FewPla: Efficient And Effective Limited Pass Learning For Large Data Quantities

Nayyar A. Zaidi, Geoffrey I. Webb, Francois Petitjean, Wray Buntine

Faculty of Information Technology, Monash University,
Clayton VIC 3800, Australia
{firtname.lastname}@monash.edu

Abstract. With ever increasing data quantities there is a pressing need for highly scalable learning algorithms. One important approach to this problem is to learn out-of-core – to learn incrementally without loading entire datasets into the memory. For this to be feasible for large quantities of data it is essential to learn in few passes through the data. Further, such learners should be low-biased – able to learn complex interactions among attributes. In this work, we have proposed a simple yet effective technique based on the already established three-pass selective k-dependence Bayesian classifier algorithm, by adding a functionality for discriminative learning of the parameters. This functionality can take an additional one to ten passes through the data. We show that the resulting algorithm – Fewpla: Few Pass Learning Algorithm – cannot only improve the performance of SKDB but is extremely competitive with state of the art in in-core machine learning method Random Forest.

1 Introduction

The staggering pace in the growth of training data has led to a renewed interest in efficient and effective learning algorithms. Out-of-core algorithms provide one avenue of attack on the problem of learning from data that is too large to fit in available memory. To learn out-of-core efficiently it is essential to learn in a minimum number of passes through the data. An effective algorithm, should be inherently low biased, so that the trained classifier can benefit from the fine grained detail implicit in very large datasets. Additionally, it is desirable that a learning algorithm should have minimal tuning parameters, as the cost of repeatedly learning with different parameter values is prohibitive when learning from massive data quantities. Preferably, any tuning parameters should have the minimal impact on the computational efficiency and classification effectiveness of the model.

Logistic Regression is a linear classifier that can be trained very quickly out-of-core with stochastic gradient descent-based optimization, but cannot take into account higher-order interactions in the data – hence is highly-biased. The reason for the high-bias of logistic regression is because the model structure is shallow. One can reduce its bias by taking into account higher-order interactions, e.g., quadratic, cubic or higher [1, 2]. This, however, increases the size of the model

drastically. Another option is to build a deeper model such as an 'Artificial Neural Network' [3]. The deep learning paradigm has been shown to achieve excellent results on many structured-datasets[1]. However, on non-structured datasets, application of deep learning opens the question of choosing the depth and the breadth of the model, as well as devising an effective back-propagation strategy and choosing activation functions. All of these factors (tuning parameters) greatly affect the convergence of the algorithm hence influencing both the efficiency and the effectiveness of the model.

Bayesian Network Classifiers (BNC) can also learn models that incorporate relevant high-order interactions between variables, and (for a certain class of Bayesian Network Classifiers) can do so out-of-core in few passes through the data without requiring many tunable parameters. In recent work we have shown how discriminative parameterization of such models can efficiently learn very accurate classifiers [4]. However, this discriminative parameterization has required in-core optimization of the parameters relative to the data.

In this paper we address the problem of learning these discriminative parameters out-of-core in few passes through the data.

- We present an effective low-biased, computationally efficient algorithm for learning from large quantities of data in four or more passes through the data. Most state of the art work – for scalable learning – in the direction of Bayesian Networks are constrained to linear models such as naive Bayes. For example, in [5], a simple logistic regression model is proposed with adaptive step size and adaptive regularization. Our work is the first that investigates higher-order BNC models for achieving low-biased models in a minimum number of passes through the data. Additionally, we propose a simple algorithm for tuning the initial step size for adaptive gradients. We also derive a form of adaptive regularization for BNC where the discriminatively trained parameters are regularized back towards MLE estimated parameters.
- We show that the accuracy of the models learned by the proposed approach is competitive to the state of the art in in-core learning.
- We present an optimized code library for learning from large quantities of data.

The rest of this paper is organized as: Section 2 presents techniques which forms the basis of our proposed approach such as Bayesian Network classifiers, KDB and selective KDB. We present the details of our proposed algorithm in Section 3. The experimental analysis is done in Section 4. We conclude in Section 5 with pointers to future work. Appendix A has the details of the code library.

2 Bayesian Network Classifiers

A BN $\mathcal{B} = \langle \mathcal{G}, \Theta \rangle$, is characterized by the structure $\mathcal{G}$ (a directed acyclic graph, where each vertex is a variable, Z_i), and a set of parameters Θ, that quantifies

[1] Note, this should not be confused with structured prediction (or structure learning) in machine learning. By structured datasets, we mean any datasets where there is an apparent structure present among features – e.g., image, speech, text, etc.

the dependencies within the structure. The variables are partitioned into a single target, the class variable $Y{=}Z_0$ and n covariates $X_1{=}Z_1, X_2{=}Z_2, \ldots X_n{=}Z_n$, called the *attributes*. The parameter Θ, contains a set of parameters for each vertex in $\mathcal{G}$: for target ($i = 0$), $\theta_{z_0|\Pi_0(\mathbf{x})}$ and for $1 \leq i \leq n$, $\theta_{z_i|\Pi_i(\mathbf{x})}$, where $\Pi_i(.)$ is a function which given the datum $\mathbf{x} = \langle x_1, x_1, \ldots, x_n \rangle$ as its input, returns the values of the attributes that are the parents of node i in structure $\mathcal{G}$. For notational simplicity, instead of writing $\theta_{Z_0=z_0|\Pi_0(\mathbf{x})}$ and $\theta_{Z_i=z_i|\Pi_i(\mathbf{x})}$, we write $\theta_{z_0|\Pi_0(\mathbf{x})}$ and $\theta_{z_i|\Pi_i(\mathbf{x})}$. Typically, for Bayesian network classifiers, the target node is considered to be the parent of every other node and also, the target node does not have any parents. Therefore, the joint probability distribution defined by BN for data point $\mathbf{x}$ with label y can be written as:

$$\mathrm{P}_{\mathcal{B}}(y, \mathbf{x}) = \theta_y \prod_{i=1}^{n} \theta_{x_i|y,\Pi_i(\mathbf{x})}. \tag{1}$$

The corresponding conditional distribution $\mathrm{P}_{\mathcal{B}}(y|\mathbf{x})$ can be computed with the Bayes rule as:

$$\mathrm{P}_{\mathcal{B}}(y|\mathbf{x}) = \frac{\mathrm{P}_{\mathcal{B}}(y, \mathbf{x})}{\mathrm{P}_{\mathcal{B}}(\mathbf{x})} = \frac{\theta_y \prod_{i=1}^{n} \theta_{x_i|y,\Pi_i(\mathbf{x})}}{\sum_{y' \in \mathcal{Y}} \theta_{y'} \prod_{i'=1}^{n} \theta_{x_{i'}|y',\Pi_{i'}(\mathbf{x})}}. \tag{2}$$

Given a set of data points $\mathcal{D} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)}\}$, a standard objective function to optimize is the Log-Likelihood (LL) of the data – that is:

$$\begin{aligned} \mathrm{LL}(\mathcal{B}) &= \sum_{j=1}^{N} \log \mathrm{P}_{\mathcal{B}}(y^{(j)}, \mathbf{x}^{(j)}), \\ &= \sum_{j=1}^{N} \left(\log \theta_{y^{(j)}} + \sum_{i=1}^{n} \log \theta_{x_i^{(j)}|\Pi_i(\mathbf{x}^{(j)})} \right), \end{aligned} \tag{3}$$

with constraints: with $\sum_{y \in \mathcal{Y}} \theta_y = 1$, and $\sum_{x_i \in \mathcal{X}_i} \theta_{x_i|\Pi_i(\mathbf{x})} = 1$. Maximizing Equation 3 to optimize the parameters (θ) is the maximum-likelihood estimation of the parameters. Another option is to directly optimize for $\mathrm{P}_{\mathcal{B}}(y|\mathbf{x})$ by maximizing the Conditional Log-Likelihood (CLL). Optimizing CLL is generally considered a more effective objective function (for classification) since it directly optimizes the mapping from features to class labels. The CLL can be defined as:

$$\begin{aligned} \mathrm{CLL}(\mathcal{B}) &= \sum_{j=1}^{N} \log \mathrm{P}_{\mathcal{B}}(y^{(j)}|\mathbf{x}^{(j)}), \\ &= \sum_{j=1}^{N} \left(\log \mathrm{P}_{\mathcal{B}}(y^{(j)}, \mathbf{x}^{(j)}) - \log \sum_{y'}^{|\mathcal{Y}|} \mathrm{P}_{\mathcal{B}}(y', \mathbf{x}^{(j)}) \right), \\ &= \sum_{j=1}^{N} \left(\log \theta_{y^{(j)}} + \sum_{i=1}^{n} \log \theta_{x_i^{(j)}|\Pi_i(\mathbf{x}^{(j)})} \right) - \log \left(\sum_{y'}^{|\mathcal{Y}|} \theta_{y'} \prod_{i'=1}^{n} \theta_{x_{i'}|y',\Pi_{i'}(\mathbf{x}^{(j)})} \right). \end{aligned} \tag{4}$$

Note that, the optimization assumes that the structure $\mathcal{G}$ is given. Therefore, the objective function in Equations 3 and 4 can be written as: $\mathrm{LL}(\Theta)$ and $\mathrm{CLL}(\Theta)$ respectively.

2.1 Discriminative Bayesian Network Classifier

One can combine both LL and CLL objective functions with a weighted BN formulation [4]. A weighted BN has an additional set of parameters – that is, $\mathcal{B} = \langle \mathcal{G}, \Theta, \mathbf{w} \rangle$. For every parameter θ, there is an additional parameter w. We can define the joint probability as: $\mathrm{P}_{\mathcal{B}}(y, \mathbf{x}) = \theta_y^{w_y} \prod_{i=1}^{n} \theta_{x_i|y,\Pi_i(\mathbf{x})}^{w_{x_i,y,\Pi_i(\mathbf{x})}}$. For weighted BN, several optimization options were discussed in [4] to manage Θ and $\mathbf{w}$. We will only be interested in the 'weighted' parametrization (as it has been shown to converge the fastest) which is a two step process. Step one fixes the values of the $\mathbf{w}$ parameters (to 1) and learns Θ by optimizing the LL objective function. Step two treats Θ as a constant scale and optimizes for $\mathbf{w}$ by optimizing a CLL objective function. The scale is critical in speeding up the optimization and in regularisation. The modified CLL function for data point $\mathbf{x}$ with label y takes the form:

$$\log \mathrm{P}_{\mathcal{B}}(y|\mathbf{x}) = (w_y \log \theta_y + \sum_{i=1}^{n} w_{y,x_i,\Pi_i} \log \theta_{x_i|y,\Pi_i(\mathbf{x})}) - \log \sum_{y'}^{|\mathcal{Y}|} (\theta_{y'}^{w_y} \prod_{i'=1}^{n} \theta_{x_{i'}|y',\Pi_{i'}(\mathbf{x})}^{w_{y,x_{i'},\Pi_i}}). \quad (5)$$

Note, the question of determining the parents of each node, i.e, learning function $\Pi_i(\mathbf{x})$ remains open. As discussed in Section 1, there is a need to learn this function as efficiently as possible.

2.2 k-dependence Bayesian classifier

One can avoid learning the structure of a BN by assuming: $\Pi_i(\mathbf{x}) \rightarrow \{\}$. This leads to the naive Bayes (NB) classifier. NB is the simplest of the BNCs, assuming that all attributes are independent given the class. It estimates the joint probability using: $\log_{\mathcal{B}} \mathrm{P}(y, \mathbf{x}) = \log \theta_y + \sum_{i=1}^{n} \log \theta_{x_i|y}$.

TAN (tree-augmented naive Bayes) is a structural augmentation of NB where every attribute has as parents the class and at most one other attribute. The structure is determined by using an extension of the Chow-Liu tree [6], that utilizes conditional mutual information to find a maximum spanning tree. This alleviates some of NB's independence assumption and therefore reduces its bias at the expense of increasing its variance. This results in better classification performance on larger data sets. TAN, provides an intermediate bias-variance trade-off, standing between NB on one hand and unrestricted BNCs on the other.

KDB relaxes NB's independence assumption by allowing every attribute to be conditioned on the class and, at most, k other attributes. Like TAN, KDB requires two passes over the training data for learning: the first pass collects the statistics to perform calculations of mutual information for structure learning, and the second performs the parameter learning based on the structure created in the former step. The structure learning encompasses:

- Compute $\mathrm{MI}(X_i;Y)$ for all attributes and $\mathrm{MI}(X_i,X_j;Y)$ for all pairs of attributes and order all attributes based on MI (high to low) in a list $\mathcal{L}$
- For $i=0$ to size($\mathcal{L}$)
  - Select attribute X_i and intialize $\mathcal{L}_i$ as list of attribute i parents. Add Y to $\mathcal{L}_i$ and set $k' = \min(i-1,k)$
  - While $counter > 0$
    - $X_j = \mathrm{argmax}_j\{\mathrm{MI}(L_i,L_j|Y); \forall\ 1 \leq j < i \text{ and } X_j \notin \mathcal{L}_i\}$
    - Add X_j to $\mathcal{L}_i$
    - $counter = counter - 1$
  - Sort $\mathcal{L}'$ (high to low)
  - Add k' elements from $\mathcal{L}'$ as parent of attribute X_i

2.3 Selective k-dependence Bayesian classifier

The parameter k controls KDB's bias-variance trade-off. Higher k results in higher variance and lower bias. Unfortunately there is no apriori means to preselect a value of k for which this trade-off will minimize error for a given training set as this is a complex interplay between the data quantity and the complexity and strength of the interactions between the attributes. A further factor that can increase KDB's error is that attributes that carry no useful information about the class must be treated as if they do, which invariably introduces some noise into the estimates. Discarding these attributes can both reduce error and classification time. Selective KDB (SKDB), extends KDB to select between attribute subsets and values of k in a single additional pass through the training data.

In the general case, attribute selection is a complex combinatorial task. For d attributes there are 2^d alternative attribute subsets that could be explored. Many attribute selection algorithms utilize either forward selection or backwards elimination hill-climbing strategies [7, 8]. SKDB seeks to perform both attribute and best-k selection in a single pass. For attribute selection, it takes advantage of the order already selected by KDB based on mutual information with the class. Given that the attributes are sorted on this order, it consider only the attribute sets $\{x_1,\ldots x_i\}, 1 \leq i \leq d$, that is attribute sets that each contain the i attributes that have the greatest mutual information with the class. These d attribute subsets are evaluated simultaneously in a single pass through the data using leave-one-out cross validation (LOOCV). It is possible to efficiently evaluate all d subsets simultaneously, because a KDB classifier using subset $\{x_1,\ldots x_i\}$ is a minor addition to a KDB classifier using subset $\{x_1,\ldots x_{i-1}\}$. Calculating a KDB classifier for all attributes already incorporates all the calculations for all the KDB classifiers using subsets that we consider. Similarly for each k, evaluation of the model that allows each attribute to have k parents subsumes the calculations required to evaluate the model allowing $k-1$ parents.

SKDB uses an incremental cross validation [9], which performs LOOCV on BNCs very efficiently. A naive approach to LOOCV will for each holdout example recalculate from scratch all of the joint frequency counts required by a BNC. Incremental cross-validation first gathers the counts for all training examples in a single pass through the data. Then, when classifying a holdout example, its

counts are removed from the table. SKDB collects the complete counts in its second pass through the training data and performs LOOCV in a third pass. An outline of SKDB process is given in Algorithm 1. It can be seen that in just a single extra pass through the data, SKDB selects the best number of attributes and the right value of k for a standard KDB algorithm.

Algorithm 1: Selective-KDB

Input: Data ($\mathcal{D}$), k^{max}
Output: Θ,b
Let $\mathcal{L}$ be the list of all attributes sorted (high to low) based on MI
Let $\mathcal{P}$ be a $(k+1) \times |\mathcal{Y}|$ matrix of posterior probabilities and $\mathcal{E}$ be a $(k \times n)$ matrix of LOOCV results. Note $k = 0$ is the case of naive Bayes
Let us denote by $\Theta^{\downarrow \mathbf{x}}$ a BN with data point $\mathbf{x}$ discounted from its CPT
for $i = 1 \rightarrow$ *size($\mathcal{D}$)* **do**
  Pick i'th data point: $\mathbf{x}$
  for $k' = 0 \rightarrow k^{max}$ **do**
    $\mathcal{P}[k'][y^*] = \mathrm{P}_{\Theta^{\downarrow \mathbf{x}}}(y^*|\mathbf{x})$, $\forall y^* \in |\mathcal{Y}|$
    for $l = 0 \rightarrow size(\mathcal{L})$ **do**
      $X_j = \mathcal{L}$.nextelement
      $\mathcal{P}[k'][y^*] = \mathcal{P}[k'][y^*] \times \mathrm{P}_{\Theta^{\downarrow \mathbf{x}}}(x_j|y^*, \Pi_j^{k'}(\mathbf{x}))$, $\forall y^* \in |\mathcal{Y}|$, where $\Pi_j^{k'}(\mathbf{x}))$ denotes the k' first parent-values of X_j in $\mathbf{x}$
      $\mathcal{E}[k'][l] = \mathcal{E}[k'][l] + \mathrm{error}(\mathcal{P}[k'], y^{\mathbf{x}})$. Here $\mathcal{P}[k']$ is the vector of posterior-probabilities only considering the top l attributes. Error is the Root-Mean-Square error
    end
  end
end
Select b and k indexes with the best $\mathcal{E}$
return *b,k*

3 FewPLA

Our proposed FewPLA algorithm learns a discriminative Bayesian network classifier using selective KDB to learn the structure. An outline is given in Algorithm 2. It can be seen that the first three passes over the data are the same as the SKDB algorithm. Additionally it randomly selects K data points – this is a small subset of the data and can be easily stored in the memory. We call this dataset $\mathcal{D}^V$ – the validation set. It will be used in later passes for tuning step size and the regularization parameters. Based on the output of SKDB, the parameter Θ is truncated (based on the number of attributes and k selected) – and $\mathbf{w}$ is initialized which is of identical size to Θ. This is one of the great advantage of using SKDB, as in most cases a smaller k is generally chosen and a small subset of attributes will not be selected. This greatly reduces the size of parameter space spanned by Θ and $\mathbf{w}$ – leading to faster training time in the subsequent discriminative passes.

Discriminative learning starts by learning the step size of the SGD optimization. Before explaining the algorithm for choosing the step size, let us explain

Algorithm 2: The FewPLA algorithm

Input: Training set $\mathcal{D}$ with attributes $\{X_1, \ldots, X_n, Y\}$
Input: k^{max} (default = 2), I (default = 1)
Output: FewPLA model parameterized by Θ and $\mathbf{w}$

$\mathcal{G}$ =learnStructureKDB($\mathcal{T}$) # Pass 1
Θ = learnParameters($\mathcal{T}$, $\mathcal{G}$) # Pass 2
b, k = selectiveKDB($\mathcal{T}$, $\mathcal{G}$, Θ) # Pass 3
Sample K data points in the last pass and store them in memory – $\mathcal{D}^V$
Truncate Θ to attribute subset $\{1 \ldots b\}$ and set maximum no. of parents to k
Book Keeping
Allocate $\mathbf{w}$ and initialize it to 1
Determine η_0 through cross-validation # Algorithm: 3
Intialize $t = 0$
for $iter = 4 \rightarrow I$ **do**
  # Pass 'iter'
  for $i = 1 \rightarrow size(\mathcal{D})$ **do**
    Pick i'th data point: $\mathbf{x}^t$
    for $j = 1 \rightarrow n$ **do**
      $\eta_j = \frac{\eta_0}{\sqrt{G_j * G_j}}$
      $g_j = \frac{\partial f(\mathbf{w}^t)}{w_j^t} + \lambda^t (w_j^t - 1)$
      $w_j^{t+1} = w_j^t + \eta_j g_j$
      $G_j = G_j + g_j$
    end
    t = t + 1
  end
  Draw a sample from $\mathcal{D}^V$ and update: $\lambda^{t+1} = \lambda^t + \alpha \sum_j^n g_j (w_j^t - 1)$
end
return Θ, $\mathbf{w}$

the objective function that FewPLA optimizes. We will deviate from our previous notation of superscript to denote a data point j as $\mathbf{x}^{(j)}$ in dataset $\mathcal{D}$, to $\mathbf{x}^t$, which can be interpreted as data point at time t. FewPLA passes over the data I times – each pass consist of looping over all the data points in the data. An iteration t encompasses selecting a data point $\mathbf{x}^t$ and optimizing the following objective function where $\mathbf{w}^t$ denotes the set of parameters at iteration t. This adds a regularisation term to $l^t(\mathbf{w}) = \log \mathrm{P}_{\mathcal{B}}(y^t | \mathbf{x}^t; \mathbf{w}^t)$ (defined in Equation 5):

$$f(\mathbf{w}^t) = l(\mathbf{w}^t) + \frac{\lambda^t}{2} \|\mathbf{w}^t - \mathbf{1}\|^2. \tag{6}$$

The goal of the regularization is to regularize the values towards the MLE parameters. Of course, since we are using weighted objective function of the form of Equation 5, we can achieve this objective by regularizing $\mathbf{w}$ towards 1. At

each step of the optimization, FewPLA makes the following update:

$$\mathbf{w}^{t+1} = \mathbf{w}^t + \eta \frac{\partial f(\mathbf{w}^t)}{\partial \mathbf{w}^t}, \tag{7}$$

where η is the step size and $\frac{\partial f(\mathbf{w}^t)}{\partial \mathbf{w}^t}$ is defined for class parameters as:

$$\frac{\partial \log \mathrm{P}_{\mathcal{B}}(y^t|\mathbf{x}^t;\mathbf{w}^t)}{\partial w^t_{y:k}} = \left(\mathbf{1}_{y^t=k} - \mathrm{P}(k|\mathbf{x}^t)\right) \log \theta_y + \lambda^t (w^t_y - 1), \tag{8}$$

and for non-class parameters as:

$$\frac{\partial \log \mathrm{P}_{\mathcal{B}}(y^t|\mathbf{x}^t;\mathbf{w}^t)}{\partial w^t_{y:k,x_i:j,\Pi_i(\mathbf{x}):l}} = \left(\mathbf{1}_{y^t=k} - \mathrm{P}(k|\mathbf{x}^t)\right) \mathbf{1}_{x_i=j} \mathbf{1}_{\Pi_i(\mathbf{x})=l} \log \theta_{x_i|y,\Pi_i(\mathbf{x})} + \lambda^t (w^t_{y,x_i,\Pi_i(\mathbf{x})} - 1). \tag{9}$$

Here $w_{y:k}$ denotes the parameter associated with the class variable taking value k and $w_{y:k,x_i:j,\Pi_i:l}$ denotes the parameter associated with the class variable taking value k, attribute x_i taking value j and parents $\Pi_i(\mathbf{x})$ taking values l. The partial derivates from Equations 8 and 9 can be fed in Equation 7 to find an updated parameter, however, there are two issues. There are two parameters: η – the step size and λ – the regularization parameter that greatly control the performance of any limited pass learning system. In the following, we will describe how FewPLA handles these two hyper-parameters.

Choosing the right regularization parameter can be computationally demanding[2]. The regularization algorithm that FewPLA employs has two distinctive properties. First, it regularizes the parameters $\mathbf{w}$ back towards 1, i.e., the penalty term added is: $\frac{\lambda^t}{2}\|\mathbf{w}^t - \mathbf{1}\|^2$. Second, it uses 'adaptive regularization' [10], i.e., the regularization parameter is also updated at every iteration t. We will show in the following how adaptive regularization leverages the validation data ($\mathcal{D}^V$) to choose a locally optimal regularization parameter.

It can be seen that Equation 6 is actually a function of three sets of parameters: $\mathbf{w}, \Theta, \lambda$. Since we have fixed Θ, we can ignore them – but, our problem is that of learning both $\mathbf{w}$ and λ. However, learning $\mathbf{w}$ and λ together is not trivial. Fixing λ, we have a solution to find the optimal $\mathbf{w}$ based on solving for Equation 6. However, naively fixing $\mathbf{w}$ and optimizing for λ is not a realistic task: λ shoots off to infinity. So to alternate between optimizing $\mathbf{w}$ and λ, we need to define the update of $\mathbf{w}$ in terms of λ.

The idea behind adaptive regularization is to introduce the λ parameter in the optimization by using a simple trick. Writing explicitly, for a parameter $\mathbf{w}$, we make the following update: $\mathbf{w}^{t+1} = \mathbf{w}^t + \eta(\frac{\partial f(\mathbf{w}^t)}{\partial \mathbf{w}^t} + \lambda(\mathbf{w}^t - \mathbf{1}))$. We can slightly modify our optimization problem so now λ appears explicitly.

$$f(\lambda^*; \mathbf{w}^{t+1}) = \sum_{\mathbf{x} \in \mathcal{D}^V} l(\mathbf{w}^{t+1}). \tag{10}$$

[2] The standard approach is to train multiple models with different values of regularization parameter λ on the data and test the accuracy over a disjoint validation data set. However, this approach is infeasible for our limited pass learning algorithm.

Note, $l(\mathbf{w}^{t+1})$ is $\log \mathrm{P}_{\mathcal{B}}(y|\mathbf{x};\mathbf{w}^{t+1})$ and is equal to:

$$\log \mathrm{P}_{\mathcal{B}}(y|\mathbf{x};\mathbf{w}^{t+1}) = (\Omega_y \log \theta_y + \sum_{i=1}^{n} \Omega_{y,x_i,\Pi_i(\mathbf{x})} \log \theta_{x_i|y,\Pi_i(\mathbf{x})}) - \log \sum_{y'}^{|\mathcal{Y}|} \exp\left(\Omega_y \log \theta_{y'} + \sum_{i'=1}^{n} \Omega_{y,x_{i'},\Pi_{i'}(\mathbf{x})} \log \theta_{x_{i'}|y',\Pi_{i'}(\mathbf{x})}\right),$$

where

$$\Omega_y = w_y^t - \eta(\frac{\partial f(\mathbf{w})}{\partial w_y^t} + \lambda^t(w_y^t - 1)),$$
$$\Omega_{y,x_i,\Pi_i(\mathbf{x})} = w_y^t - \eta(\frac{\partial f(\mathbf{w})}{\partial w_{y,x_i,\Pi_i(\mathbf{x})}^t} + \lambda^t(w_{y,x_i,\Pi_i(\mathbf{x})}^t - 1).$$

The SGD update for λ can simply be carried as:

$$\lambda^{t+1} = \lambda^t + \alpha \frac{\partial f(\lambda^*;\mathbf{w}^{t+1})}{\partial \lambda}, \tag{11}$$

which is actually an inner optimization problem (for λ) with-in the outer main optimization procedure (for optimizing $\mathbf{w}$). Again, there is a parameter α that characterizes the step size of this inner SGD. Taking the derivative of Equation 10 with-respect-to λ reveals:

$$\frac{\partial f(\lambda^*;\mathbf{w}^{t+1})}{\partial \lambda} = \sum_{j}^{n} g_j(w_j - 1), \tag{12}$$

where g_i is equal to: $\frac{\partial l^t(\mathbf{w})}{\partial \mathbf{w}_i}$ – that is the partial derivate w.r.t to dimension i. This leads to the adaptive regularization used by FewPLA in Algorithm 2 – $\lambda^t + \alpha \sum_j^n g_j(w_j - 1)$.

Let us discuss the issue of step size. The performance of any SGD algorithm is extremely sensitive to step-size: step size too large results in divergence and a step size too small results in slower convergence. It is vital for FewPLA to choose a step size that leads to convergence in few iterations. A number of options were tested including a fixed learning rate, AdaGrad [11], AdaDelta [12], NoPeskyLearning [13] and adaptive gradients of the form $\frac{\eta_0}{1+\lambda t}$ [14]. Note, all of these methods except NoPeskyLearning have tuneable parameters that get tuned using cross-validation. FewPLA is based on AdaGrad [3] as we found it to be working the best – provided we choose η_0 properly. Note, NoPeskyLearning rate did not result in good performance. We attribute this to the sparseness of the Hessian for high-order models such as selective KDB.

[3] AdaGrad is an adaptive step size algorithm, i.e, at iteration t, it makes the following update in dimension j: $\frac{\eta_0}{\sqrt{G_j * G_j}}$, where G_j is the accumulated gradient in dimension j and η_0 is the initial step size.

For AdaGrad, FewPLA learns the value of η_0 by doing cross-validation. For this it saves K data points in the memory and finds the best value of η_0 through an approach described in Algorithm 3. The algorithm starts with a large interval 10^6 and 10^{-6} and evaluates the performance with different values with-in the interval. It finds the best interval and applies the algorithm recursively to find an optimal value of the step size until the difference in the performance is less than ϵ. An outline of the evaluateLineSearch() function is given in Algorithm 4. The EvaluateFunction() does a 90:10 split of the validation dataset – $\mathcal{D}^V$. It learns a classifier on the 90% of the data with step size of $10^{\alpha[i]}$ and test on the remaining 10%. The RMSE performance is returned.

Algorithm 3: Line-search

Input: Validation set $\mathcal{D}^V$, $\epsilon = 0.01, High = 6, Low = -6, Scale = 10$
Output: η_0
while $(|P_A - P_B| > \epsilon)$ **do**
  $P_A, P_B, high, low =$ evaluateLineSearch$(High, Low, Scale)$ # Algorithm 4
  $High \leftarrow high$, $Low \leftarrow low$
end
return $\eta_0 = (Scale^{High} + Scale^{Low})/2$

Algorithm 4: evaluateLineSearch

Input: Validation set $\mathcal{D}^V$, int $High$, int Low, int $Scale$
Output: $P_A, P_B, \alpha_A, \alpha_B$
Intitialize α and $perf$ to be a vector of size $(Scale + 1)$
$\rho = \frac{High - Low}{Scale}$
$\alpha[0] \leftarrow Low$, $\alpha[Scale] \leftarrow High$
for $i = 1 \rightarrow size(\alpha)$ **do**
  $\alpha[i] \leftarrow \alpha[i-1] + \rho$
  perf[i] $\leftarrow$ EvaluateFunction($\mathcal{D}^V$, $Scale^{\alpha[i]}$)
end
$j \leftarrow$ smallestInVector$(perf)$
return $perf[j-1], perf[j+1], \alpha[j-1], \alpha[j+1]$

3.1 Tunable Parameters

FewPLA has two input tunable parameters – k^{max} and I. The performance of FewPLA does depend directly on these two parameters. Note, for bigger datasets, a big k^{max} is to be expected. However, a larger value of k^{max} will result in a longer third pass and hence will increase the training time. On the other hand, a lower value of k^{max} can reduce classification accuracy. The number of iterations I can also affect the performance. Typically just one iteration can lead to a good result, but on some datasets more iterations are desirable. We recommend $k^{max} = 5$ and $I = 10$ as a parameter setting that our experience suggests works well across many datasets.

There are a few other parameters that can affect (though not drastically) the performance of FewPLA – e.g., size of validation set ($\mathcal{D}^V$) – K in Algorithm 2, ϵ and $Scale$ in Algorithm 3, α – the step size of inner-optimization as shown in Equation 11 and the initial value of λ. The only significant parameter of these is the α parameter. We have set it to 10^{-3} in our experiments. The algorithm is quite robust to this parameter, but it can affect the convergence and hence the results in some cases.

4 Experiments

In this section, we will evaluate the performance of FewPLA on 14 standard benchmark datasets from UCI repository [15]. The details of datasets are shown in Table 1 – only datasets with more than 100 000 data points are selected in this study. All datasets except `Satellite` are from the UCI repository and in the public domain. The details about the `Satellite` dataset can be found in [16]. All

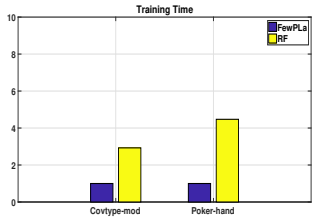

Fig. 1. Training time comparison of FewPLA and RF on two representative datasets. Results are normalized w.r.t FewPLA.

Domain	Case	Att	Class
Satellite	8705159	138	24
MNIST	8100000	784	10
KDDcup	5209460	41	40
Poker-hand	164860	10	10
Covtype	581012	43	7
UPSExtended	341462	676	2
Census-income	299285	41	2
SkinSegmentation	245057	3	2
WearableComputing	165632	18	5
Localization	164860	4	11
TwitterAbsolutSigma	140607	77	2
MiniBooNE_PID	130065	48	2
Diabetes	101766	46	2
Waveform	100000	18	3

Table 1. Details of Datasets (listed in descending order of their sizes). Biggest three datasets are shown in bold.

these datasets have a mixture of both quantitative and qualitative attributes. We discretized the quantitative attributes using the Minimum Description Length (MDL) discretization method [17]. Also a missing value is treated as a separate attribute value and taken into account exactly like other values.

We include two benchmark classifiers – naive Bayes and Random Forest. The two classifiers can be seen as being on two extremes. Naive Bayes is a single pass learning algorithm, but is highly biased. Random Forest on the other hand is an in-core learner but is very low-biased and a state-of-the-art classifier. All experiments are computed on a standard desktop with 64GB of RAM. The number of trees in Random Forest ensemble are set to 100. If an out-of-memory (OOM) exception is raised, we resort to 50 trees and if the exception due to OOM occurs again, we resort to 10 trees.

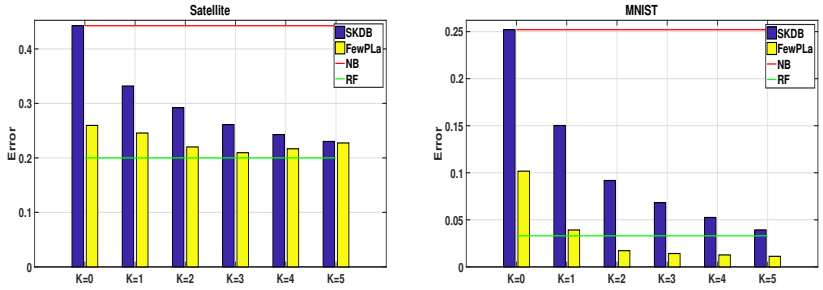

Fig. 2. Comparison of FewPLA 0-1 Loss performance with SKDB, naive Bayes and Random Forest on the two biggests datasets.

FewPLA is compared against naive Bayes and Random Forest in terms of the error (0-1 Loss) in Figures 2 and 3. Note, all RF results are with 100 trees, except on `satellite` and `MNIST`, where 10 trees are used as 50 trees led to out-of-memory. Two rounds of two-fold cross-validation is used. For sake of completion, we also report the SKDB results. The bar graphs represent FewPLA error with different values of k^{max}. The number of discriminative iterations are set to 10 for all datasets. All in all, FewPLA results are with 14 passes through the data – one pass for discretization, three for SKDB and ten for discriminative training of parameters.

Note that, given k^{max}, SKDB chooses the best value of k for the Bayesian network with the generative parameterization in effect during the third pass. For example – let us call the best value of k to be k^* and in most cases $k^* < k^{max}$. In the plots, for FewPLA and SKDB, results are only shown up to k^*. Note that NB and RF are not a function K, so in Figure 3, we have shown their performance as straight lines providing a benchmark for the relative performances of FewPLA with varying k.

4.1 FewPLA vs. SKDB

It can be seen from Figures 2 and 3, in almost all cases, FewPLA results in better performance than SKDB. The exceptions are `poker-hand` with $k = 3$ and $k = 4$ and `Covtype-mod` with $k = 4$. We attribute this to over-fitting of the discriminative parameters, as discussed below in Section 4.4.

4.2 FewPLA vs. NB and RF

FewPLA results in much better performance than NB on all datasets. Compared against RF, it can be seen that, on seven out of fourteen datasets, FewPLA with some k results in better performance than Random Forest. Note, FewPLA in our experiments is only a fourteen pass out-of-core learner – that its classification accuracy is very close to RF is extremely encouraging. Comparison of the training time on two representative datasets in shown in Figure 1. For both datasets,

training time is of FewPLA with $k = 4$. It can be seen that it is an order of magnitude faster than standard RF with 100 trees.

4.3 FewPLA vs. In-core KDB

To get an idea of the quality of parameters obtained with optimization in FewPLA, on a small subset of our datasets for which it was feasible to do so, we ran the in-core quasi-Newton algorithm in place of the out-of-core SGD to optimize the discriminative parameters. The results are shown in Figure 4. Except on `Pokerhand` dataset with $K = 2$, it can be seen that the two models actually lead to similar performance. We attribute the cases in which SGD achieves lower error than quasi-Newton to overfitting from the latter.

4.4 FewPLA – Limitations

SKDB choses the best k. As it is difficult to overfit such a parameter that can assume only one of six discrete values, performance at k will usually be no worse than that at $k-1$. This can be seen in Figure 3, where the blue bars representing SKDB error decrease with increasing values of K.

However, this is not always the case for FewPLA. It can be seen that on many datasets, e.g., `Satellite`, `Poker-hand`, `Covtype-mod`, `TwitterAbsoluteSigma`, `Diabetes` and `Waveform`, FewPLA with a lower value of k results in better performance than with a higher k. The yellow bars representing FewPLA error do get higher with increasing values of K. We believe that this is due to over-fitting – the discriminative parameterization much more closely fits the data. This is why it usually decreases error. But it also leaves itself open to overfitting. This can be overcome by replacing the adaptive regularization in FewPLA with hand-tuning, but doing so increases the number of parameters requiring careful tuning, implying many runs with different parameters, greatly reducing the practical value of FewPLA when learning from massive data.

Another limitation of FewPLA is that it works with qualitative data only. Quantitative attributes needs to be discretized – though one can do unsupervised discretization, supervised MDL-based discretization that we recommend for FewPLA requires at least one more pass through the data and if the data can not be loaded into memory, requires a sample or subset of the data to determine the cut-points.

Note, the Random Forest results that we have reported in this work are on discretized data. Since RF can handle quantitative attributes, it is very likely that its performance will be better with quantitative attributes. However, for sake of comparison, we have kept the dataset similar across all the compared models to show the effectiveness and efficiency of FewPLA on quantitative datasets.

5 Conclusion and Future Work

In this paper, we proposed a simple yet effective limited pass algorithm that can learn from large quantities of data. Our work is motivated from the fact that for

large quantities of data, learning algorithms that are inherently out-of-core and are low-biased, preferably with minimal tuning parameters are desirable. Our proposed algorithm FewPLA learns in as little as 14 passes through the data. The data does not need to reside in memory, therefore, the size of the dataset is not a bottleneck. The bias of the model can be controlled by the parameter k^{max} and number of iterations can be selected based on the time available. We show that the performance of FewPLA is extremely competitive across a wide range of datasets when compared against state of the art in in-core classification – Random Forest.

In summary, the proposed algorithm FewPLA is the first work that employs the power of higher-order BN classifiers with efficient discriminative learning of parameters. It adopts a sophisticated adaptive regularization strategy with the salient feature of pulling the estimates back towards MLE estimates. It also includes an effective mechanism for choosing the initial step size for adaptive gradient-based optimization.

Important directions for future work include:

- As discussed in the previous section, FewPLA with a higher k does not always give better performance than a lower k. In the future, we seek FewPLA modification so that this problem is overcome. We intend to determine the accuracy of the model by adding a LOOCV pass after every SGD iteration. This can be done efficiently by removing data-point from the counts and decrementing the parameter $\mathbf{w}$ accordingly. This will also have the effect of choosing the best number of iterations.
- A better regularization mechanism to prevent over-fitting will be sought.

6 Code

The details of the software library `fupla` is given in Appendix A. The library along with running instructions can be downloaded from Github: `https://github.com/nayyarzaidi/fupla.git`.

7 Acknowledgments

This research has been supported by the Australian Research Council (ARC) under grant DP140100087, and by the Asian Office of Aerospace Research and Development, Air Force Office of Scientific Research under contract FA2386-15-1-4007.

A `fupla` – Code Library

FewPLA provides an extremely efficient library for learning from large quantities of data. A typical command line argument is of the form: `java -cp /fupla.jar fupla.TwoFoldXValOOCFupla -t /dataset.arff -V -X 2 -K 4 -W -S -D -O "adagrad" -I 10 -E 0.05 -C -R -L 0.01 -A`. This does two rounds of two-fold ($X = 2$) cross validation on dataset name dataset.arff and runs FewPLA

with $K = 4$. The $-W$ flag should be left as it is. The $-S$ flag tells the software to do a SKDB pass. $-O$ with 'adagrad' means to use adaptive gradient algorithm. Other options are: $\{\text{adagrad}, \text{adaptiv}, \text{adadelta}, \text{nplr}\}$. $-I$ flag specifies the number of iterations. $-E$ flag species the step size of the problem. However, if $-C$ flag is set, it runs the Algorithm 3. $-R$ flag specifies to add regularization where $-L$ is the value of λ. The $-A$ flag states to do a adaptive regularization instead.

References

1. J. Langford, L. Li, and A. Strehl, "Vowpal wabbit online learning project," 2007.
2. N. A. Zaidi, G. I. Webb, M. J. Carman, F. Petitjean, and J. Cerquides, "ALR^n: Accelerating higher-order logistic regression," *Machine Learning*, vol. 104, pp. 151–194, 2016.
3. B. D. Ripley, *Pattern Recognition and Neural Networks*. Cambridge University Press, 1996.
4. N. A. Zaidi, G. I. Webb, M. J. Carman, F. Petitjean, W. Buntine, M. Hynes, and D. S. H., "Efficient parameter learning of Bayesian network classifiers," *Machine Learning*, vol. 106, pp. 1–44, 2016.
5. B. Dalessandro, T. Raeder, and F. Provost, "Scalable hands-free transfer learning for online advertising," in *ACM KDD*, 2014.
6. C. K. Chow and C. N. Liu, "Approximating discrete probability distributions with dependence trees," *IEEE Transactions on Information Theory*, vol. 14, pp. 462–467, 1968.
7. P. Langley and S. Sage, "Induction of selective Bayesian classifiers," in *UAI*. Morgan Kaufmann, 1994, pp. 399–406.
8. D. Koller and M. Sahami, "Toward optimal feature selection," in *Proceedings of the Thirteenth International Conference on Machine Learning (ICML)*, L. Saitta, Ed. Morgan Kaufmann Publishers, 1996, pp. 284–292.
9. R. Kohavi, "The power of decision tables," in *Proceedings of the European Conference on Machine Learning ECML-95*, vol. 912, 1995, pp. 174–189.
10. S. Rendle, "Learning recommender systems with adaptive regularization," in *ACM International Conference on Web Search and Data Mining*, 2012, pp. 133–142.
11. J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization," *J. Mach. Learn. Res.*, vol. 12, pp. 2121–2159, 2011.
12. M. D. Zeiler, "ADADELTA: an adaptive learning rate method," *CoRR*, vol. abs/1212.5701, 2012.
13. T. Schaul, S. Zhang, and Y. LeCun, "No more pesky learning rates," in *Proceedings of the 30th International Conference on Machine Learning*, vol. 28, no. 3. PMLR, 2013, pp. 343–351.
14. L. Bottou, "Stochastic gradient descent tricks." in *Neural Networks: Tricks of the Trade (2nd ed.)*, 2012, vol. 7700, pp. 421–436.
15. A. Frank and A. Asuncion, "UCI machine learning repository," 2010. [Online]. Available: http://archive.ics.uci.edu/ml
16. F. Petitjean, J. Inglada, and P. Gançarski, "Satellite Image Time Series Analysis under Time Warping," *IEEE Transactions on Geoscience and Remote Sensing*, vol. 50, no. 8, pp. 3081–3095, 2012.
17. U. M. Fayyad and K. B. Irani, "On the handling of continuous-valued attributes in decision tree generation," *Machine Learning*, vol. 8, no. 1, pp. 87–102, 1992.

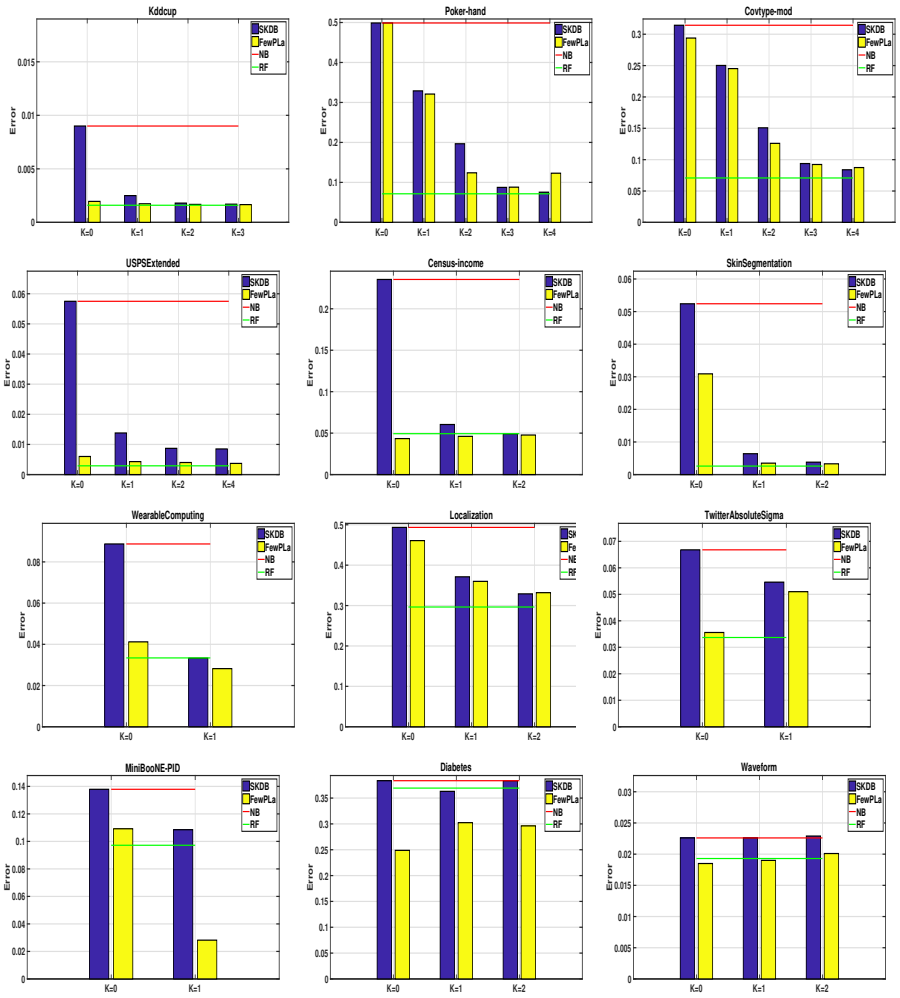

Fig. 3. Comparison of FewPLA 0-1 Loss performance with SKDB, naive Bayes and Random Forest on 12 datasets. Datasets are decreasing order of their size.

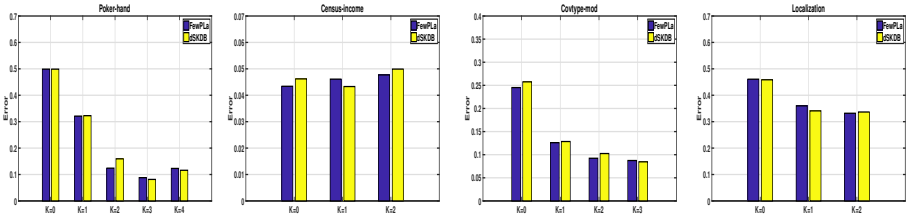

Fig. 4. Comparison of FewPLA 0-1 Loss performance with in-core discriminative SKDB trained with quasi-Newton. Note the Y-axis scale varies across different figures.