Chapter 1

Online learning of a weighted selective naive Bayes classifier with non-convex optimization

Carine Hue, Marc Boullé and Vincent Lemaire

Abstract We study supervised classification for data streams with a high number of input variables. The basic naïve Bayes classifier is attractive for its simplicity and performance when the strong assumption of conditional independence is valid. Variable selection and model averaging are two common ways to improve this model. This process leads to manipulate a weighted naïve Bayes classifier. We focus here on direct estimation of weighted naïve Bayes classifiers. We propose a sparse regularization of the model log-likelihood which takes into account knowledge relative to each input variable. The sparse regularized likelihood being non convex, we propose an online gradient algorithm using mini-batches and random perturbation according to a metaheuristic to avoid local minima. In our experiments, we first study the optimization quality, then the classifier performance under varying its parameterization. The results confirm the effectiveness of our approach.

Key words: supervised classification, naïve Bayes classifier, non-convex optimization, stochastic optimization, variables selection, sparse regularization

1.1 Introduction

Due to a continuous increase of storage capacities, data acquisition and processing have deeply evolved during the last decades. Henceforth, it is common to process data including a very large number of variables. Data amounts are so massive that it hardly seems possible to fully load them: online processing is then applied and data are seen only once. In this context, we consider the supervised classification problem where Y is a categorical target variable with J values $C_1, \dots, C_J$ and $X = (X_1, \dots, X_K)$ is the set of K explanatory variables, numerical or categorical.

Orange Labs, 2 avenue Pierre Marzin, 22300 Lannion, email : Firstname.Name@orange.com

Among the solutions to the problems of learning on data streams, the incremental learning algorithms are one of the most used techniques. These algorithms are able to update their model using just the new examples.

In this article we focus on one of the most used classifier in the literature i.e. the naïve Bayes classifier. A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with naive conditional independence assumption. The explanatory variables $(X_k)_{k=1,\dots,K}$ are assumed to be independent given the target variable Y. Despite this strong assumption this classifier has proved to be very effective [Hand and Yu, 2001] on many real applications and is often used on data stream for supervised classification [Gama, 2010].

This "naïve" assumption allows us to compute the model directly from the univariate conditional estimates $P(X_k|C)$. For an instance denoted n, the probability of the target modality C conditionally to the value of the explanatory variables is computed according to the formulae [1]:

$$P_w(Y = C|X = x^n) = \frac{P(Y = C)\prod_{k=1}^{K} p(x_k^n|C)}{\sum_{j=1}^{J} P(C_j)\prod_{k=1}^{K} p(x_k^n|C_j)} \tag{1.1}$$

The literature shows that variable selection [Koller and Sahami, 1996], [Langley et al., 1992] or model averaging [Hoeting et al., 1999] can improve the classification results for batch learning. These two processes can be mixed iteratively. Moreover Boullé in [Boullé, 2006] shows the close relation between weighting variables and averaging naive Bayes classifiers in the sense that, in the end, the two processes produce a similar single model where a weight is given to each explanatory variable. Equation 1.1 is just turned to the following equation :

$$P_w(Y = C|X = x^n) = \frac{P(Y = C)\prod_{k=1}^{K} p(x_k^n|C)^{w_k}}{\sum_{j=1}^{J} P(C_j)\prod_{k=1}^{K} p(x_k^n|C_j)^{w_k}} \tag{1.2}$$

In this paper, we particularly focus on weighing variables for data streams. We are not interested by learning ensemble of models which are then combined by fusion or selection [Kuncheva and Rodríguez, 2007] or ensemble of models where individual classifiers do not share the same subset of used variables [Godec et al., 2010]. One of the advantages of the classifier described by Equation (1.2) in the context of data stream is its low complexity for deployment, which only depends on the number of explanatory variables : a weighted naïve Bayes classifier is completely described by its weight vector $w = (w_1, w_2, \dots, w_K)$. The interpretation of the results is also simpler that in case of ensemble of models.

[1] We consider in this paper that estimates of prior probabilities $P(Y = C_j)$ and of conditional probabilities $p(x_k|C_j)$ are available. In our experiments, these probabilities are estimated using univariate discretization or grouping according to the MODL method (see [Boullé, 2007b])

Within the 'weighted naive Bayes classifier' family, we can distinguish:

- classifiers with weights equal to 1. It corresponds to the standard naïve Bayes classifier that uses all the explanatory variables.
- classifiers with boolean weights. It corresponds to the selective naïve Bayes classifiers which selects a subset of explanatory variables. The selection is generally done by optimizing a criteria over $\{0,1\}^K$. However, when the variables number is high, such a browsing is infeasible and only a sub-optimal browsing of the space $\{0,1\}^K$ can be completed.
- classifiers with continuous weights in $[0,1]^K$. Such classifiers can be obtained by averaging classifiers with boolean weights with a weighting proportional to the posterior probability of the model [Hoeting et al., 1999] or proportionally to their compression rate [Boullé, 2007a]. However, for datasets with a very high number of variables, we observe that the models issued from averaging keep a lot of variables, which make the obtained classifiers both costly to deploy and difficult to interpret.

In the work presented in this paper, we are interested in direct estimation of the weight vector by optimization of the regularized log-likelihood in $[0,1]^K$. Our main expectation is to obtain parcimonious robust models with less variables and equivalent performance. Preliminary works [Guigourès and Boullé, 2011] have shown the interest of such a direct estimation of the weights.

Moreover, the purpose of this work has been also to focus on (i) a proposition of a sparse regularization of the log-likelihood in Section 1.2 consistent with previous offline approach [Boullé, 2007a] (ii) the setup of an online and anytime algorithm with limited budget dedicated to the optimization of the regularized criterion in Section 1.3 (iii) an evaluation of the obtained models in terms of parcimony, predictive performance and robustness. Experiments are presented in Section 1.4, before the conclusion and future work statement.

1.2 Construction of a regularized criterion

Given a dataset $D_N = (x_n, y_n)_{n=1}^N$, we are looking for the minimization of the negative log-likelihood, which is given by:

$$ll(w, D_N) = -\sum_{n=1}^{N} \left(\log P(Y = y^n) + \sum_{k=1}^{K} \log p(x_k^n | y^n)^{w_k} - \log \left(\sum_{j=1}^{J} P(C_j) \prod_{k=1}^{K} p(x_k^n | C_j)^{w_k} \right) \right) \tag{1.3}$$

Considered as a classical optimization problem, the regularization of the log-likelihood is performed by the addition of a regularization term, also called prior term, which expresses constraints on the weight vector w. The regularized criterion is :

$$CR^{D_N}(w) = -\sum_{n=1}^{N} ll(w,(x_n,y_n)) + \lambda f(w, X_1, \ldots, X_K) \tag{1.4}$$

where ll refers to the log-likelihood, f is the regularization function, and λ the regularization weight.
Several objectives have guided our choice of the regularization function:

1. Its sparsity, i.e. it favors the weight vectors composed of as much null components as possible. The L^p norm functions are usually employed with the addition of a regularization term of the form $\sum_{k=1}^{K} |w_k|^p$. All these functions are increasing and hence favor the weight vectors with low components. For $p >= 1$, the norm function L^p is convex, which makes the optimization easier and renders this function attractive. This explain the success of L^2 regularization in many contexts. For ill-posed linear problems, the ridge regression also called Tikhonov regularization [Hoerl and Kennard, 1970] uses the L^2 norm. However, the minimization of the regularization terms for $p > 1$ does not necessarily lead to variables elimination whereas the choice $p \leq 1$ favors sparse weight vectors. The Lasso method and its variants [Trevor et al., 2015] exploit the advantages of the value $p = 1$, which enables sparsity and convex optimization. For $p < 1$, the L^p regularization more exploits the sparsity effect of the norm but conducts to non convex optimization.
2. Its ability to take into account a B_k coefficient associated to each explanatory variable so that, for equivalent likelihoods, the "simple" variables are preferred to "complex" ones. By weighting the term with L^p norm by such a coefficient, we obtain a penalization term of the form: $\sum_{k=1}^{K} B_k * |w_k|^p$. This coefficient is supposed to be known before the optimization. If no knowledge is available, this coefficient is fixed to 1. It can be used to include expert knowledge. In our case, this coefficient translates the preparation cost of the variable, i.e. the discretization cost for a numerical variable, resp. the grouping cost for a categorical variable described in equations (2.4), resp. (2.7), of [Boullé, 2007b].
3. Its consistency with the regularized criterion of the MODL naïve Bayes classifier with binary selection of variables [Boullé, 2007a]. In order that the two criteria coincide for $\lambda = 1$ and w_k with boolean values, we finally use the regularization term:

$$f(w, X_1, \ldots, X_K) = \sum_{k=1}^{K} (\log K - 1 + B_k) * w_k^p$$

1.3 Optimization algorithm: gradient descent with mini-batches and variable neighborhood search

Let $p_n = P(Y = y^n)$, $p_j = P(Y = C_j)$, $a_{k,n} = p(x_k^n | y^n)$, $a_{k,j} = p(x_k^n | C_j)$ be all constant quantities in this optimization problem.
The regularized criterion to minimize can be written:

$$CR^{D_N}(w) = -\sum_{n=1}^{N} \left\{ \log p_n + \sum_{k=1}^{K} (w_k * \log a_{k,n}) - \log \left(\sum_{j=1}^{J} p_j \prod_{k=1}^{K} (a_{k,j})^{w_k} \right) \right\} + \lambda \sum_{k=1}^{K} (\log K - 1 + B_k) * {w_k}^p \tag{1.5}$$

Let us note that for $w = \{0\}^K$, i.e. without using explanatory variables, the criterion value is equal to

$$CR^{D_N}(\{0\}^K) = -\sum_{n=1}^{N} \log p_n = -N \sum_{j=1}^{J} p_j \log p_j \tag{1.6}$$

that is to say N times the Shannon entropy. For each n there is actually a j_n such that $y_n = C_{j_n}$. If we denote by N_j the number of instances among n such that $y_n = C_j$, then

$$-\sum_{n=1}^{N} \log p_n = -\sum_{j=1}^{J} N_j \log p_j = -\sum_{j=1}^{J} N * \frac{N_j}{N} \log p_j = -N \sum_{j=1}^{J} p_j \log p_j \tag{1.7}$$

We want to optimize the criterion $CR^{D_N}(w)$ subject to the constraint that w takes its values in $[0,1]^K$ in order to obtain interpretable models. Our objective function consists of two terms. The first term is a convex function of w. In order to see this, let us represent its partial term $LL_n(w)$ in the following form :

$$LL_n(w) = \alpha_n + < c_n, w > + \log \left(\sum_{j=1}^{J} \exp^{-<b_{n,j}, w> - \beta_j} \right), \tag{1.8}$$

where $\alpha_n = -\log p_n$, $c_n^{(k)} = -\log a_{k,n}, k = 1, \ldots, K$, vectors $b_{n,j} \in R^K$ have components $b_{n,j}^{(k)} = -\log a_{k,j}^n, k = 1, \ldots, K$, and $\beta_j = -\log P(C_j)$. The first and second term of LL_n are resp. constant and linear in w and then are both convex. The third term is convex because of the log-convexity of $\exp(x)$ (see for instance [Lange, 2004] for definition and property of log-convexity). The second term (regularization term) is more complicated : it is not convex for $p < 1$ and its partial derivative are unbounded at the points with zero components. This makes impossible to establish theoritical guarantees even for convergence to a local solution. Efficient approaches have recently been proposed, which exploit sparsity and stochastic algorithms [Bach and Moulines, 2013], [Pilanci et al., 2015]. However, these approaches rely on convex optimization criteria. In our case of a non convex optimization problem, the main available approach is the simplest gradient method [Nesterov, 2004]. This criterion is not convex but differentiable at each weight vector with partial derivative:

$$\frac{\partial CR^{D_N}(w)}{\partial w_\gamma} = -\sum_{n=1}^{N} \left\{ \log a_{\gamma,n} - \frac{\sum_{j=1}^{J} p_j \log a_{\gamma,j} \prod_{k=1}^{K} (a_{k,j})^{w_k}}{\sum_{j=1}^{J} p_j \prod_{k=1}^{K} (a_{k,j})^{w_k}} \right\} + \lambda(\log K - 1 + B_k) * p * {w_\gamma}^{p-1} \tag{1.9}$$

The gradient $\nabla CR^{D_N}(w^t)$ is the vector of partial derivatives for $\gamma = 1, \ldots, K$. To respect the constraint that w takes its values in $[0,1]^K$, we have been interested in projected gradient descent algorithm [Bertsekas, 1976] i.e. a gradient descent algorithm for which, at each iteration, the obtained w vector is projected on $[0,1]^K$.
Several objectives have guided our choice for the algorithmic structure:

1. online algorithm: the algorithm structure is adapted to data stream processing and it does not need the processing of the entire dataset;
2. anytime algorithm: the algorithm is interruptible and is able to return the best optimization given a budgeted computational time.

Within a classical batch gradient descent algorithm, the weight vector is updated at each iteration t according to the gradient computed on all the instances. If the weight vector obtained at iteration t is denoted by w^t, the projected update at $t+1$ iteration is performed according to the equation:

$$w^{t+1} = P_{[0,1]^K}[w^t - \eta_t \nabla CR^{D_N}(w^t)] \tag{1.10}$$

where the η step may, according to the variants, be a scalar constant or vary across the iterations and/or vary according to the weight vector components. We have chosen to compute η according to the Rprop method detailed later in this section. The projection $P_{[0,1]^K}$ on $[0,1]^K$ just consists in bounding obtained values in interval $[0,1]$. This batch approach assumes that the entire dataset is available to start the optimization.
In its stochastic version, the update is done using the gradient computed on one single instance. The gradient descent may turn out to be chaotic if the variance of the gradient from one instance to another one is high.
Aiming for an online approach, we have retained a variant mixing batch and stochastic, namely mini-batch approach [Dekel et al., 2012] which consists in directing the descent according to gradients computed on successive data batches of length L. To be able to compare descent paths when the size of

```
Inputs  : $D$ : data stream
          $N$ : historical depth to evaluate the criterion
          $L$ : batch size used for weights update with $L << N$
          $w^0$ : initial weight vector
          $\eta_0$ : initial step vector
          Max : maximal number of iterations
          Tol : tolerated number of successive degradations
Outputs: $w^* = argminCR^D(w)$
          $t_{total}$ = performed iterations number
Initialization:
Iteration index : $t = 0$;
Number of successive criterion degradations : $N_{deg} = 0$ ;
while (Criterion improvement or $N_{deg} < Tol$) and $t < Max$ do
    $D_{t,L}$=t-th batch of size $L$
    $D_{t,N}$=data historical of size $N$ ending at the end of $D_{t,L}$
    $w^{t+1} = P_{[0,1]^K}\left(w^t - \eta_t \frac{1}{L} \nabla CR^{D_{t,L}}(w^t)\right)$
    Compute $\eta_{t+1}$
    Compute the criterion value on data historical of size $N$: $CR^{D_{t,N}}(w^{t+1})$
    if $CR^{D_{t,N}}(w^{t+1}) < CR^{D_{t-1,N}}(w^t)$ (i.e. criterion improvement) then
        Best value storage: $w^* = w^{t+1}$
    else
        Increment the counter of successive degradations : $N_{deg} = N_{deg} + 1$
    end
    $t = t + 1$;
end
```

Algorithm 1: Projected gradient descent with mini-batches (PGDMB)

mini-batches varies, we used a gradient standardized by the size of the mini-batches. The projected gradient descent with mini-batches is summarized in Algorithm 1.

The optimal value for step η_t has been the subject of several studies leading to more or less costly algorithms. We turned to the Rprop method [Riedmiller and Braun, 1993]: the step computation is specific for each vector component i.e. η is a step vector of dimension K, and each vector component is multiplied by a factor which is bigger, resp. smaller than 1, if the partial derivative sign change, resp. doesn't change from one iteration to another.
As far as the computational complexity is concerned, each iteration needs a criterion evaluation on a sample of size N that is to say a $O(K * N)$ complexity. The classical batch algorithm is obtained for $L = N$ and the stochastic one for $L = 1$.

As the criterion to be optimized is non convex, it often shows many local minima. It is then common to start several gradient descents with distinct random initializations (multi-start approach) hoping that one of these descent paths converges to a lower minimum. In order to make the optimization efficient and not to waste computational time at the beginning of each descent, it is also possible to modify the solution obtained after a given number of iterations in order to get out of a potential local minimum. We propose

```
Inputs  :  T : total maximal number of iterations
           NeighSize: initial neighborhood size
Inputs  : (PGDMB): $D$ : data stream
           $N$ : historical depth to evaluate the criterion
           $L$ : batch size used for weights update
           $\eta_0$ : initial step vector
           Max : maximal number of iterations for one PGDMB optimization
           Tol: tolerated number of successive degradations
Outputs: $w^* = argminCR^D(w)$
Initialization :
Initial weight vector for the first projected gradient descent with mini-batches
(PGDMB) : $w_1^0 = \{0.5\}^K$ ;
Initial optimal weight vector : $w^* = w_1^0$;
Initial iterations sum $SumT = 0$;
while $SumT < T$ do
    Compute $(w_m^*, t_{total}^m) = PGDMB(D, N, L, w_m^0, \eta_0, \text{Max}, \text{Tol})$
    $SumT = SumT + t_{total}^m$
    if $Improvement\ on\ w^*$ then
        Storage of $w^* = w_m^*$
    else
        $\text{NeighSize} = min(2 * \text{NeighSize}, 1)$
    end
    $w_{m+1}^0 = P_{[0,1]^K}(w_m^* + Random([-\text{NeighSize}, \text{NeighSize}]))$
end
```

Algorithm 2: Projected gradient descent with variable neighbor search (PGDMB-VNS)

to use a metaheuristic so that the current solution is regularly randomized within a neighborhood of variable size. This randomization is inspired from the Variable Neighborhood Search [Hansen and Mladenovic, 2001]. Our approach denoted PGDMB-VNS is described in Algorithm 2. The projected gradient descent with mini-batches is runned several times with different initialization for the weight vector w. The initial weight vectors are generated in a neighborhood of the current optimal weight vector. If the last projected gradient descent improves the optimal weight vector, then the size of the neighborhood is reduced. Otherwise it is increased. This neighborhood variation enables either to exploit promising areas or to explore new areas of the weight vector space.
It can be noticed that, for a neighborhood that completely covers $[0, 1]^K$, the PGDMB-VNS algorithm is equivalent to a multi-start algorithm with random initializations. Besides, we stress that the random perturbations can lead to a non-null component for a weight set to zero after a precedent run. One variable can re-appear during the data stream reading.
The PGDMB-VNS algorithm is anytime in the sense that an estimation of the criterion argmin is available at the end of the first gradient descent and that it is improved afterwards according to the available budget and inter-

ruptible at any time. Its entire complexity is $O(T * K * N)$ where T is the total number of budgeted iterations.

1.4 Experiments

The purpose of the first experiments is to evaluate the optimization quality obtained with PGDMB-VNS according to the size L of the mini-batches and to the total number of iterations T. To study the intrinsic quality regardless of the associated classifier predictive performance, we have set the λ weight value to 0, which means that we directly optimize the non regularized likelihood. The second part of the experiments deals with predictive performance of the classifier obtained by optimization of regularized criterion ($\lambda \neq 0$).
For the whole experiments, the parameters for PGDMB algorithm are set to the following values:

- $w_0 = \{0.5\}^K$
- $\eta_0 = \{10^{-2}\}^K$ with a multiplication by 0.5, resp. 1.2, in case of sign change, resp. no sign change, between two succesive gradients
- Max $= 100$ the iteration maximal number (i.e. the number of treated mini-batches). We have checked that this threshold had never been reached for the 36 tested datasets.
- Tol $= 5$ the number of authorized successive degradations

Improvement of the criterion is considered for a decreasing of at least $\epsilon = 10^{-4}$ with regard to the precedent criterion value. The weights smaller than 10^{-3} are set to 0.
When a VNS metaheuristic is applied, the initial neighborhood size is set to 1/16.

The whole experiments have been done in 10-fold-cross-validation on the 36 UCI datasets described in Table 1.1. According to the values of L and N, it can be necessary to use the instances in several mini-batches. In this case, the datasets are randomly shuffled between two mini-batches.
In the results, 'SNB' stands for the performance of a selective naïve Bayes classifier with model averaging [Boullé, 2007a].

1.4.1 Experiments on optimization quality

First of all, we have studied the PGDMB algorithm performance, that is to say, the performance of the projected gradient descent algorithm according to the mini-batch size denoted L, without using MS or VNS metaheuristic.

Dataset	Ni	Nv	Nc	Dataset	Ni	Nv	Nc
Abalone	4177	8	28	Mushroom	8416	22	2
Adult	48842	15	2	PenDigits	10992	16	10
Australian	690	14	2	Phoneme	2254	256	5
Breast	699	10	2	Pima	768	8	2
Bupa	345	6	2	Satimage	768	8	6
Crx	690	15	2	Segmentation	2310	19	7
Flag	194	29	8	Shuttle	58000	9	7
German	1000	24	2	SickEuthyroid	3163	25	2
Glass	214	10	6	Sonar	208	60	2
Heart	270	13	2	Soybean	376	35	19
Hepatitis	155	19	2	Spam	4307	57	2
Horsecolic	368	27	2	Thyroid	7200	21	3
Hypothyroid	3163	25	2	Tictactoe	958	9	2
Ionospehre	351	34	2	Vehicle	846	18	4
Iris	150	4	3	Waveform	5000	21	3
LED	1000	7	10	WaveformNoise	5000	40	3
LED17	10000	24	10	Wine	178	13	3
Letter	20000	16	26	Yeast	1484	9	10

Table 1.1 Description of the 36 UCI datasets: Ni=instances number, Nv=initial number of variables, Nc=class number.

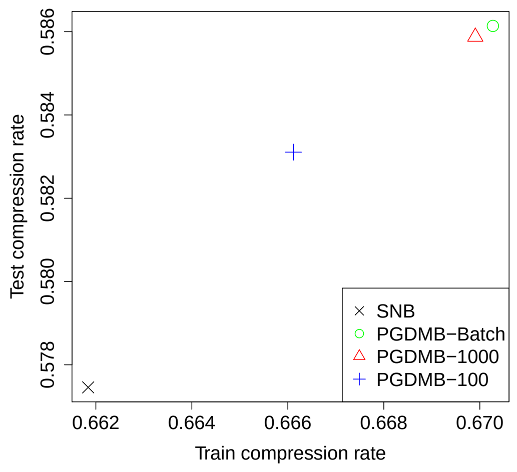


Fig. 1.1 Train and test mean compression rate for 36 UCI datasets

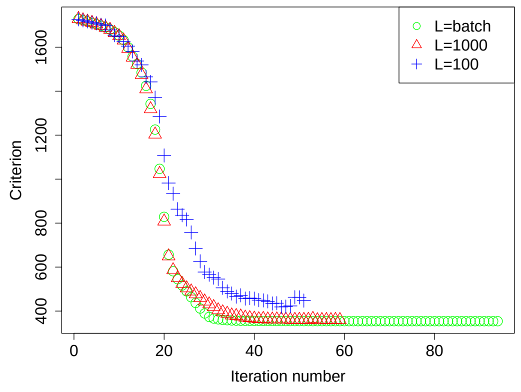


Fig. 1.2 Criterion convergence paths according to the mini-batches size (PGDMB) for the Phoneme dataset

We have chosen the compression rate as optimization quality indicator. It measures the complement to 1 of the negative logarithm of the model likelihood normalized by the Shannon entropy. As noticed in Section 1.3, for the "random" classifier with only null weights, the negative logarithm of the likelihood is equal to the Shannon entropy, which leads to a compression rate equal to 0. The closer the rate is to 1, the higher is the model likelihood. For model less competitive than the random model, compression rate is negative. The value of the compression rate on train data is then a good indicator of the optimization quality as the non regularized criterion is reduced to the negative log-likelihood.
Figure 1.1 presents the train and test compression rate averaged on 36 UCI datasets for various mini-batches sizes $L = 100, 1000, N$. In the last case, the choice $L = N$ corresponds to a batch algorithm. The train and test compression rates obtained with SNB classifier [Boullé, 2007a] serve as a baseline. The obtained results indicate that, the larger the mini-batches size is, the better is the optimization quality. Moreover, the results obtained for $L = 1000$ and $L = N$ are very similar. The train compression rate is significantly better for batch mode than for $L = 1000$ for 8 of the 16 datasets with $N > 1000$.
Figure 1.2 presents as an example the serie of the criterion values obtained during optimization according to the mini-batches size $L = 100, 1000, N$ for the Phoneme dataset. For all the 36 datasets, when the mini-batches size decreases, the convergence is faster but more chaotic .

We have compared the optimization quality for the PGDMB algorithm without and with metaheuristic. Two metaheuristics have been tested: multi-start (PGDMB-MS) and variable neighborhood search (PGDMB-VNS).
To get computational complexity equivalent to that of the univariate MODL pretreatment, that is to say $O(K * N * \log(K * N))$, we have fixed the total

number of authorized iterations T proportional to $\log(K * N)$. More precisely, we have chosen $T = \log(K * N) * 2^{\text{OptiLevel}}$ where OptiLevel is an integer which enables us to tune the desired optimization level.
For each of the two metaheuristics, we have studied the influence of the optimization level $\text{OptiLevel} = 3, 4, 5$. Since the obtained algorithm stores the best solution each time it is encountered, the metaheuristic can only improve the train compression rate. We have measured in a first step if the improvement was significant or not. For a MS metaheuristic, the train compression rate is significantly improved for resp. $7, 16, 18$ of the 36 datasets with an optimization level equal resp. to $3, 4, 5$. For a VNS metaheuristic, the train compression rate is significantly improved for resp. $18, 19, 23$ of the 36 datasets with an optimization level equal resp. to $3, 4, 5$. The VNS metaheuristic seems then better than the MS metaheuristic: the guided exploration within a variable sized neighborhood from the best minimum encountered enables a more fruitful exploration than a purely random exploration.
Figure 1.3 illustrates this iterations "waste" phenomenon with multi-start

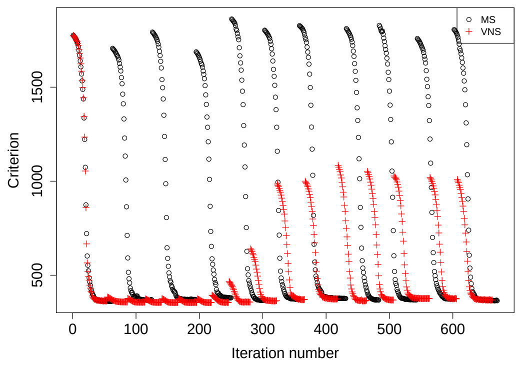


Fig. 1.3 Criterion convergence paths according to the metaheuristic used for the Phoneme dataset and an optimization level equal to 5.

at the beginning of each start.

Experiments presented in this section have illustrated the effect of the mini-batches size on the optimization quality. They have also illustrated that the higher the size, the better the optimization quality, and that the VNS metaheuristic works better than the MS metaheuristic. For the rest of the experiments, we then retain a PGDMB-VNS algorithm with mini-batches size fixed to $L = 1000$ and an optimization level set to $\text{OptiLevel} = 5$.

1.4.2 Regularized classifier performance

We present the classifier performance according to the setting of the regularization weight λ and to the p exponent of the regularization function $|w_k|^p$. Three values have been tested for λ, $0.01, 0.1, 0.5$ and three values for p, $0.5, 1, 2$. The AUC performance for the nine regularized classifiers are presented in Figure 1.4. The performance of the non-regularized classifier obtained with $\lambda = 0$ and of the SNB classifier have been added as a baseline.

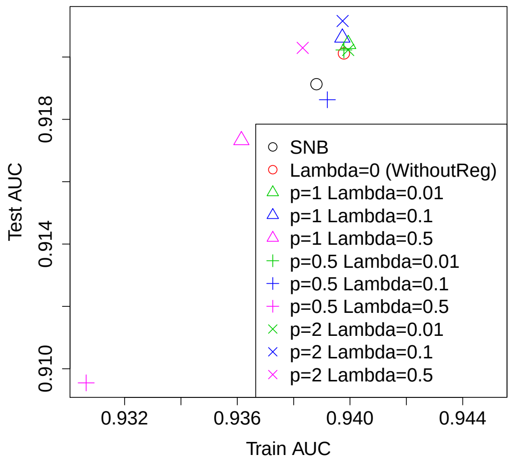
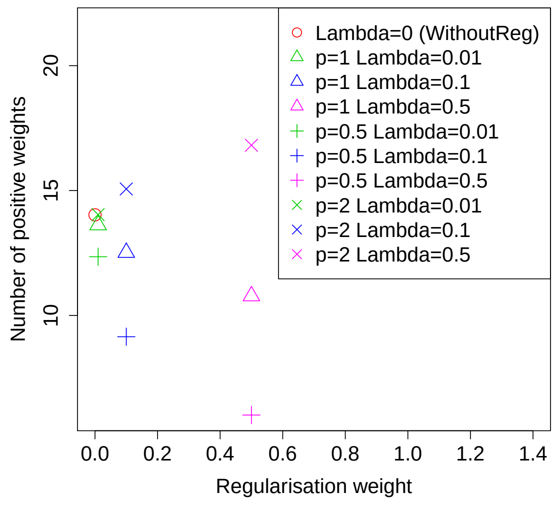


Fig. 1.4 Train and test AUC averaged for 36 UCI datasets according to the weight and the type of regularization.

For the highest regularization weight, $\lambda = 0.5$ (in purple in the Figure), the AUC performance are deteriorated with regards to the performance obtained without regularization (red circles in the Figure) whatever the p value. For the other weight values $\lambda = 0.01$ and $\lambda = 0.1$, the performance are similar for all p values and slightly greater or equal on average to those of the non-regularized classifier. These two regularization weights lead to statistical performance equivalent to those of non regularized classifier.

Figure 1.5 presents a study on the sparsity of the obtained classifiers. In this Figure, the number of kept variables and their weights sum are presented according to the weight and type of regularization. First, it shows that the smaller p, the smaller the non-null weights number. The quadratic regularization ($p = 2$) leads to non sparse classifiers. Among the regularization with



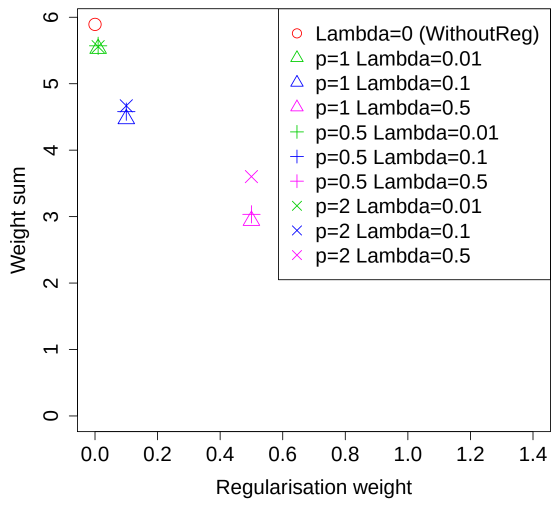


Fig. 1.5 Kept variable number and weight sum averaged for 36 UCI datasets according to the weight and the type of regularization

absolute value ($p = 1$) and the squared root one ($p = 0.5$), the second one enables the most important reduction of the number of kept variables.
As far as the weights sum is concerned, all the p regularization exponents enable to reduce the weights sum on average. Moreover, given a λ weight, the quadratic regularization has a less important impact on the weights sum reduction than the two other regularizations whose performance are very close for this indicator.
Considering both aspects of statistical performance and classifier sparsity, the compromise $p = 1$ and $\lambda = 0.1$ seems the most favorable. Without deteriorating the performance of the non regularized classifier, it enables a significant reduction of the number of selected variables. This reduction makes the classifier more interpretable and less complex to deploy.

1.5 Conclusion

We have proposed a sparse regularization of the log-likelihood for a weighted naïve Bayes classifier. We described and experimented a gradient descent algorithm, which treats online mini-batches data and optimizes the weights classifier through a more or less extensive exploration according to the iterations budget. The experiments have shown the interest of using mini-batches and a metaheuristic for deeper optimization. Moreover, a parameterization study of the regularization points out that the optimal choice was a regularization term with the L_1 norm and a weight $\lambda = 0.1$. Experiments on substantially larger datasets are necessary to evaluate the performance of our approach on real data streams and will be the subject of future work. Furthermore, we will also consider the possibility of solving our optimization problem in two steps : a first convex optimization step using convex approximation of the complete criterion and a second non convex optimization step that could be solved by the method of Composite Minimization [Nesterov, 2013].

References

[Bach and Moulines, 2013] Bach, F. and Moulines, E. (2013). Non-strongly-convex smooth stochastic approximation with convergence rate O(1/n). In *Neural Information Processing Systems (NIPS)*, pages –, United States.

[Bertsekas, 1976] Bertsekas, D. P. (1976). On the goldstein-levitin-polyak gradient projection method. *Automatic Control, IEEE Transactions on*, 21(2):174–184.

[Boullé, 2006] Boullé, M. (2006). Regularization and averaging of the selective naive bayes classifier. In *International Joint Conference on Neural Network Proceedings*, pages 1680–1688.

[Boullé, 2007a] Boullé, M. (2007a). Compression-based averaging of selective naive bayes classifiers. *Journal of Machine Learning Research*, 8:1659–1685.

[Boullé, 2007b] Boullé, M. (2007b). *Recherche d'une représentation des données efficace pour la fouille des grandes bases de données*. PhD thesis, Ecole Nationale Supérieure des Télécommunications.

[Dekel et al., 2012] Dekel, O., Gilad-Bachrach, R., Shamir, O., and Xiao, L. (2012). Optimal distributed online prediction using mini-batches. *J. Mach. Learn. Res.*, 13(1):165–202.

[Gama, 2010] Gama, J. (2010). *Knowledge Discovery from Data Streams*. Chapman & Hall/CRC, 1st edition.

[Godec et al., 2010] Godec, M., Leistner, C., Saffari, A., and Bischof, H. (2010). On-line random naive bayes for tracking. In *International Conference on Pattern Recognition (ICPR)*, pages 3545–3548. IEEE Computer Society.

[Guigourès and Boullé, 2011] Guigourès, R. and Boullé, M. (2011). Optimisation directe des poids de modèles dans un prédicteur bayésien naif moyenné. In *13èmes Journées Francophones "Extraction et Gestion de Connaissances" (EGC 2011)*, pages 77–82.

[Hand and Yu, 2001] Hand, D. J. and Yu, K. (2001). Idiot's Bayes—Not So Stupid After All? *International Statistical Review*, 69(3):385–398.

[Hansen and Mladenovic, 2001] Hansen, P. and Mladenovic, N. (2001). Variable neighborhood search: Principles and applications. *European Journal of Operational Research*, 130(3):449–467.

[Hoerl and Kennard, 1970] Hoerl, A. E. and Kennard, R. W. (1970). Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12:55–67.

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

[Koller and Sahami, 1996] Koller, D. and Sahami, M. (1996). Toward optimal feature selection. In *International Conference on Machine Learning*, pages 284–292.

[Kuncheva and Rodríguez, 2007] Kuncheva, L. I. and Rodríguez, J. J. (2007). Classifier ensembles with a random linear oracle. *IEEE Trans. Knowl. Data Eng.*

[Lange, 2004] Lange, K. (2004). *Optimization*. Springer Texts in Statistics. Springer.

[Langley et al., 1992] Langley, P., Iba, W., and Thompson, K. (1992). An analysis of bayesian classifiers. In *National Conference on Artificial Intelligence*, pages 223–228.

[Nesterov, 2004] Nesterov, Y. (2004). *Introductory lectures on convex optimization : a basic course*. Applied optimization. Kluwer Academic Publ., Boston, Dordrecht, London.

[Nesterov, 2013] Nesterov, Y. (2013). Gradient methods for minimizing composite functions. *Mathematical Programming*, 140(1):125–161.

[Pilanci et al., 2015] Pilanci, M., Wainwright, M. J., and Ghaoui, L. (2015). Sparse learning via boolean relaxations. *Math. Program.*, 151(1):63–87.

[Riedmiller and Braun, 1993] Riedmiller, M. and Braun, H. (1993). A direct adaptive method for faster backpropagation learning: The RPROP algorithm. In *IEEE International Conference On Neural Networks*, pages 586–591.

[Trevor et al., 2015] Trevor, H., Robert, T., and Martin, W. (2015). *Statistical Learning with Sparsity: The Lasso and Generalizations*. Chapman and Hall/CRC.