10$ and for the method of repeated twofold CV we use $J = 5$. The MSLR-WEB10K (fold 1) and Yahoo datasets are used for the experiments because theoretical result are usually pronounced with big datasets.

### 7.9.1 Results: Pointwise Analysis

This section analyses the plots of bias and variance estimated using the pointwise analysis. Figure 7.6 shows the plots for the method of bootstrapping and repeated twofold...
Chapter 7: Understanding Bias and Variance of Rank-Learners

The trends of plots of the MSLR-WEB10K and Yahoo datasets are largely similar to each other. They are also similar across Methods 1 and 2, which implies that both the methods are effective to capture the trend of bias and variance.

Broadly, for both the datasets the plots corroborate our hypothesis mentioned earlier that bias and variance would trade-off with increasing $p$.

As for the ensemble variance, increasing $p$ which decreases the individual tree variances, does not necessarily translate into decreased ensemble variance because besides the single tree variance the correlation is the other factor in deciding the ensemble variance (cf. Equation 7.5). It is due to the increased correlation the ensemble variance keeps increasing as we increase the sub-sample size per tree.

We know that the bias does not depend on the peculiarities of the training set, whereas the variance does. Given that the training set sizes of both the methods of bootstrap and repeated twofold CV are sufficiently close to each other (containing approximately 63% and 50% of original sample respectively), the bias in both the methods has been found to be of similar absolute values (from $\approx 0.56$ to $\approx 0.6$). The small changes in bias across the two methods are likely due to randomness and the small change in training set size. But the variances do differ greatly ($\approx 3.5$ times for the last configurations), and the difference is according to our conjecture made earlier which is, the estimate of method of repeated twofold CV is greater than that of method of bootstrapping as the former’s training sets are disjoint whereas the latter’s are not. The increasing trend of the variance estimates, however, is captured by both the methods, which implies that any of the two methods can be used to visualize the trend.

The ensemble variance with very small $p$ is still quite small which may seem to be counter-intuitive. Wager et al. [155] show for a regression dataset that reducing the correlation greatly (by using a very small $K$) causes high ensemble variance. The apparent discrepancy between their findings and ours can be explained by the fact that they used a quite smaller dataset (20640 instances and 8 features) whereas the minimum configuration of our setting uses 5% queries per tree which is 300 queries for MSLR-WEB10K (with 120 documents per query) and 23000 queries for Yahoo datasets (with 23 documents per query), and 136 and 519 features respectively. Hence in our case reducing sample size per tree does not heavily increase the individual tree variances, thereby resulting in comparatively low ensemble variance. In contrast, in their study, when they...
drastically reduce \( K \), the reduction in correlation alone cannot limit the significantly higher individual tree variance, thereby resulting in higher ensemble variance.

### 7.9.2 Results: Listwise Analysis

This section analyses plots of SRE and VRE quantities. We see that the trends of the plots are according to our conjecture that variance and bias trade-off with increasing sub-sample size per tree, which corroborates that our formulations for analysing bias and variance of ranking error (SRE and VRE) are indeed reliable.

Also, like the pointwise analysis, the trends of the plots of method of repeated twofold CV are similar to that of method of bootstrapping. The absolute values of SRE are similar across different methods, which is expected. The values of VRE estimates of method of bootstrapping are less than that of method of repeated twofold CV for both the versions of VRE, which is also expected. The value of SRE on Yahoo dataset is less than that on MSLR-WEB10K (approximately half). The VRE, however, maintains comparable values across the two datasets.

On the MSLR-WEB10K, SRE appears to flatten off and even increase slightly for large values of \( p \). However, in the pointwise analysis (cf. Figure 7.6) the bias was found to be ever-decreasing with increasing \( p \), which means that the scores of individual documents, on average, are better predicted with increasing \( p \). A possible explanation for this...
apparently different findings is that although the optimal (minimal) mean-squared error results in optimal (maximal) NDCG, but it is not necessarily the case that reducing mean-squared error will always increase NDCG. An example of this is simple to imagine. Another perspective for explaining this slightly increasing trend of SRE on MSLR dataset is that the error rate heavily depends on SRE (since VRE is quite small), so the trend of the SRE curve almost mimics the ranking error curve (cf. Figure 7.8).

7.10 Listwise Bias and Variance Comparison of Random Forest and LambdaMart

Table 7.4: Comparison between RF-point (RF-p) and LambdaMart (LMart) using SRE and VRE on MSLR-WEB10K and Yahoo datasets.

So far we have used only the RF-point algorithm. This section performs a small comparative study of RF-point and LambdaMart algorithms using the SRE and VRE estimates. We compute SRE and VRE using the method of repeated twofold CV on MSLR-WEB10K and Yahoo datasets – we preferred the method of repeated twofold
CV because LambdaMart is a deterministic algorithm, so using the method of bootstrapping (which uses overlapped samples) might greatly underestimate the variance of LambdaMart.

Table 7.4 shows the results. We see that on both the datasets for both the algorithms, SRE is the main contributor to the error rate. However, this does not necessarily mean that VRE is unimportant since on the Yahoo dataset the SRE of LambdaMart is higher than that of RF-point, but due to the opposite trend in VRE, the error rate of LambdaMart is lower than that of RF-point.

7.11 Discussion

Brain and Webb [13] advocate for designing algorithms that meet the specific need of the dataset in terms of bias-variance. Our investigation supports their recommendation that for big datasets variance reduction may not need to be the key goal. In our experiments, the value of SRE has been found to be far greater than VRE which implies that SRE is the main contributor to the ranking error rather than VRE. This is a probable explanation as to why RF-based algorithms were outperformed by LambdaMart on MSLR-WEB10K dataset (cf. Table 6.12) since the underlying framework of LambdaMart, namely gradient boosting mainly focuses on bias reduction.

Many of the existing methods (such as some of which discussed in Section 7.5) investigate as to how to reduce correlation among the trees and, at the same time, to increase strength of individual trees. Our analysis opens another perspective to view this aspect which is through the bias-variance relationship. At a higher level we can view this dilemma as the classical bias-variance trade-off.

Given that absolute values of bias differs in the two investigated datasets, an interesting research direction would be to investigate what aspects of an LtR dataset causes an LtR algorithm to have higher/lower bias-variance. Another interesting direction for extending this work is to investigate alternative definitions of listwise bias and variance and decompositions of ranking error as indicated in Section 7.8.2. Also, further research is required to find out which of our proposed listwise variance definitions are more reliable. Yet another direction for future work is to conduct a thorough study to explain relative performance of different LtR algorithms using the listwise bias and variance.
7.12 Conclusion

This chapter presented what we believe to be the first investigation into the bias-variance analysis of LtR algorithms with an emphasize on the random forest based rank-learners. We have analyzed bias and variance from both pointwise and listwise perspectives. Our formulations of bias and variance were found to be working well in practice in the sense that we observed the classical bias-variance tradeoff for varying the parameter sub-sample size per tree.
Chapter 8

Conclusion

Since its inception, use of learning-to-rank systems are becoming ubiquitous in a plethora of domains in the broad field of information retrieval. Although the accuracy of the current LtR algorithms is satisfactory, we believe there is still a large amount of room for improvement. Moreover, computational efficiency and theoretical understanding of these algorithms are often overlooked. In this thesis an attempt has been made to contribute in these fields with an effective and efficient supervised learning framework, namely the random forest.

In this chapter we explain how the research goals that were set out in Chapter 1 have been addressed in the thesis. We then summarize the major findings of this research. This is followed by several specific practical recommendations regarding RF-based LtR algorithms that have emerged from this research. Finally, avenues for extending this research are explored.

8.1 Addressing Research Goals

In Chapter 1 we listed the goals of this research. The first goal was to investigate scalability issues of RF-based rank-learners. We have addressed this problem in Chapters 3 and 4. In Chapter 3 we successfully utilized the hierarchical structure of RF framework to improve scalability. In Chapter 4 based on the theory of correlation between the trees, we improved both scalability and performance of these algorithms.

The second goal was to design novel RF-based LtR algorithms. In Chapter 5 we developed a listwise, a hybrid and a pairwise algorithm (namely RF-list, RF-hybrid and RF-pair respectively), and adapted another one to LtR, namely a random-splitting approach (RF-rand). We thoroughly investigated their performance trade-offs, and analysed their
strengths and weaknesses. In Chapter 6 we further investigated a variety of approaches for improving performance of RF-based pointwise algorithms.

The third goal was to identify stable LtR algorithms across datasets with diverse properties. In Chapter 6 we conducted extensive experiments on eight publicly available datasets using a number of state-of-the-art algorithms, and identified the robust algorithms among them.

The fourth goal was to develop a better theoretical understanding of RF-based rank-learners. In Chapter 7 by formalising the bias and variance of generic rank-learners, we developed a framework for explaining ranking performance in terms of the systematic predictions and overfitting aspects of the model. We also examined the empirical bias-variance profiles of an RF-based algorithm.

8.2 Major Findings

Below we summarize the major findings that have resulted from this research.

- **Undersampling non-relevant documents** for highly imbalanced LtR training sets using random undersampling results in improved training time for generic LtR algorithms without significant degradation in performance. [Chapter 3]

- The random undersampling approach which preserves the true distribution of non-relevant documents in the undersampled training set appears to outperform a deterministic undersampling which selectively includes the non-relevant documents. [Chapter 3]

- We have been able to further improve the efficacy of the above-mentioned undersampling approach in conjunction with the random forest based LtR algorithms by performing undersampling at the tree-level instead of at ensemble level. By doing this, the possible information loss (due to the downsampling of non-relevant documents) has been greatly minimized. [Chapter 3]

- **Reducing correlation** between the trees by using less training data per tree exhibits a twofold benefit: (1) it has drastically reduced the training time (and slightly reduces evaluation time), and (2) for the smaller datasets it has improved accuracy. For the larger datasets, although we have not observed improvement in performance over the baseline setting (i.e., using a bootstrapped sample per tree), this technique can still be recommended due to the reduction in training time. [Chapter 4]
• By measuring the amount of correlation between the trees and variance of individual trees we have validated our hypothesis that the sub-sampling method indeed reduces the correlation, which in turn appears to reduce the ensemble variance. [Chapter 4]

• Our listwise algorithm based on the random forest directly optimises a rank-based metric instead of a surrogate metric, and has demonstrated better performance on several datasets. However, this has come at a cost of increased training time (which is also true for many state-of-the-art LtR algorithms which directly optimise a rank-based metric). [Chapter 5]

• Different splitting criteria can be combined successfully in a single tree using our hybrid algorithm. This has been shown to be true for multiple datasets. This technique is especially useful when a splitting criterion (like the listwise splitting) is computationally demanding. [Chapter 5]

• While our pairwise greedy algorithm based on RF has offered slight performance improvement over its pointwise counterpart on smaller datasets, on the larger datasets its performance has not been found to be comparable. It is thus comparatively less attractive for further investigation. [Chapter 5]

• Completely random splitting criterion has yielded better performance than expected (but not better than the pointwise splitting), especially on the smaller datasets. This algorithm offers a large computational advantage. [Chapter 5]

• Although a concern may be raised that the listwise algorithm may be prone to overfitting, our experiments has demonstrated that it is mostly not the case. This reaffirms the predominant opinion (in classification and regression realm) that using unlimited trees in a random forest does not usually cause overfitting. [Chapter 5]

• Our investigation has suggested that the depth of a tree plays an important role in performance of RF-based rank-learners. For instances, we have found that the deeper a tree is grown, the less important the distinction between the various objective functions. [Chapter 5]

• Our investigation into the entropy versus mean-squared error based splitting criterion with the RF-based pointwise algorithms demonstrated that for the smaller datasets the entropy-based splitting criterion appears to be performing slightly better than the mean-squared error based splitting. However, as the training data grows, the mean-squared error based splitting appears to outperform the entropy-based splitting. [Chapter 6]
For the big datasets, using a larger value of the parameter $K$ (i.e., the number of features to consider for a split) has resulted in significantly better performance. Specifically, in our experiment we have found that setting $K$ to approximately $1/4M$ outperforms the standard practice of using $\log(M) + 1$ or $\sqrt{M}$. For smaller datasets, however, the default value of $K$ has worked quite well; this is a positive finding due to the reduced need for parameter tuning. [Chapter 6]

Conventional unweighted aggregation of predictions of individual trees performs similar to a weighted average where the weights of the predictions are assigned based on the ranking accuracy of the corresponding trees.

In our extensive comparison of different LtR systems, no consistent winning algorithm was observed across the different datasets. For the smaller datasets, RF-point-S% (i.e., the sub-sampled version of the pointwise algorithm) was, on average, found to be the best among the investigated RF-based algorithms and other state-of-the-art systems, which is followed by RF-list-S%. [Chapter 6]

For relatively small datasets, the RF-based algorithms have demonstrated relatively robust performance across the datasets with diverse properties. This has been found both in experiments with the HP2004, TD2004, NP2004, MQ2007, MQ2008 and Ohsumed datasets, and in investigation of the learning curve on the two big datasets. This indicates that the RF-based LtR algorithms are promising candidates for domains where getting large amount of labelled training data can be difficult, such as domain-specific search and enterprise search. [Chapter 6]

On the two larger datasets, namely MSLR-WEB10K and Yahoo, LambdaMart and the RF-based algorithms have dominated the winners list. The efficacy of listwise objective appears to be dependent on the properties of the dataset. [Chapter 6]

In our investigation into the bias and variance profiles of RF-based LtR algorithms, the classical bias-variance trade-off has been observed for both classification and regression based objective functions. [Chapter 7]

8.3 Practical Recommendations

Based on this research we make the following suggestions to practitioners when using RF-based LtR algorithms in practice:

- When using the top-$k$ retrieval approach, including all non-relevant documents in the training set is not likely to greatly improve ranking performance of an LtR
algorithm. If the training dataset is highly imbalanced in terms of its relevance label distribution, and if the computational resources are limited, some non-relevant documents per query (per tree) can safely be removed (at random) from the training set; in the training set more queries should be included instead.

- The trees may be learnt using reduced samples in order to improve both performance and the learning time, irrespective of the splitting criteria (i.e., pointwise or listwise). The correct size of the sub-samples should be decided using the held-out data.

- If computational resources are sufficient, listwise splitting should be used to improve performance.

- If the dataset size is relatively small, the entropy-based splitting may be preferred, otherwise the mean-squared error based splitting is likely to yield better performance.

- If the dataset size is relatively large, a larger value of $K$ should be used (the value may be decided based on held-out data), otherwise the default values (i.e., $\log(M) + 1$ or $\sqrt{M}$) are likely to work well.

- For a new dataset, it is better to test many different algorithms and not only the most popular ones. In general the tree ensemble based algorithms, and in particular, the RF-based algorithms are likely to be among the top performers.

8.4 Extending this Research

This thesis opens a number of interesting research avenues which are discussed below:

- The prevalent practice among the researchers is to evaluate a dataset using average metric (e.g., NDCG@10) across all the queries of the test set. However, some queries are more difficult than others [3]. Our listwise objective function devised in Chapter 5 can easily be adapted to measure robustness as well as average accuracy by incorporating the variance term in the gain calculation.

- A hybridization of pointwise and listwise loss functions can be designed. One such idea is as follows. A few queries could be used to calculate a listwise loss, while the rest can be used to calculate a pointwise loss. The total loss can then be calculated as a combination of these two losses.

- To reduce the computational complexity of RF-list on big datasets, the listwise loss may be calculated locally based on the only the two leaves at hand. However,
a concern is that this may result in sub-optimal performance due to ignoring the rest of the leaves of a tree.

• In the learning phase of our listwise algorithm we have optimized NDCG which is a standard choice among the researchers due to its several merits (as discussed in Section 2.3. While Macdonald et al. [100] report that optimizing ERR does not give a better result, we still think it is worth investigating the effect of optimizing some other metrics besides NDCG in the RF-based listwise algorithms.

• Menze et al. [103] show some improvement using oblique splitting instead of axis-aligned splitting in classification and regression problems. Similar ideas can be applied to LtR. This has a theoretical appeal as using oblique cuts expands the hypothesis space thereby possibly decreasing model bias, and Brain and Webb [13] hint that for large dataset bias may be more influential in the error rate than variance.

• The missing values of an LtR training set are replaced by zero which has seemingly become a standard practice [94]. However, the random forest literature includes a large number of techniques to impute these values [68]. This may have a significant effect in performance as in some LtR datasets a large portion of the values are zeros (e.g., in Yahoo dataset 57% of the values are zero).

• We did not perform any post-processing of the learnt trees of the ensemble. Two immediate options are as follows. (1) Weighting the individual trees based on their performance on validation set (i.e., after they are learnt). However, a concern regarding this idea is that weighting may reduce the benefit of the diversity of an RF. (2) Refining the labels of the final data partitions (i.e., leaf nodes) using the validation set as advocated by Wager [154] for classification/regression task.

• Improving RF-based LtR algorithms may have a direct benefit for the boosting based LtR. For example, Mohan et al. [109] develop a gradient-boosted ensemble in a pointwise fashion (termed as IGBRT) that is built upon the predicted scores of an RF-based pointwise algorithm, and report better results than basic boosting and RF. As such, better predictions of a RF-based LtR algorithm (i.e., the initial stage of IGBRT) may improve the accuracy of IGBRT and similar algorithms.

• So far mainly the discriminative models have been used as learning models. We found only one work by Gupta [65] which tackles the LtR problem using generative models. Use of generative models for LtR task could be a direction for future research.

• Feature engineering can be performed manually, or using machine learning techniques [111]. Random forest-based feature selection has been found to be quite
effective in other domains such as bioinformatics and biomedical imaging [60]. In LtR, boosted tree ensemble based feature selection [118] and coordinate ascent based techniques [36] has started to get attention from the research community.

- The findings of our theoretical analysis in Chapter 7 in terms of bias-variance relationship of the ensembles can be used to guide the design of improved LtR algorithms. Our methods are able to identify whether bias or variance is the root cause of ranking error of a model, so when designing an LtR algorithm appropriate effort can be devoted to the specific error component. On this note, since the main goal of randomized tree ensembles is to reduce variance, future research may be devoted to reducing bias of these models for the LtR problem. Bias-corrected RF has already been studied in the literature for classification/regression [175] which could be adapted to the LtR task.

- Our findings in Chapter 6 discovered that performance of LtR algorithms may vary greatly from dataset to dataset. Characteristics of the datasets could be analyzed more thoroughly to investigate relative performances of different algorithms. For example, one might investigate how the accuracy of pointwise and listwise objective functions depend on the individual characteristics of the datasets. This thread of research is related to the comparatively less-explored area of measuring quality of the training sets [58, 112, 164].

- The problem of rank aggregation [139] has received much attention in the machine learning community. Despite the similarity between the two domains, an extensive comparison between LtR and rank aggregation methods looks missing in the literature. Techniques that apply the LtR algorithms on a rank-aggregation setting [99] could be a starting point for this line of research. As for the ordinal regression and collaborative filtering algorithms, although some early works on LtR were motivated by these problems (e.g., [49, 74]), we have not found any recent survey on the comparison between the LtR algorithms and these algorithms.

- Some relatively recently explored topics in LtR are online LtR [75] and semi-supervised LtR [147], whereas in random forests literature the online random forests [132] and semi-supervised random forests [32] are emerging techniques. Both of these techniques can be married together to yield effective online LtR with RF and semi-supervised LtR with RF.
Appendix A

Appendix to Chapter 2

A.1 Implementation Details of Evaluation Phase

Due to the presence of big datasets in our methodology, we design an efficient test-bed for the evaluation phase. A few details of this implementation are discussed below.

A.1.1 Storing the Models

The trees are printed in text files to get a twofold benefit. Firstly, for big datasets storing large number of trees in memory is not feasible. Secondly, once stored, the models can be reused. For every node of a tree, we store the feature index and the split-value in a single line. The structure of the stored file is flexible in the sense that in addition to storing the feature index and split-value, additional information may be recorded as needed. For evaluating an instance, we traverse through a (stored) tree (for a test instance) in $O(\log n)$ time.

A.1.2 Parallelism

We employ parallelism in both the training and evaluation phases as explained below.

Training Phase. According to the implementation of the Weka Library, each tree can be learnt in parallel depending on the available CPU cores of a machine.

Evaluation Phase. We implement parallelism in the evaluation phase as follows. The test set and a tree are first loaded into memory. The test set is then, according to the available CPU-cores, divided into a number of disjoint sets. Finally, each CPU-core
is assigned a task of extracting the score from the loaded tree and a (disjoint) subset of the test examples.
Appendix B

Appendix to Chapter 3

B.1 Oversampling Results

Table B.1: Performance comparison among RF-point (baseline), RF-point with oversampling (overall), and RF-point with oversampling (per query). An average over five independent runs is reported (and each run is the result of five-fold cross-validation), and the winning value is given in *italic* font.

<table>
<thead>
<tr>
<th>Metric</th>
<th>Baseline</th>
<th>Oversampling (overall)</th>
<th>Oversampling (per query)</th>
</tr>
</thead>
<tbody>
<tr>
<td>MQ2007</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.4360</td>
<td>0.4332</td>
<td>0.4377</td>
</tr>
<tr>
<td>MAP</td>
<td>0.4524</td>
<td>0.4518</td>
<td>0.4547</td>
</tr>
<tr>
<td>MQ2008</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.2234</td>
<td>0.2238</td>
<td>0.2271</td>
</tr>
<tr>
<td>MAP</td>
<td>0.4693</td>
<td>0.4728</td>
<td>0.4736</td>
</tr>
<tr>
<td>Ohsumed</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.4206</td>
<td>0.4096</td>
<td>0.4109</td>
</tr>
<tr>
<td>MAP</td>
<td>0.4213</td>
<td>0.4057</td>
<td>0.4057</td>
</tr>
<tr>
<td>TD2004</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.3513</td>
<td>0.3204</td>
<td>0.3263</td>
</tr>
<tr>
<td>MAP</td>
<td>0.2574</td>
<td>0.2349</td>
<td>0.2358</td>
</tr>
<tr>
<td>NP2004</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.7860</td>
<td>0.7071</td>
<td>0.7339</td>
</tr>
<tr>
<td>MAP</td>
<td>0.6647</td>
<td>0.5773</td>
<td>0.6038</td>
</tr>
<tr>
<td>HP2004</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>NDCG@10</td>
<td>0.8082</td>
<td>0.7904</td>
<td>0.8061</td>
</tr>
<tr>
<td>MAP</td>
<td>0.7174</td>
<td>0.6894</td>
<td>0.7016</td>
</tr>
</tbody>
</table>

Table B.1 shows the results of RF-point after applying an oversampling technique on the training set.
Appendix C

Appendix to Chapter 4

C.1 Variance Calculation of a Random Forest Using Correlation

The variance of a random forest at a datapoint $x_k$, $\sigma_{RF}^2$ that consists of $E$ trees is given by:

$$
\sigma_{RF}^2(x_k) = \rho_{RF}(x_k) \sigma_T^2(x_k) + \frac{(1 - \rho_{RF}(x_k)) \sigma_T^2(x_k)}{E},
$$

(C.1)

where $\sigma_T^2$ is the single tree variance, and $\rho_{RF}$ is the correlation between two trees at a datapoint $x_k$.

The above equation may be derived in different ways. Below we show our derivation.

Given $s_i(x_k)$ is the prediction of $i$th tree for instance $x_k$,

$$
\sigma_{RF}^2(x_k) \triangleq \text{Var} \left( \frac{1}{E} \sum_{i=1}^{E} s_i(x_k) \right)
$$

\begin{align*}
= & \frac{1}{E^2} \text{Var} \left( \sum_{i=1}^{E} s_i(x_k) \right) \\
= & \frac{1}{E^2} \left[ \sum_{i=1}^{E} \text{Var}(s_i(x_k)) + \sum_{i=1}^{E} \sum_{j=1, j \neq i}^{E} \text{Cov}(s_i(x_k), s_j(x_k)) \right]
\end{align*}
For ease of exposition we change the notation as $\text{Var}(s_i) = \sigma_i^2$. Assuming $\sigma_i^2(x_k) = \sigma_{k,j}^2(x_k)$, and $\text{Cov}(s_i(x_k), s_j(x_k)) = \text{Cov}(s_i(x_k), s_j(x_k)) = \rho(x_k)\sigma_i^2(x_k)$:

$$
\sigma_{RF}^2(x_k) = \frac{1}{E^2}[E\sigma_i^2(x_k) + E(E - 1)\rho(s_i(x_k), s_j(x_k))\sigma_i^2(x_k)]
$$

$$
= \frac{\sigma_i^2(x_k)}{E} + \rho(s_i(x_k), s_j(x_k))\sigma_i^2(x_k) - \frac{\rho(s_i(x_k), s_j(x_k))\sigma_i^2(x_k)}{E}
$$

$$
= \rho(s_i(x_k), s_j(x_k))\sigma_i^2(x_k) + \frac{\sigma_i^2(x_k)(1 - \rho(s_i(x_k), s_j(x_k)))}{E}.
$$
Appendix D

Appendix to Chapter 5

D.1 Results of Different Node Exploration Strategies

Table D.1: Results of RF-list-S5 with different node exploration strategies on MSLR-WEB10K (fold 1) dataset.

<table>
<thead>
<tr>
<th>Traversal Strategy</th>
<th>NDCG@10</th>
<th>MAP</th>
</tr>
</thead>
<tbody>
<tr>
<td>breadth-first</td>
<td>0.4225</td>
<td>0.3411</td>
</tr>
<tr>
<td>random-first</td>
<td>0.4243</td>
<td>0.3430</td>
</tr>
<tr>
<td>relevant-first</td>
<td>0.4238</td>
<td>0.3443</td>
</tr>
<tr>
<td>biggest-first</td>
<td>0.4230</td>
<td>0.3425</td>
</tr>
</tbody>
</table>

Table D.1 shows results of RF-list-S5 with different node exploration strategies on MSLR-WEB10K (fold 1) dataset. The approaches are: (1) breadth-first (breadth), (2) random-first (rand), (3) most relevant-first (rel) and (4) biggest-first (big). 5% data were used per tree.

D.2 Derivation of Time Complexities

Here we derive the training time complexities for the RF-point and RF-list.

Assuming that the recursive partitioning procedure results in a balanced tree of depth \( \log N \) (where \( N \) is the total number of training instances), then the cost of computing the tree can be written as:

\[
\text{cost}_\text{tree} = \sum_{h=0}^{\lfloor \log N \rfloor} 2^h \text{cost}_\text{node}(h) \quad \text{(D.1)}
\]

where \( \text{cost}_\text{node}(h) \) denotes the cost of computing a split-point for a node at depth \( h \) in the tree. (Note that in a balanced tree there will be \( 2^h \) nodes at depth \( h \).)
D.2.1 RF-point

For the pointwise algorithm, at each node a set of log \( M \) candidate features are investigated in order to determine the best split point. Each node at depth \( h \) will contain \( \frac{N}{2^h} \) data points on average. For each candidate feature this data will need to be both sorted (an \( O(n \log(n)) \) operation) and iterated over to determine the best split point (an \( O(n) \) operation – using running sums for computing entropies). Thus the cost per node can be calculated as follows:

\[
\text{cost}_{\text{node}}(h) = O(\log(M)(\frac{N}{2^h} \log(\frac{N}{2^h}) + \frac{N}{2^h})) = O(\log(M)\frac{N}{2^h} \log(\frac{N}{2^h})) \quad (D.2)
\]

And the total cost to build a tree in a pointwise manner is given by:

\[
\text{cost}_{\text{tree}} = \sum_{h=0}^{\lfloor \log N \rfloor} 2^h O(\log(M)\frac{N}{2^h} \log(\frac{N}{2^h})) = O(\sum_{h=0}^{\lfloor \log N \rfloor} \log(M)N \log(\frac{N}{2^h}))
\]

\[
= O(\log(M)N \log(2^h(N))) \quad (D.3)
\]

D.2.2 RF-list

For the listwise algorithm, we again, for each of the log \( M \) candidate features, need to sort and iterate over \( \frac{N}{2^h} \) data points at each node. Evaluating the objective function for each possible split-point, however, now requires iterating over all \( 2^h \) leaves of the tree and computing a DCG value for each of \( Q \) queries. If we use an appropriate data structure, where the leaves are maintained in sorted order of average relevance value (such that insertion/deletion is logarithmic in the number of leaves) and the relevance label distribution of documents for each query is maintained at each leaf node (such that Equation 5.4 can be used to update the DCG value of a query independently of its number of documents at the leaf), the cost of recomputing NDCG (across all queries) at each split point is given by:

\[
\text{cost}_{\text{NDCG}}(h) = O(2^h + 2^hQ) = O(2^hQ) \quad (D.4)
\]

Thus the cost to identify the best split point at each node becomes:
\[
\text{cost}_{\text{node}}(h) = \mathcal{O}(\log(M)\left(\frac{N}{2^h}\log\left(\frac{N}{2^h}\right) + \frac{N}{2^h}\text{cost}_{\text{NDCG}}(h)\right))
\]
\[
= \mathcal{O}(\log(M)\left(\frac{N}{2^h}\log\left(\frac{N}{2^h}\right) + NQ\right))
\]

(D.5)

And the total cost to build a tree in a listwise manner is given by:

\[
\text{cost}_{\text{tree}} = \sum_{h=0}^{[\log N]} 2^h \mathcal{O}(\log(M)\left(\frac{N}{2^h}\log\left(\frac{N}{2^h}\right) + NQ\right))
\]
\[
= \mathcal{O}(\log(M)N\left(\sum_{h=0}^{[\log N]} \log(N) - h + 2^hQ\right))
\]
\[
= \mathcal{O}(\log(M)N(\log^2(N) - \log(N) + Q2^{\log(N)}))
\]
\[
= \mathcal{O}(\log(M)N(\log^2(N) + QN))
\]

(D.6)

D.3 Implementation Details of RF-list

This section discusses a few subtleties of the implementation of RF-list algorithm.

D.3.1 Choice of Data Structures

The choice of data structure is crucial for building a tree of RF-list because here, unlike RF-point we need (1) to keep track the query-wise information, and (2) to maintain a global sorted list of leaves. The two top choices for manipulating a list of items in Java are arraylist and hashmap. There is a trade-off between the hashmap and arraylist. Hashmap is useful when the look-up operations are too frequent, otherwise arraylist is a natural choice. A problem with hashmap, however, is that it cannot be kept sorted, and to sort its items it needs to be converted into array-list which takes linear time. Treemap is another data structure that is kept sorted, but an additional logarithmic cost is incurred at the time of insertion. In our implementation we use hashmap and arraylist depending on the situation.

D.3.2 Sorting routine

Sorting is required in two places: (1) when selecting a split-value for a feature, the training examples need to be sorted based on the values of the feature in question, and
(2) For a potential split-point, when the NDCG values need to be (re-)computed based on the scores of the two new children nodes, the leaves of a tree need to be sorted based on their assigned scores. For the former of these two scenarios, the quicksort algorithm is used that takes $O(n \log n)$ time. As for the latter sorting task, once the leaves of a tree are sorted, the next split of a leaf in fact requires only (a) the deletion of that leaf from the (global) list of leaves and (b) the addition of the two new leaves in appropriate places of the sorted order. Hence we do not repeat the sorting of the entire list of leaves, instead we simply merge the previous sorted list of leaves (after extracting the leaf in question) with the new 2-item (sorted) list (corresponding to the two children). We thus reduce the $O(n \log n)$ time to $O(n)$. 
Appendix E

Appendix to Chapter 6

E.1 Implementations Used and Parameter Settings for Baseline Algorithms

Mart [92] is based on gradient boosted regression tree ensemble [52] and is a pointwise algorithm. We use a publicly available implementation in RankLib\(^1\), setting its parameters as follows: number of trees = 500, number of leaves for each tree = 7 (according to [69, Ch. 10], any value between 4 and 8 will work well). The rest of the parameters are kept unchanged.

RankSVM [83] is a pairwise linear LtR algorithm based on SVM, and has been used as a standard method for many years. We use the publicly available source code\(^2\).

RankBoost [49] is a pairwise algorithm which uses AdaBoost framework. Instead of using standard exponential loss, it uses a pairwise loss. We use the implementation in the RankLib package with the number of trees set to 500.

AdaRank [166] was discussed in Section 5.3. We use the implementation in RankLib with the number of base learners set to 500 and NDCG@10 as the optimization metric (during learning).

CoorAsc [105] was discussed in Section 5.3. We use the implementation in RankLib.

LambdaMart [159] was discussed in Section 5.3. A variation of this algorithm won the Yahoo LtR Challenge [25]. We use an open-source implementation of it mentioned in [55]\(^3\). The parameter settings are as follows: number of trees = 500, number of leaves for

\(^1\)https://people.cs.umass.edu/~vdang/ranklib.html
\(^3\)https://code.google.com/p/jforests/
each tree = 31 (Ganjisaffar et al. [56] report that value close to this has been found to be worked well for MSLR-WEB10K). On smaller datasets (including the learning curve experiments in Section 6.2.3.2), to mitigate overfitting we set it, like Mart, to 7. The rest of the parameters are kept unchanged. We note that during training, all the six baselines make use of validation set.


[35] Zhicheng Cui, Wenlin Chen, Yujie He, and Yixin Chen. Optimal action extraction for random forests and boosted trees.


[40] Wenkui Ding, Xiubo Geng, and Xu-Dong Zhang. Learning to rank from noisy data.


