Journal of Machine Learning Research 10 (2009) 1367-1385 Submitted 1/09; Revised 4/09; Published 7/09

A Parameter-Free Classification Method for Large Scale Learning

Marc Boullé MARC.BOULLE@ORANGE-FTGROUP.COM
Orange Labs
2, avenue Pierre Marzin
22300 Lannion, France

Editor: Soeren Sonnenburg

Abstract

With the rapid growth of computer storage capacities, available data and demand for scoring models both follow an increasing trend, sharper than that of the processing power. However, the main limitation to a wide spread of data mining solutions is the non-increasing availability of skilled data analysts, which play a key role in data preparation and model selection.

In this paper, we present a parameter-free scalable classification method, which is a step towards fully automatic data mining. The method is based on Bayes optimal univariate conditional density estimators, naive Bayes classification enhanced with a Bayesian variable selection scheme, and averaging of models using a logarithmic smoothing of the posterior distribution. We focus on the complexity of the algorithms and show how they can cope with data sets that are far larger than the available central memory. We finally report results on the Large Scale Learning challenge, where our method obtains state of the art performance within practicable computation time.

Keywords: large scale learning, naive Bayes, Bayesianism, model selection, model averaging

1. Introduction

Data mining is "the non-trivial process of identifying valid, novel , potentially useful, and ultimately understandable patterns in data" (Fayyad et al., 1996). Several industrial partners have proposed to formalize this process using a methodological guide named CRISP-DM, for CRoss Industry Standard Process for Data Mining (Chapman et al., 2000). The CRISP-DM model provides an overview of the life cycle of a data mining project, which consists in the following phases: business understanding, data understanding, data preparation, modeling, evaluation and deployment. Practical data mining projects involve a large variety of constraints, like scalable training algorithms, understandable models, easy and fast deployment. In a large telecommunication companies like France Telecom, data mining applies to many domains: marketing, text mining, web mining, traffic classification, sociology, ergonomics. The available data is heterogeneous, with numerical and categorical variables, target variables with multiple classes, missing values, noisy unbalanced distributions, and with numbers of instances or variables which can vary over several orders of magnitude. The most limiting factor which slows down the spread of data mining solutions is the data preparation phase, which consumes 80% of the process (Pyle, 1999; Mamdouh, 2006) and requires skilled data analysts. In this paper, we present a method[1] (Boullé, 2007) which aims at automatizing the data preparation and modeling phases of a data mining project, and which performs well on a large variety of problems. The interest of the paper, which extends the workshop abstract (Boullé, 2008), is

1. The method is available as a shareware, downloadable at `http://perso.rd.francetelecom.fr/boulle/`.

to introduce the algorithmic extensions of the method to large scale learning, when data does not fit into memory, and to evaluate them in an international challenge.

The paper is organized as follows. Section 2 summarizes the principles of our method and Section 3 focuses on its computational complexity, with a detailed description of the extensions introduced for large scale learning. Section 4 reports the results obtained on the Large Scale Learning challenge (Sonnenburg et al., 2008). Finally, Section 5 gives a summary and discusses future work.

2. A Parameter-Free Classifier

Our method, introduced in Boullé (2007), extends the naive Bayes classifier owing to an optimal estimation of the class conditional probabilities, a Bayesian variable selection and a compression-based model averaging.

2.1 Optimal Discretization

The naive Bayes classifier has proved to be very effective on many real data applications (Langley et al., 1992; Hand and Yu, 2001). It is based on the assumption that the variables are independent within each output class, and solely relies on the estimation of univariate conditional probabilities. The evaluation of these probabilities for numerical variables has already been discussed in the literature (Dougherty et al., 1995; Liu et al., 2002). Experiments demonstrate that even a simple equal width discretization brings superior performance compared to the assumption using a Gaussian distribution. In the MODL approach (Boullé, 2006), the discretization is turned into a model selection problem and solved in a Bayesian way. First, a space of discretization models is defined. The parameters of a specific discretization are the number of intervals, the bounds of the intervals and the output frequencies in each interval. Then, a prior distribution is proposed on this model space. This prior exploits the hierarchy of the parameters: the number of intervals is first chosen, then the bounds of the intervals and finally the output frequencies. The choice is uniform at each stage of the hierarchy. Finally, the multinomial distributions of the output values in each interval are assumed to be independent from each other. A Bayesian approach is applied to select the best discretization model, which is found by maximizing the probability $p(Model|Data)$ of the model given the data. Owing to the definition of the model space and its prior distribution, the Bayes formula is applicable to derive an exact analytical criterion to evaluate the posterior probability of a discretization model. Efficient search heuristics allow to find the most probable discretization given the data sample. Extensive comparative experiments report high performance.

The case of categorical variables is treated with the same approach in Boullé (2005), using a family of conditional density estimators which partition the input values into groups of values.

2.2 Bayesian Approach for Variable Selection

The naive independence assumption can harm the performance when violated. In order to better deal with highly correlated variables, the selective naive Bayes approach (Langley and Sage, 1994) exploits a wrapper approach (Kohavi and John, 1997) to select the subset of variables which optimizes the classification accuracy. Although the selective naive Bayes approach performs quite well on data sets with a reasonable number of variables, it does not scale on very large data sets with hundreds of thousands of instances and thousands of variables, such as in marketing applications or text mining. The problem comes both from the search algorithm, whose complexity is quadratic in the number of variables, and from the selection process which is prone to overfitting. In Boullé (2007), the overfitting problem is tackled by relying on a Bayesian approach, where the best model is found by maximizing the probability of the model given the data. The parameters of a variable selection model are the number of selected variables and the subset of variables. A hierarchic prior is considered, by first choosing the number of selected variables and second choosing the subset of selected variables. The conditional likelihood of the models exploits the naive Bayes assumption, which directly provides the conditional probability of each label. This allows an exact calculation of the posterior probability of the models. Efficient search heuristic with super-linear computation time are proposed, on the basis of greedy forward addition and backward elimination of variables.

2.3 Compression-Based Model Averaging

Model averaging has been successfully exploited in Bagging (Breiman, 1996) using multiple classifiers trained from re-sampled data sets. In this approach, the averaged classifier uses a voting rule to classify new instances. Unlike this approach, where each classifier has the same weight, the Bayesian Model Averaging (BMA) approach (Hoeting et al., 1999) weights the classifiers according to their posterior probability. In the case of the selective naive Bayes classifier, an inspection of the optimized models reveals that their posterior distribution is so sharply peaked that averaging them according to the BMA approach almost reduces to the maximum a posteriori (MAP) model. In this situation, averaging is useless. In order to find a trade-off between equal weights as in bagging and extremely unbalanced weights as in the BMA approach, a logarithmic smoothing of the posterior distribution, called compression-based model averaging (CMA), is introduced in Boullé (2007). Extensive experiments demonstrate that the resulting compression-based model averaging scheme clearly outperforms the Bayesian model averaging scheme.

3. Complexity Analysis

In this section, we first recall the algorithmic complexity of the algorithms detailed in Boullé (2005, 2006, 2007) in the case where all the data fits in central memory, then introduce the extension of the algorithms when data exceeds the central memory.

3.1 When Data Fits into Central Memory

The algorithm consists in three phases: data preprocessing using discretization or value grouping, variable selection and model averaging.

3.1.1 PREPROCESSING

In the case of numerical variables, the discretization Algorithm 1 exploits a greedy-heuristic. It first sorts the input values, then starts with initial single value intervals and searches for the best merge between adjacent intervals. This merge is performed if the cost (evaluation) of the discretization decreases after the merge and the process is reiterated until no further merge can decrease the discretization cost. With a straightforward implementation of the algorithm, the method runs in $O(N^3)$ time, where N is the number of instances. However, the method can be optimized in $O(N \log N)$ time owing to the additivity of the discretization cost with respect to the intervals and using a maintained sorted list of the evaluated merges (such as an AVL binary search tree for example).

Algorithm 1 Discretization

```
Input: $X_k$    // Numerical input variable
Output: $D$    // Best discretization
1: // Initialization
2: Sort the input values
3: $D =$ Elementary discretization with one interval per value value
4: // Optimization
5: $cost_{best} = cost(D), m_{best} = \emptyset$
6: repeat
7:     for all merge $m$ between adjacent intervals do
8:         if $(cost(D \cup m) < cost_{best})$ then
9:             $cost_{best} = cost(D \cup m), m_{best} = m$
10:        end if
11:    end for
12:    if $(cost_{best} = cost(D))$ then
13:        $D = D \cup m_{best}$
14:    end if
15: until no improvement
```

The case of categorical variables is handled using a similar greedy heuristic. Overall, the pre-processing phase is super-linear in time and requires $O(KN \log N)$ time, where K is the number of variables and N the number of instances.

3.1.2 VARIABLE SELECTION

In the variable selection Algorithm 2, the method alternates fast forward and backward variable selection steps based on randomized reorderings of the variables, and repeats the process several times in order to better explore the search space and reduce the variance caused by the dependence over the order of the variables.

Evaluating one variable selection for the naive Bayes classifier requires $O(KN)$ time, since the conditional probabilities, which are linear in the number of variables, require $O(K)$ time per instance. Updating the evaluation of a variable subset by adding or dropping one single variable requires only $O(1)$ time per instance and thus $O(N)$ time for all the instances. Therefore, one loop of fast forward or backward variable selection, which involves $O(K)$ adds or drops of variables, can be evaluated in $O(KN)$ time.

In order to bound the time complexity, each succession of fast forward and backward variable selection is repeated only twice ($MaxIter = 2$ in algorithm 2), which is sufficient in practice to be close to convergence within a small computation time.

The number of repeats of the outer loop is fixed to $\log_2 N + \log_2 K$, so that the overall time complexity of the variable selection algorithm is $O(KN(\log K + \log N))$, which is comparable to that of the preprocessing phase.

Algorithm 2 Variable Selection

```
Input: $X = (X_1, X_2, \ldots X_K)$ // Set of input variables
Output: $B$ // Best subset of variables
 1: $B = \emptyset$ // Start with an empty subset of variables
 2: for Step=1 to $\log_2 KN$ do
 3:    // Fast forward backward selection
 4:    $S = \emptyset$ // Initialize an empty subset of variables
 5:    $Iter = 0$
 6:    repeat
 7:       $Iter = Iter + 1$
 8:       $X' = \text{Shuffle}(X)$ // Randomly reorder the variables to add
 9:       // Fast forward selection
10:       for $X_k \in X'$ do
11:          if $(cost(S \cup \{X_k\}) < cost(S))$ then
12:             $S = S \cup \{X_k\}$
13:          end if
14:       end for
15:       $X' = \text{Shuffle}(X)$ // Randomly reorder the variables to remove
16:       // Fast backward selection
17:       for $X_k \in X'$ do
18:          if $(cost(S - \{X_k\}) < cost(S))$ then
19:             $S = S - \{X_k\}$
20:          end if
21:       end for
22:    until no improvement or $Iter \geq MaxIter$
23:    // Update best subset of variables
24:    if $(cost(S) < cost(B))$ then
25:       $B = S$
26:    end if
27: end for
```

3.1.3 Model Averaging

The model averaging algorithm consists in collecting all the models evaluated in the variable selection phase and averaging then according to a logarithmic smoothing of their posterior probability, with no overhead on the time complexity.

Overall, the preprocessing, variable selection and model averaging have a time complexity of $O(KN(\log K + \log N))$ and a space complexity of $O(KN)$.

3.2 When Data Exceeds Central Memory

Whereas standard scalable approaches, such as data sampling or online learning, cannot process all the data simultaneously and have to consider simpler models, our objective is to enhance the scalability of our algorithm without sacrificing the quality of the results.

With the $\mathrm{O}(KN)$ space complexity, large data sets cannot fit into central memory and training time becomes impracticable as soon as memory pages have to be swapped between disk and central memory.[2] To avoid this strong limitation, we enhance our algorithm with a specially designed chunking strategy.

3.2.1 CHUNKING STRATEGY

First of all, let us consider the access time from central *m*emory (t_m) and from sequential *r*ead (t_r) or random disk *a*ccess (t_a). In modern personal computers (year 2008), t_m is in the order of 10 nanoseconds. Sequential read from disk is so fast (based on 100 Mb/s transfer rates) that t_r is in the same order as t_m: CPU is sometimes a limiting factor, when parsing operations are involved. Random disk access t_a is in the order of 10 milliseconds, one million times slower than t_m or t_r. Therefore, the only way to manage very large amounts of memory space is to exploit disk in a sequential manner.

Let S=$\mathrm{O}(KN)$ be the size of the data set and M the size of the central memory. Our chunking strategy consists in minimizing the number of exchanges between disk and central memory. The overhead of the scalable version of the algorithms will be expressed in terms of sequential disk read time t_r.

3.2.2 PREPROCESSING

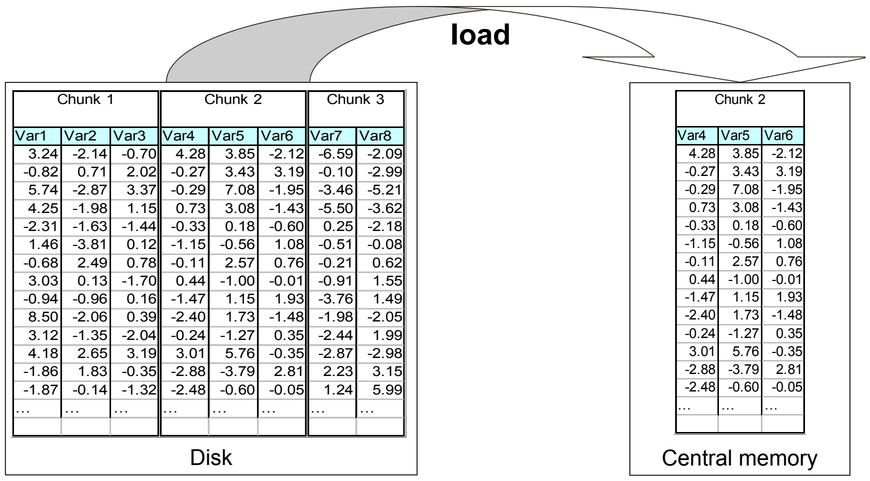
For the preprocessing phase of our method, each variable is analyzed once after being loaded into central memory. The discretization Algorithm 1 extensively accesses the values of the analyzed variable, both in the initialization step (values are sorted) and in the optimization step. We partition the set of input variables into $C = \lceil S/M \rceil$ chunks of $K_1, \ldots, K_C$ variables, such that each chunk can be loaded into memory ($K_c N < M, 1 \leq c \leq C$). The preprocessing phase loops on the C subsets, and at each step of the loop, reads the data set, parses and loads the chunk variables only, as shown in Figure 1. Each memory chunk is preprocessed as in Algorithm 1, in order to induce conditional probability tables (CPT) for the chunk variables.

After the preprocessing, the chunk variables are recoded by replacing the input values by their index in the related CPT. The recoded chunk of variables are stored in C files of integer indexes, as shown in Figure 2. These recoded chunks are very fast to load, since the chunk variables only need to be read, without any parsing (integer indexes are directly transfered from disk to central memory).

The scalable version of the preprocessing algorithm is summarized in Algorithm 3. Each input variable is read C times, but parsed and loaded only once as in the standard case. The overhead in terms of disk read time is $\mathrm{O}((C-1)KNt_r)$ time for the load steps. In the recoding step, each variable is written once on disk, which involves an overhead of $\mathrm{O}(KNt_r)$ (disk sequential write time is similar to disk sequential read time). Overall, the overhead is $\mathrm{O}(CKNt_r)$ time.

This could be improved by first splitting the input data file into C initial disk chunks (without parsing), with an overall overhead of only $\mathrm{O}(3KNt_r)$ time (read and write all the input data before preprocessing, and write recoded chunks after preprocessing). However, this involves extra disk space and the benefit is small in practice, since even C read steps are not time expensive compared to the rest of the algorithm.

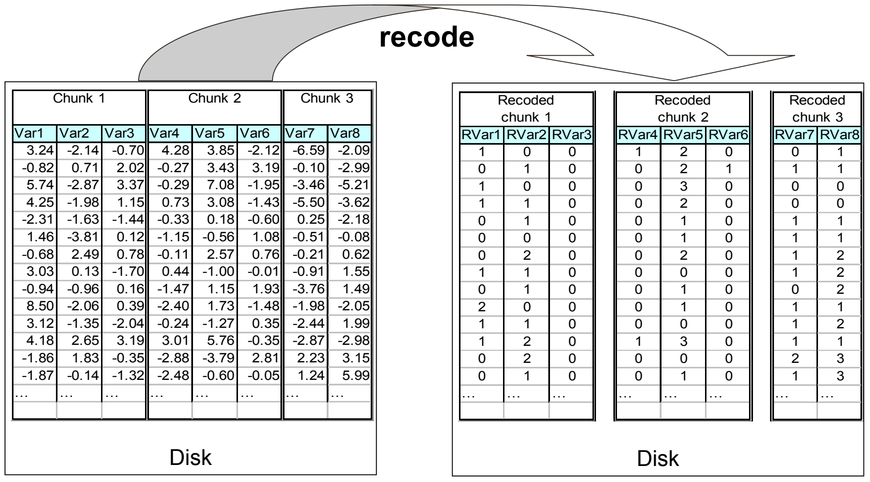
2. On 32 bits CPU, central memory is physically limited to 4 Go, or even to 2 Go on Windows PCs.



Chunk 1			Chunk 2			Chunk 3	
Var1	Var2	Var3	Var4	Var5	Var6	Var7	Var8
3.24	-2.14	-0.70	4.28	3.85	-2.12	-6.59	-2.09
-0.82	0.71	2.02	-0.27	3.43	3.19	-0.10	-2.99
5.74	-2.87	3.37	-0.29	7.08	-1.95	-3.46	-5.21
4.25	-1.98	1.15	0.73	3.08	-1.43	-5.50	-3.62
-2.31	-1.63	-1.44	-0.33	0.18	-0.60	0.25	-2.18
1.46	-3.81	0.12	-1.15	-0.56	1.08	-0.51	-0.08
-0.68	2.49	0.78	-0.11	2.57	0.76	-0.21	0.62
3.03	0.13	-1.70	0.44	-1.00	-0.01	-0.91	1.55
-0.94	-0.96	0.16	-1.47	1.15	1.93	-3.76	1.49
8.50	-2.06	0.39	-2.40	1.73	-1.48	-1.98	-2.05
3.12	-1.35	-2.04	-0.24	-1.27	0.35	-2.44	1.99
4.18	2.65	3.19	3.01	5.76	-0.35	-2.87	-2.98
-1.86	1.83	-0.35	-2.88	-3.79	2.81	2.23	3.15
-1.87	-0.14	-1.32	-2.48	-0.60	-0.05	1.24	5.99
…	…	…	…	…	…	…	…

Chunk 2		
Var4	Var5	Var6
4.28	3.85	-2.12
-0.27	3.43	3.19
-0.29	7.08	-1.95
0.73	3.08	-1.43
-0.33	0.18	-0.60
-1.15	-0.56	1.08
-0.11	2.57	0.76
0.44	-1.00	-0.01
-1.47	1.15	1.93
-2.40	1.73	-1.48
-0.24	-1.27	0.35
3.01	5.76	-0.35
-2.88	-3.79	2.81
-2.48	-0.60	-0.05
…	…	…

Figure 1: Preprocessing: the input file on disk is read several times, with one chunk of variables parsed and loaded into central memory one at a time.



Chunk 1			Chunk 2			Chunk 3	
Var1	Var2	Var3	Var4	Var5	Var6	Var7	Var8
3.24	-2.14	-0.70	4.28	3.85	-2.12	-6.59	-2.09
-0.82	0.71	2.02	-0.27	3.43	3.19	-0.10	-2.99
5.74	-2.87	3.37	-0.29	7.08	-1.95	-3.46	-5.21
4.25	-1.98	1.15	0.73	3.08	-1.43	-5.50	-3.62
-2.31	-1.63	-1.44	-0.33	0.18	-0.60	0.25	-2.18
1.46	-3.81	0.12	-1.15	-0.56	1.08	-0.51	-0.08
-0.68	2.49	0.78	-0.11	2.57	0.76	-0.21	0.62
3.03	0.13	-1.70	0.44	-1.00	-0.01	-0.91	1.55
-0.94	-0.96	0.16	-1.47	1.15	1.93	-3.76	1.49
8.50	-2.06	0.39	-2.40	1.73	-1.48	-1.98	-2.05
3.12	-1.35	-2.04	-0.24	-1.27	0.35	-2.44	1.99
4.18	2.65	3.19	3.01	5.76	-0.35	-2.87	-2.98
-1.86	1.83	-0.35	-2.88	-3.79	2.81	2.23	3.15
-1.87	-0.14	-1.32	-2.48	-0.60	-0.05	1.24	5.99
…	…	…	…	…	…	…	…

Recoded chunk 1			Recoded chunk 2			Recoded chunk 3	
RVar1	RVar2	RVar3	RVar4	RVar5	RVar6	RVar7	RVar8
1	0	0	1	2	0	0	1
0	1	0	0	2	1	1	1
1	0	0	0	3	0	0	0
1	1	0	0	2	0	0	0
0	1	0	0	1	0	1	1
0	0	0	0	1	0	1	1
0	2	0	0	2	0	1	2
1	1	0	0	0	0	1	2
0	1	0	0	1	0	0	2
2	0	0	0	1	0	1	1
1	1	0	0	0	0	1	2
1	2	0	1	3	0	1	1
0	2	0	0	0	0	2	3
0	1	0	0	1	0	1	3
…	…	…	…	…	…	…	…

Figure 2: Recoding: the input file on disk is read and recoded once, and several recoded chunks of variables are written on disk.

Algorithm 3 Scalable preprocessing

```
Input: input data on disk
Output: recoded data on several recoded disk chunks, conditional probability tables
 1: // Compute chunking parameters
 2: estimate input data size $S$ and available central memory size $M$
 3: compute number of chunks: $C = \lceil S/M \rceil$
 4: partition the set of variables into $C$ balanced chunks of variables
 5: // Main loop
 6: for all chunk do
 7:     // Load chunk into central memory
 8:     for all variable do
 9:         read input data
10:         if (variable in current chunk) then
11:             parse and load variable
12:         end if
13:     end for
14:     // Preprocessing
15:     for all variable in memory chunk do
16:         call the discretization or value grouping algorithm
17:         keep the conditional probability table (CPT) into memory
18:     end for
19:     // Recoding
20:     for all variable in memory chunk do
21:         write a recoded chunk by replacing each input value by its index in the CPT
22:     end for
23: end for
```

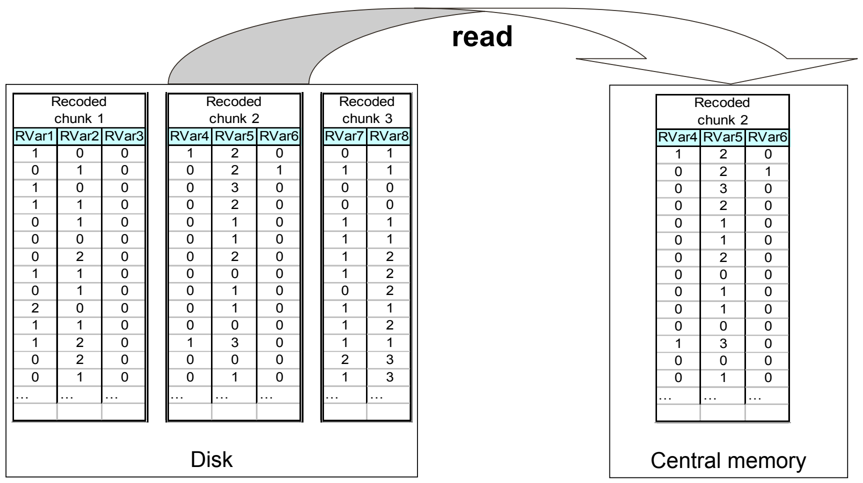
3.2.3 VARIABLE SELECTION

In the variable selection phase, the algorithm performs $O(\log K + \log N)$ steps of fast forward backward variable selection, based on random reordering of the variables. Each step evaluates the add or drop of one variable at a time, so that the whole input data has to be read.

In the scalable version of the variable selection phase, we exploit the recoded chunks of variables as shown in Figure 3: one single chunk is loaded in memory at a time. Instead of reordering randomly all the variables, we first reorder randomly the list of recoded chunks, then reorder randomly the subset of variables in the recoded chunk currently loaded in memory. This “double stage” reordering is the only place where the scalable algorithm does not exactly mirror the behavior of the standard algorithm.

Each step of fast forward backward variable selection involves reading each recoded chunk $O(1)$ times (exactly $2 * MaxIter$ times with $MaxIter = 2$), thus implies an overhead of $O(KNt_r)$ read time.

Overall, the algorithmic overhead consists of additional steps for reading the input data: $C \approx S/M$ steps for the preprocessing phase and $(\log K + \log N)$ read for variable selection phase, with an overhead of $O(KN(S/M + \log K + \log N)t_r)$ time. Since sequential disk read time t_s is comparable to memory access time, we expect that the scalable version of the algorithm has a practicable computation time.



Recoded chunk 1		
RVar1	RVar2	RVar3
1	0	0
0	1	0
1	0	0
1	1	0
0	1	0
0	0	0
0	2	0
1	1	0
0	1	0
2	0	0
1	1	0
1	2	0
0	2	0
0	1	0
…	…	…

Recoded chunk 2		
RVar4	RVar5	RVar6
1	2	0
0	2	1
0	3	0
0	2	0
0	1	0
0	1	0
0	2	0
0	0	0
0	1	0
0	1	0
0	0	0
1	3	0
0	0	0
0	1	0
…	…	…

Recoded chunk 3	
RVar7	RVar8
0	1
1	1
0	0
0	0
1	1
1	1
1	2
1	2
0	2
1	1
1	2
1	1
2	3
1	3
…	…

Recoded chunk 2		
RVar4	RVar5	RVar6
1	2	0
0	2	1
0	3	0
0	2	0
0	1	0
0	1	0
0	2	0
0	0	0
0	1	0
0	1	0
0	0	0
1	3	0
0	0	0
0	1	0
…	…	…

Figure 3: Read: the recoded chunks of variables are read and loaded (no parsing) into central memory one at a time.

4. Results on the Large Scale Learning Challenge

In this section, we summarize the Pascal Large Scale Learning Challenge (Sonnenburg et al., 2008) and describe our submission. We then analyze the advantages and limitations of our non-parametric modeling approach on large scale learning and evaluate our chunking strategy. We finaly report the challenge results.

4.1 Challenge Data Sets and Protocol

The challenge is designed to enable a direct comparison of learning methods given limited resource. The data sets, summarized in Table 1 represent a variety of domains and contain up to millions of instances, thousands of variables and tens of gigabytes of disk space.

The challenge data sets are a convenient way to evaluate the scalability of our offline method, when data set size is larger than the central memory by one order of magnitude (on computers with 32 bit CPU). In the Wild Competition Track of the challenge, the participants are "free to do anything that leads to more efficient, more accurate methods, for example, perform feature selection, find effective data representations, use efficient program-code, tune the core algorithm etc". The method have to be trained on subsets of the train data set of increasing size, with sizes $10^2, 10^3, 10^4, 10^5, \ldots$ up to the maximum number of train examples.

For all the training sessions, the accuracy is estimated using the area over the precision recall curve (aoPRC) criterion. The training time is estimated by the organizer in the final evaluation by a re-run of the algorithm on a single CPU Linux machine, 32 Gb RAM. It is noteworthy that the evaluated training time is the computation time for the optimal parameter settings, excluding data loading times and any kind of data preparation or model selection time.

Data set	Domain	Training	Validation	Dimensions	Best aoPRC	Our aoPRC
alpha	artificial	500000	100000	500	0.1345	0.2479
beta	artificial	500000	100000	500	0.4627	**0.4627**
gamma	artificial	500000	100000	500	0.0114	**0.0114**
delta	artificial	500000	100000	500	0.0798	**0.0798**
epsilon	artificial	500000	100000	2000	0.0343	0.0625
zeta	artificial	500000	100000	2000	0.0118	0.0305
fd	face detection	5469800	532400	900	0.1829	**0.1829**
ocr	character recognition	3500000	670000	1156	0.1584	0.1637
dna	dna split point detection	50000000	1000000	200	0.8045	0.8645
webspam	webspam detection	350000	50000	variable	0.0004	0.0031

Table 1: Challenge data sets and final best vs our results for the aoPRC criterion.

The overall results are summarized based on performance figures: displaying training time vs. test error, data set size vs. test error and data set size vs. training time. A final ranking is computed by aggregating a set of performance indicators computed from these figures.

4.2 Our Submission

We made one fully automatic submission, using the raw representation of the data sets, which is numerical for all the data sets except for dna where the variables are categorical.

For the two image-based data sets (fd and ocr), we also studied the impact of initial representation using centered reduced rows, and finally chose the representation leading to the best results: raw for ocr and centered-reduced for fd.

The webspam data set is represented using a tri-gram representation, which implies 16 millions of potential variables ($K = 256^3 \approx 16 * 10^6$). Although this is a sparse data set, our algorithm exploits a dense representation and cannot manage such a large number of variables. We applied a simple dimensionality reduction method inspired from the random projection technique (Vempala, 2004). We choose to build a compact representation, using a set of $K' = 10007$ (first prime number beyond 10000) constructed variables: each constructed feature is the sum of the initial variables having the same index modulo K':

$$\forall k', 0 \leq k' < K', Var_{k'} = \sum_{k, k \bmod K' = k'} Var_k.$$

As our method is available as a scoring tool for Windows end users, it is implemented for 32 bits CPUs and limited to 2 Gb RAM, thus does not benefit from the 32 Gb RAM allowed in the challenge. The algorithmic extensions presented in Section 3.2 are thus extensively exploited for the largest training sessions.

4.3 Non-Parametric Modeling with Large Data Sets

In Boullé (2007), we have analyzed the accuracy performance of our method using extensive experiments. In the section, we give new insights on our non-parametric approach and its impact on large scale learning. Actually, the MODL method considers a space of discretization models which grows with the size of the training data set. The number of intervals is taken between 1 to N intervals, where N is the number of training instances, the boundaries come from the input data and the class conditional parameters per interval are confined to a finite number of frequencies (rather than continuous distributions in [0, 1]), on the basis on instance counts in the data. Both the space of discretization models and the prior distribution on the model parameters are data-dependent. More precisely, they depends on the input data, and the best model, with the maximum a posteriori probability, is selected using the output data.

To illustrate this, we chose the delta data set and exploit the 500000 labeled instances for our experiment. We use training sets of increasing sizes 100, 1000, 10000, 100000 and 400000, and keep the last 100000 instances as a test set. The best variable according to the MODL criterion is Var150 for all data set sizes above 1000. In Figure 4, we draw the discretization (with boundaries and conditional probability of the positive class) of Var150, using the MODL method and the unsupervised equal frequency method with 10 intervals, as a baseline. For data set size 100, the empirical class distribution is noisy, and the MODL method selects a discretization having one single interval, leading to the elimination of the variable. When the data set size increases, the MODL discretizations contain more and more intervals, with up to 12 intervals in the case of data set size 400000. Compared to the fixed size equal frequency discretization, the MODL method builds discretizations which are both more accurate and robust and obtain a good approximation of the underlying class distribution.

This also demonstrates how the univariate MODL preprocessing method behaves as a variable ranking method with a clear threshold: any input variable discretized into one single interval does not contain any predictive information and can be eliminated. In Figure 5, we report the number of selected variables per data set size, which goes from 6 for data set size 100 to about 350 for data set size 400000. Figure 5 also report the test AUC (area under the ROC curve (Fawcett, 2003)), which significantly improves with the data set size and exhibits a clear correlation with the number of selected variables.

One drawback of this approach is that all the training data have to be exploited simultaneously to unable a deep optimization of the model parameters and fully benefit from their expressiveness. Although learning with very large data sets is possible, as shown in this paper, the approach is limited to offline learning and does not apply to the one-pass learning paradigm or to learning on stream.

Overall, our non-parametric approach build models of increasing complexity with the size of the training data set, in order to better approximate the underlying class distribution. Small training data sets lead to simple models, which are fast to deploy (few selected variables with simple discretization), whereas large training data sets allow more accurate models at the expense of more training time and deployment time.

4.4 Impact of the Chunking Strategy

In Section 3.2, we have proposed a chunking strategy that allows to learn with training data sets which size is far larger than central memory. The only difference with the standard algorithm is the “double stage” random reordering of the variables, which is performed prior to each step of fast forward or backward variable selection in Algorithm 2. In this section, we evaluate our chunking strategy and its impact on training time and test AUC.

We exploit the delta data set with trainings sets of increasing sizes 100, 1000, 10000, 10000, 400000, and a test set of 100000 instances. The largest training size fits into central memory on our

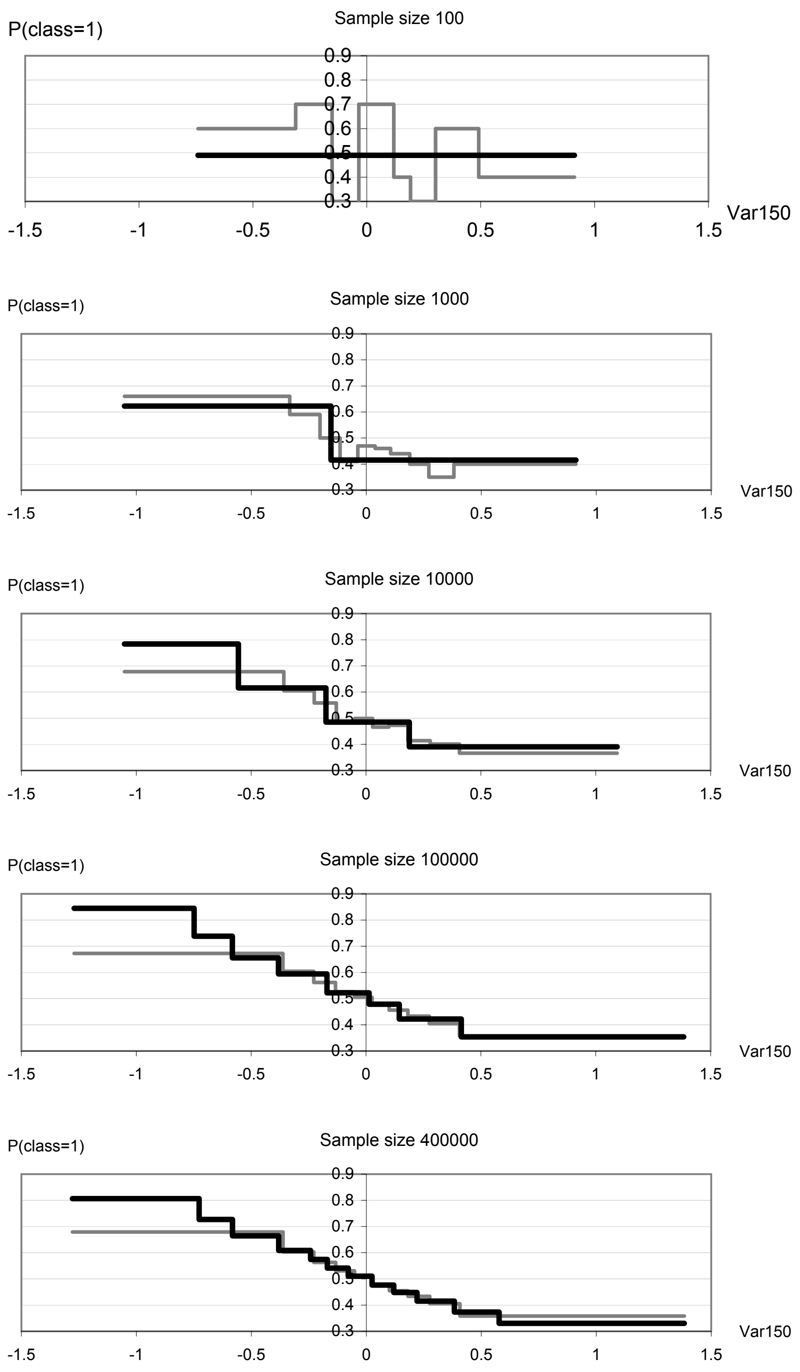


Figure 4: Discretization of Var150 for the delta data set, using the MODL method (in black) and the 10 equal frequency method (in gray).

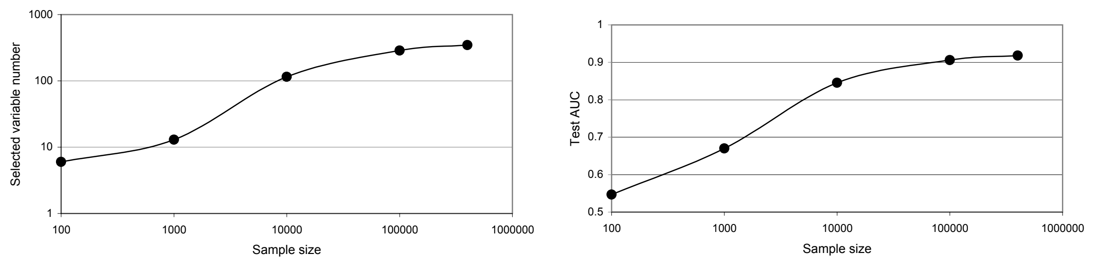


Figure 5: Data set size versus selected variable number and test AUC for the delta data set.

machine (Pentium 4, 3.2 Ghz, 2 Go RAM, Windows XP), which allows to compare the standard algorithm with the chunking strategy. In Figure 6, we report the wall-clock variable selection training time and the test AUC for the standard algorithm (chunk number = 1) and the chunking strategy, for various numbers of chunks.

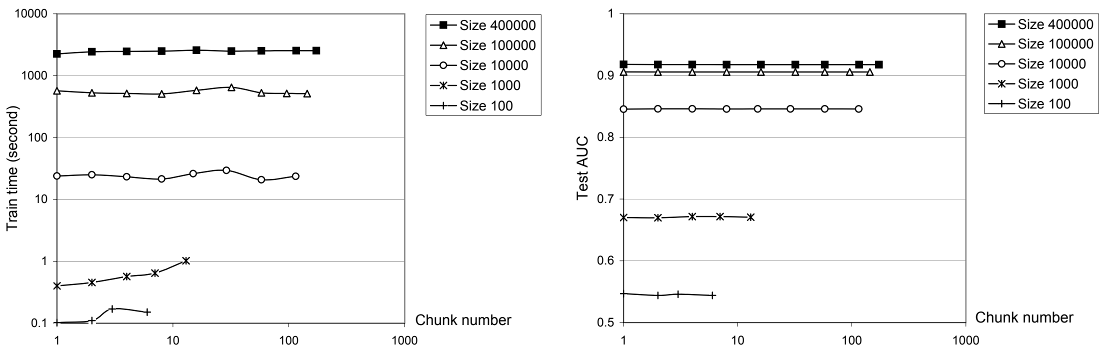


Figure 6: Impact on the chunking strategy on the wall-clock variable selection training time and the test AUC for the delta data set.

The results show that the impact on test AUC is insignificant: the largest differences go from 0.003 for data set size 100 to 0.0004 for data set size 400000. The variable selection training time is almost constant with the number of chunks. Compared to the standard algorithm, the time overhead is only 10% (for data set size above 10000), whatever be the number of chunks. Actually, as analyzed in Section 3.2, the recoded chunks of variables are extremely quick to load into memory, resulting in just a slight impact on training time.

We perform another experiment using the fd data set, with up to 5.5 millions of instances and 900 variables, which is one order of magnitude larger than our central memory resources. Figure 7 reports the wall-clock full training time, including load time, preprocessing and variable selection. When the size of the data sets if far beyond the size of the central memory, the overhead in wall-clock time is by only a factor two. This overhead mainly comes from the preprocessing phase of the algorithm (see Algorithm 3), which involves several reads of the initial training data set. Overall, the wall-clock training time is about one hour per analyzed gigabyte.

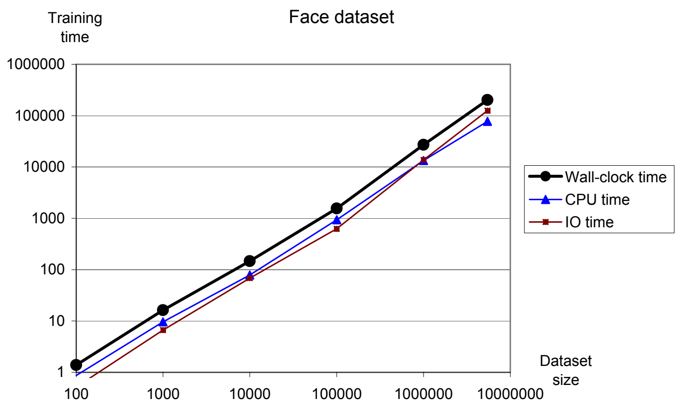


Figure 7: Training time for the fd data set.

Rank	Score	Submitter	Title
1	2.6	chap - Olivier Chapelle	Newton SVM
2	2.7	antoine - Antoine Bordes	SgdQn
3	3.2	yahoo - Olivier Keerthi	SDM SVM L2
4	3.5	yahoo - Olivier Keerthi	SDM SVM L1
5	5	MB - Marc Boulle	Averaging of Selective Naive Bayes Classifiers final
6	5.8	beaker - Gavin Cawley	LR
7	6.8	kristian - Kristian Woodsend	IPM SVM 2
8	7.3	ker2 - Porter Chang	CTJ LSVM01
9	7.7	antoine - Antoine Bordes	LaRankConverged
10	8	rofu - Rofu yu	liblinear

Table 2: Challenge final ranking.

4.5 Challenge Results

The challenge attracted 49 participants who registered, for a total of 44 submissions. In the final evaluation, our submission is ranked 5^{th} using the the aggregated performance indicator of the organizers, as shown in Table 2. To go beyond this aggregated indicator and analyze the intrinsic multi-criterion nature of the results, we now report and comment separately the training time and accuracy performance of our method.

4.5.1 TRAINING TIME

Most of the other competitors exploit linear SVM, using for example Newton based optimization in the primal (Chapelle, 2007), Sequential Dual Method optimization (Hsieh et al., 2008) or stochastic gradient optimization (Bordes and Bottou, 2008). These method were applied using a specific data preparation per data set, such as 0-1, L1 or L2 normalization, and the model parameters were tuned using most of the available training data, even for the smallest training sessions. They exploit incremental algorithms, similar to online learning methods: they are very fast and need only a few passes on the data to reach convergence.

We report in Figure 8 the training time vs data set size performance curves (in black for our results).[3] As far as the computation time only is considered, our offline training time is much longer than that of the online methods of the other competitors, by about two orders of magnitude. However, when the overall process is accounted for, with data preparation, model selection and data loading time, our training time becomes competitive, with about one hour of training time per analyzed gigabyte.

4.5.2 ACCURACY

Table 1 reports our accuracy results in the challenge for the area over the precision recall curve (aoPRC) criterion in the final evaluation of the challenge. The detailed results are presented in Figure 9, using the data set size vs accuracy performance curves.

On the webspam data set, the linear SVM submissions of Keerthi obtain excellent performance, which confirms the effectiveness of SVM for text categorization (Joachims, 1998). On the alpha data set, our method (based on the naive Bayes assumption) fails to exploit the quadratic nature of this artificial data set and obtains poor results. Except for this data set, our method always ranks among the first competitors and obtains the 1^{st} place with a large margin on four data sets: beta, gamma, delta and fd. This is a remarkable performance for a fully automatic method, which exploits the limited naive Bayes assumption.

Overall, our method is highly scalable and obtains competitive performance fully automatically, without tuning any parameter.

5. Conclusion

We have presented a parameter-free classification method that exploits the naive Bayes assumption. It estimates the univariate conditional probabilities using the MODL method, with Bayes optimal discretizations and value groupings for numerical and categorical variables. It searches for a subset of variables consistent with the naive Bayes assumption, using an evaluation based on a Bayesian model selection approach and efficient add-drop greedy heuristics. Finally, it combines all the evaluated models using a compression-based averaging schema. In this paper, we have introduced a carefully designed chunking strategy, such that our method is no longer limited to data sets that fit into central memory.

Our classifier is aimed at automatically producing competitive predictions in a large variety of data mining contexts. Our results in the Large Scale Learning Challenge demonstrates that our method is highly scalable and automatically builds state-of-the art classifiers. When the data set size is larger than the available central memory by one order of magnitude, our method exploits an efficient chunking strategy, with a time overhead of only a factor two.

Compared to alternative methods, our method requires all the data simultaneously to fully exploit the potential of our non-parametric preprocessing: it is thus dedicated to offline learning, not to online learning. Another limitation is our selective naive Bayes assumption. Although this is often leveraged in practice when large number of features can be constructed using domain knowledge, alternative methods might be more effective when their bias fit the data. Overall, our approach is

3. For readability reasons in this paper, we have represented our results in black and the others in gray, which allows to present our performance relatively to the distribution of all the results. On the challenge website, Figures 8 and 9 come in color, which allows to discriminate the performance of each method.

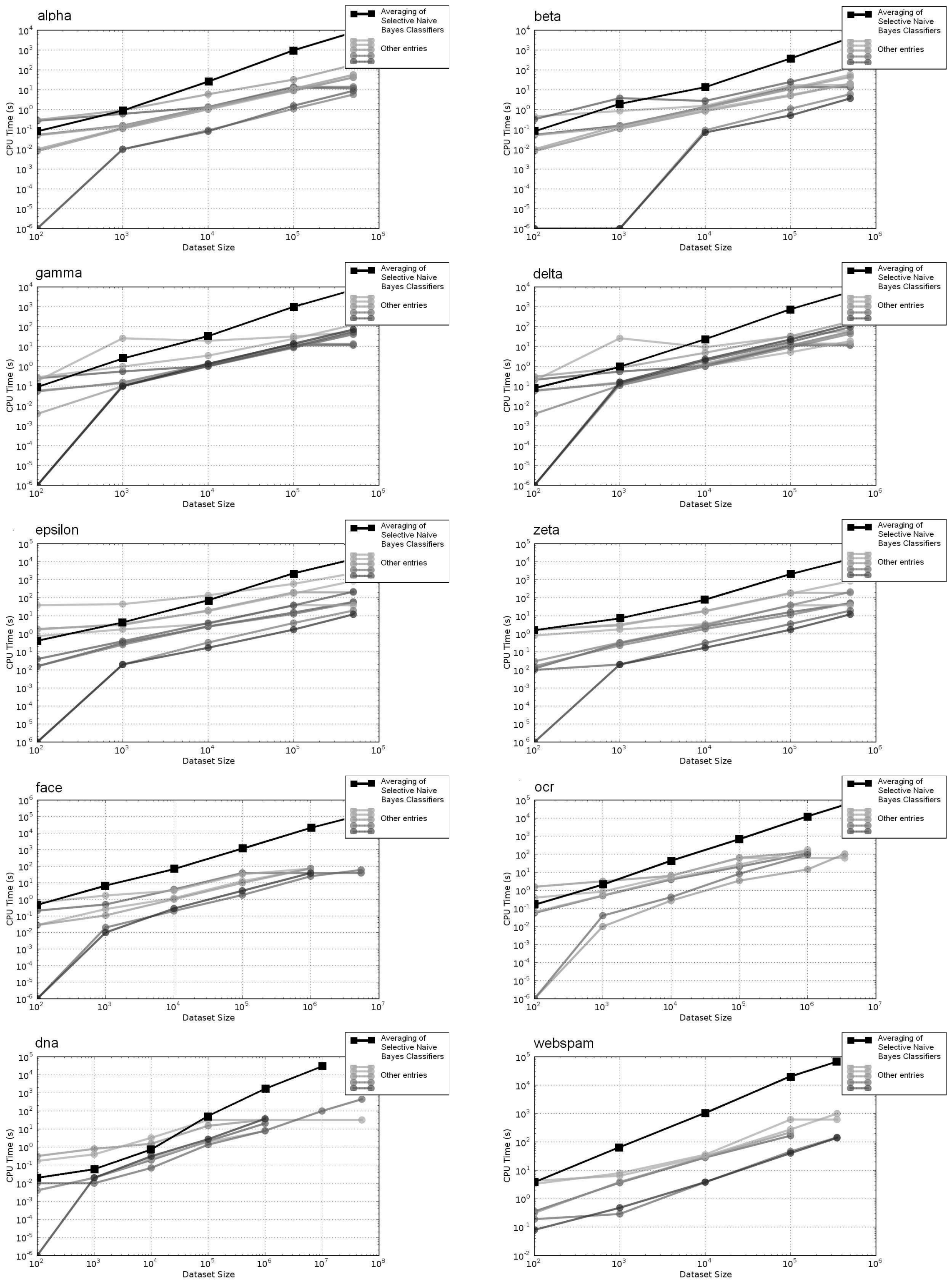


Figure 8: Final training time results: data set size versus training time (excluding data loading).

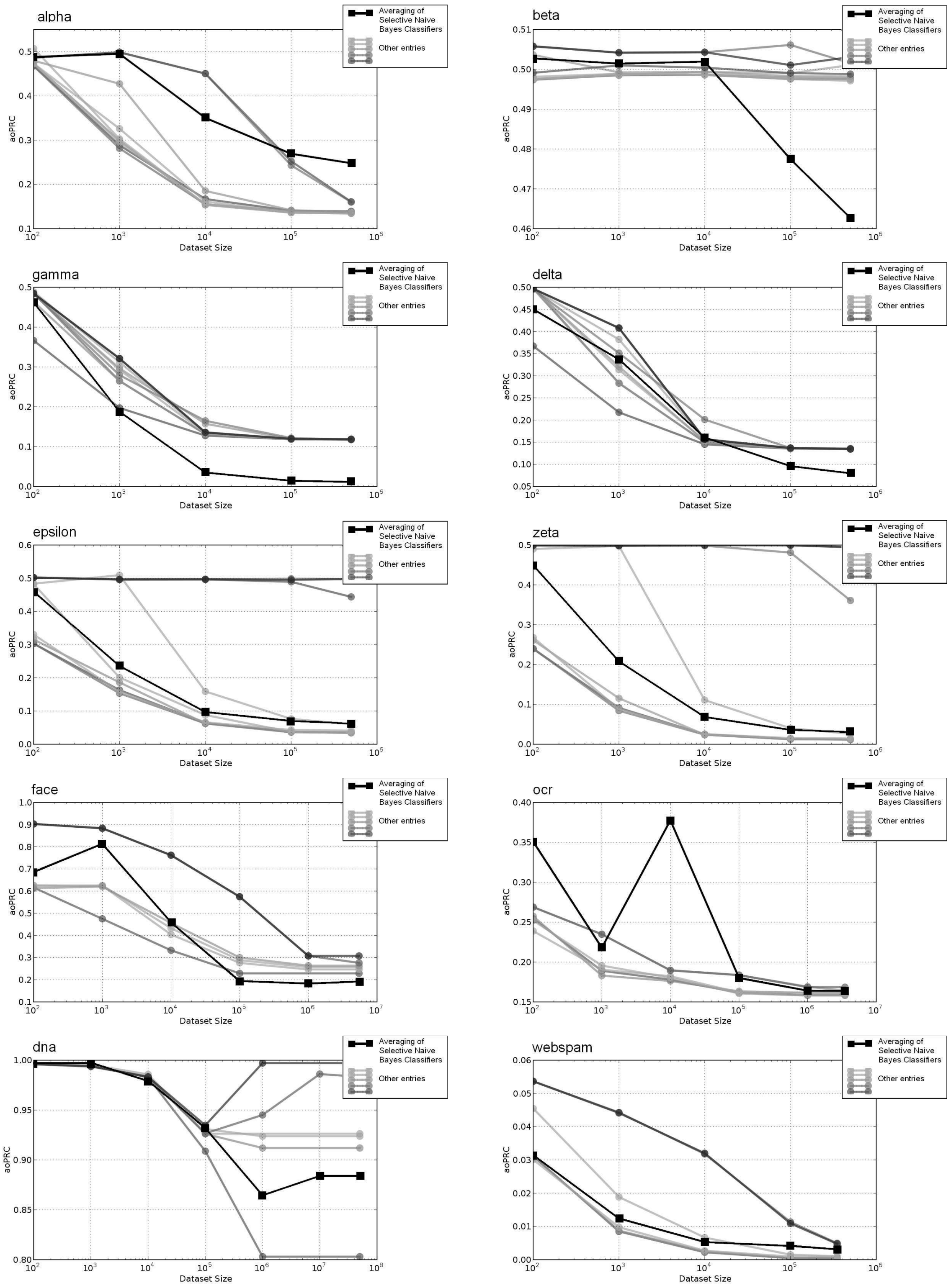


Figure 9: Final accuracy results: data set size versus test error.

of great interest for automatizing the data preparation and modeling phases of data mining, and exploits as much as possible all the available training data in its initial representation.

In future work, we plan to further improve the method and extend it to classification with large number of class values and to regression.

Acknowledgments

I am grateful to Bruno Guerraz, who got the initial Windows code to work under Linux. I would also like to thank the organizers of the Large Scale Learning Challenge for their valuable initiative.

References

A. Bordes and L. Bottou. Sgd-qn, larank: Fast optimizers for linear svms. In *ICML 2008 Workshop for PASCAL Large Scale Learning Challenge*, 2008. http://largescale.first.fraunhofer.de/workshop/.

M. Boullé. A Bayes optimal approach for partitioning the values of categorical attributes. *Journal of Machine Learning Research*, 6:1431–1452, 2005.

M. Boullé. MODL: a Bayes optimal discretization method for continuous attributes. *Machine Learning*, 65(1):131–165, 2006.

M. Boullé. Compression-based averaging of selective naive Bayes classifiers. *Journal of Machine Learning Research*, 8:1659–1685, 2007.

M. Boullé. An efficient parameter-free method for large scale offline learning. In *ICML 2008 Workshop for PASCAL Large Scale Learning Challenge*, 2008. http://largescale.first.fraunhofer.de/workshop/.

L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

O. Chapelle. Training a support vector machine in the primal. *Neural Computation*, 19:1155–1178, 2007.

P. Chapman, J. Clinton, R. Kerber, T. Khabaza, T. Reinartz, C. Shearer, and R. Wirth. *CRISP-DM 1.0 : Step-by-step Data Mining Guide*, 2000.

J. Dougherty, R. Kohavi, and M. Sahami. Supervised and unsupervised discretization of continuous features. In *Proceedings of the 12th International Conference on Machine Learning*, pages 194–202. Morgan Kaufmann, San Francisco, CA, 1995.

T. Fawcett. ROC graphs: Notes and practical considerations for researchers. Technical Report HPL-2003-4, HP Laboratories, 2003.

U.M. Fayyad, G. Piatetsky-Shapiro, and P. Smyth. Knowledge discovery and data mining: towards a unifying framework. In *KDD*, pages 82–88, 1996.

D.J. Hand and K. Yu. Idiot bayes ? not so stupid after all? *International Statistical Review*, 69(3): 385–399, 2001.

J.A. Hoeting, D. Madigan, A.E. Raftery, and C.T. Volinsky. Bayesian model averaging: A tutorial. *Statistical Science*, 14(4):382–417, 1999.

C-J. Hsieh, K-W. Chang, C-J. Lin, S. Keerthi, and S. Sundararajan. A dual coordinate descent method for large-scale linear svm. In *ICML '08: Proceedings of the 25th international conference on Machine learning*, pages 408–415, New York, NY, USA, 2008. ACM.

T. Joachims. Text categorization with support vector machines: Learning with many relevant features. In *European Conference on Machine Learning (ECML)*, pages 137–142, Berlin, 1998. Springer.

R. Kohavi and G. John. Wrappers for feature selection. *Artificial Intelligence*, 97(1-2):273–324, 1997.

P. Langley and S. Sage. Induction of selective Bayesian classifiers. In *Proceedings of the 10th Conference on Uncertainty in Artificial Intelligence*, pages 399–406. Morgan Kaufmann, 1994.

P. Langley, W. Iba, and K. Thompson. An analysis of Bayesian classifiers. In *10th National Conference on Artificial Intelligence*, pages 223–228. AAAI Press, 1992.

H. Liu, F. Hussain, C.L. Tan, and M. Dash. Discretization: An enabling technique. *Data Mining and Knowledge Discovery*, 4(6):393–423, 2002.

R. Mamdouh. *Data Preparation for Data Mining Using SAS*. Morgan Kaufmann Publishers, 2006.

D. Pyle. *Data Preparation for Data Mining*. Morgan Kaufmann Publishers, Inc. San Francisco, USA, 1999.

S. Sonnenburg, V. Franc, E. Yom-Tov, and M. Sebag. Pascal large scale learning challenge, 2008. http://largescale.first.fraunhofer.de/about/.

S. Vempala. *The Random Projection Method*, volume 65 of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*. American Mathematical Society, 2004.