Fast Binary Feature Selection with
Conditional Mutual Information

François Fleuret   FRANCOIS.FLEURET@EPFL.CH
EPFL – CVLAB
Station 14
CH-1015 Lausanne
Switzerland

Editor: Isabelle Guyon

Abstract

We propose in this paper a very fast feature selection technique based on conditional mutual information. By picking features which maximize their mutual information with the class to predict conditional to any feature already picked, it ensures the selection of features which are both individually informative and two-by-two weakly dependant. We show that this feature selection method outperforms other classical algorithms, and that a naive Bayesian classifier built with features selected that way achieves error rates similar to those of state-of-the-art methods such as boosting or SVMs. The implementation we propose selects 50 features among $40,000$, based on a training set of 500 examples in a tenth of a second on a standard 1Ghz PC.

Keywords: classification, mutual information, feature selection, naive Bayes, information theory, fast learning

1. Introduction

By reducing the number of features, one can both reduce overfitting of learning methods, and increase the computation speed of prediction (Guyon and Elisseeff, 2003). We focus in this paper on the selection of a few tens of binary features among a several tens of thousands in a context of classification.

Feature selection methods can be classified into two types, *filters* and *wrappers* (Kohavi and John, 1997; Das, 2001). The first kind are classifier agnostic, as they are not dedicated to a specific type of classification method. On the contrary the *wrappers* rely on the performance of one type of classifier to evaluate the quality of a set of features. Our main interest in this paper is to design an efficient *filter*, both from a statistical and from a computational point of view.

The most standard filters rank features according to their individual predictive power, which can be estimated by various means such as Fisher score (Furey et al., 2000), Kolmogorov-Smirnov test, Pearson correlation (Miyahara and Pazzani, 2000) or mutual information (Battiti, 1994; Bonnlander and Weigend, 1996; Torkkola, 2003). Selection based on such a ranking does not ensure weak dependency among features, and can lead to redundant and thus less informative selected families. To catch dependencies between features, a criterion based on decision trees has been proposed recently (Ratanamahatana and Gunopulos, 2003). Features which appear in binary trees build with the standard C4.5 algorithm are likely to be either individually informative (those at the top) or con-

ditionally informative (deeper in the trees). The drawbacks of such a method are its computational cost and sensitivity to overfitting.

Our approach iteratively picks features which maximize their mutual information with the class to predict, conditionally to the response of any feature already picked (Vidal-Naquet and Ullman, 2003; Fleuret, 2003). This Conditional Mutual Information Maximization criterion (CMIM) does not select a feature similar to already picked ones, even if it is individually powerful, as it does not carry additional information about the class to predict. Thus, this criterion ensures a good tradeoff between independence and discrimination. A very similar solution called Fast Correlation-Based Filter (Yu and Liu, 2003) selects features which are highly correlated with the class to predict if they are less correlated to any feature already selected. This criterion is very closed to ours but does not rely on a unique cost function which includes both aspects (i.e. information about the class and independence between features) and may be tricked in situation where the dependence between feature appears only conditionally on the object class. It also requires the tuning of a threshold δ for feature acceptance, while our algorithm does not.

Experiments demonstrate that CMIM outperforms the other feature selection methods we have implemented. Results also show and that a naive Bayesian classifier (Duda and Hart, 1973; Langley et al., 1992) based on features chosen with our criterion achieves error rates similar or lower than AdaBoost (Freund and Schapire, 1996a) or SVMs (Boser et al., 1992; Vapnik, 1998; Christiani and Shawe-Taylor, 2000). Also, experiments show the robustness of this method when challenged by noisy training sets. In such a context, it actually achieves better results than regularized AdaBoost, even though it does not require the tuning of any regularization parameter beside the number of features itself.

We also propose in this paper a fast but exact implementation based on a lazy evaluation of feature scores during the selection process. This implementation divides the computational time by two orders of magnitude and leads to a learning scheme which takes for instance one tenth of a second to select 50 out of $43,904$ features, based on 500 training examples. It is available under the GNU General Public Licence at `http://diwww.epfl.ch/~fleuret/files/cmim-1.0.tgz`.

In §2 we introduce the notation, summarize basic concepts of information theory, and present several feature selection schemes, including ours. In §3 we describe how features can be combined with standard classification techniques (perceptron, naive Bayesian, nearest neighbors and SVM with a Gaussian kernel). We propose a very fast implementation of our algorithm in §4 and give experimental results in §5 and §6. We finally analyze those results in §7 and §8.

2. Feature Selection

Our experiments are based on two tasks. The first one is a standard pattern recognition problem and consists of classifying small grey-scale pictures as *face* or *non face*, given a large number of elementary boolean edge-like features. The second one is a drug-design problem and consists of predicting the bio-activity of unknown molecules, given a large number of three-dimensional boolean properties. For clarity, we relate our notation to those two problems, but our approach is generic.

2.1 Notation

Denote by Y a boolean random variable standing for the real class of the object to classify. The value 1 (respectively 0) stands for *face* in the image classification experiments and for *active molecule* in

the drug design task (respectively for *non-face* and *inactive molecule*). Let $X_1, \ldots, X_N$ denote the N boolean features. They depend on the presence or absence of edges at certain positions in the image classification task and to certain three-dimensional properties of the molecule in the drug-design task (see §5.1.2 and §5.2).

The total number of features N is of the order of a few thousands. We denote by $X_{v(1)}, \ldots, X_{v(K)}$ the K features selected during the feature selection process. The number K of such features is very small compared to N, of the order of a few tens.

All statistical estimation is based on a few hundreds of samples, labeled by hand with their real classes and denoted

$$\mathfrak{L} = \{(x^{(1)}, y^{(1)}), \ldots, (x^{(T)}, y^{(T)})\}.$$

See §5.1 and §5.2 for a precise description of those sets. Each $x^{(t)} \in \{0, 1\}^N$ is the boolean vector of feature responses on the tth training example. Hence, $x_n^{(t)}$ is the response of the nth feature on the sample t, and $x_n^{(1)}, \ldots, x_n^{(T)}$ are independent and identically distributed realizations of X_n. We denote $x_n \in \{0, 1\}^T$ this vector of values of feature n on the training samples.

We will use those examples implicitly for all the empirical estimation during training.

2.2 Information Theory Tools

Information theory provides intuitive tools to quantify the uncertainty of random quantities, or how much information is shared by a few of them (Cover and Thomas, 1991). We consider in this section finite random variables and we denote by U, V and W The three of them.

The most fundamental concept in information theory is the entropy $H(U)$ of a random variable, which quantifies the uncertainty of U. The conditional entropy $H(U|V)$ quantifies the remaining uncertainty of U, when V is known. For instance, if U is a deterministic function of V, then this conditional entropy is zero, as no more information is required to describe U when V is known. On the contrary, if they are independent, knowing V does not tell you anything about U and the conditional entropy is equal to the entropy itself.

Our feature selection is based on the conditional mutual information

$$I(U\,;V\,|W) = H(U\,|W) - H(U\,|W, V).$$

This value is an estimate of the quantity of information shared between U and V when W is known. It can also be seen, as shown above, as the difference between the average remaining uncertainty of U when W is known and the same uncertainty when both W and V are known. If V and W carry the same information about U, the two terms on the right are equal, and the conditional mutual information is zero, even if both V and W are individually informative. On the contrary if V brings information about U which is not already contained in W the difference is large.

2.3 Conditional Mutual Information Maximization

The main goal of feature selection is to select a small subset of features that carries as much information as possible. The ultimate goal would be to choose $v(1), \ldots, v(K)$ which minimize $\hat{H}\left(Y\,|X_{v(1)}, \ldots, X_{v(K)}\right)$. But this expression can not be estimated with a training set of realistic size

as it requires the estimation of 2^{K+1} probabilities. Furthermore, even if there were ways to have an estimation, its minimization would be computationally intractable.

At the other extreme, one could do a trivial random sampling which would ensure to some extent independence between features (if different types of features are equally represented) but would not account for predictive power. This could be dealt with by basing the choice on an estimate of this predictive power. The main weakness of this approach is that although it takes care of individual performance, it does not avoid at all redundancy among the selected features. One would pick many similar features, as the ones carrying a lot of information are likely to be of a certain type. For face detection with edge-like features for instance, edges on the eyebrows and the mouth would be the only ones competitive as they are more face-specific than any other edge, yet numerous enough.

We propose an intermediate solution. Our approach deals with the tradeoff between individual power and independence by comparing each new feature with the ones already picked. We say that a feature X' is good only if $\hat{I}(Y ; X'|X)$ is large for *every* X already picked. This means that X' is good only if it carries information about Y, and if this information has not been caught by any of the X already picked. More formally, we propose the following iterative scheme

$$\nu(1) \quad = \quad \arg\max_n \hat{I}(Y ; X_n) \tag{1}$$

$$\forall k,\ 1 \leq k < K, \quad \nu(k+1) \quad = \quad \arg\max_n \underbrace{\left\{ \min_{l \leq k} \hat{I}\left(Y ; X_n | X_{\nu(l)}\right) \right\}}_{s(n,k)}. \tag{2}$$

As said before, $\hat{I}\left(Y ; X_n | X_{\nu(l)}\right)$ is low if either X_n does not bring information about Y or if this information was already caught by $X_{\nu(l)}$. Hence, the score $s(n, k)$ is low if at least one of the features already picked is similar to X_n (or if X_n is not informative at all).

By taking the feature X_n with the maximum score $s(n, k)$ we ensure that the new feature is both informative and different than the preceding ones, at least in term of predicting Y.

The computation of those scores can be done accurately as each score $I(Y ; X_n | X_{\nu(l)})$ requires only estimating the distribution of triplets of boolean variables. Despite its apparent cost this algorithm can be implemented in a very efficient way. We will come back in details to such an implementation in §4.

Note that this criterion is equivalent to maximizing $I(X, X_{\nu(k)} ; Y) - I(X_{\nu(k)} ; Y)$, which is proposed in (Vidal-Naquet and Ullman, 2003).

2.4 Theoretical Motivation

In Koller and Sahami (1996) the authors propose using the concept of the Markov blanket to characterize features that can be removed without hurting the classification performance. A subfamily of features M is a blanket for a feature X_i if X_i is conditionally independent of the other feature and the class to predict given M. However, as the authors point out, such a criterion is stronger than what is really required which is the conditional independence between X_i and Y given M.

CMIM is a forward-selection of features based on an approximation of that criterion. This approximation considers families M composed of a unique feature already picked. Thus, a feature X can be discarded if there is one feature X_ν already picked such that X and Y are conditionally independent given X_ν. This can be expressed as $\exists k,\ I(Y ; X | X_{\nu(k)}) = 0$. Since the mutual information is positive, this can be re-written

$$\min_{k} I(Y \, ; X \, | \, X_{\nu(k)}) \quad = \quad 0.$$

Conversely, the higher this value, the more X is relevant. A natural criterion consists of ranking the remaining features according to that quantity, and to pick the one with the highest value.

2.5 Other Feature Selection Methods

This section lists the various feature selection methods we have used for comparison in our experiments.

2.5.1 RANDOM SAMPLING

The most trivial form of feature selection consist of a uniform random subsampling without repetition. Such an approach leads to features as independent as the original but does not pick the informative ones. This leads to poor results when only a small fraction of the features actually provide information about the class to predict.

2.5.2 MUTUAL INFORMATION MAXIMIZATION

To avoid the main weakness of the random sampling described above, we have also implemented a method which picks the K features $\nu(1), \ldots, \nu(K)$ maximizing individually the mutual information $\hat{I}\left(Y \, ; X_{\nu(l)}\right)$ with the class to predict. Selection based on such a ranking does not ensure weak dependency among features, and can lead to redundant and poorly informative families of features.

In the following sections, we call this method MIM for Mutual Information Maximization.

2.5.3 C4.5 BINARY TREES

As proposed by Ratanamahatana and Gunopulos (2003), binary decision trees can be used for feature selection. The idea is to grow several binary trees and to rank features according to the number of times they appear in the top nodes. This technique is proposed in the literature as a good filter for naive Bayesian classifiers, and is a good example of a scheme able to spot statistical dependencies between more than two features, since the choice of a feature in a binary tree depends on the statistical behavior conditionally on the values of the ones picked above.

Efficiency was increased on our specific task by using randomization (Amit et al., 1997) which consist of using random subsets of the features instead of random subsets of training examples as in bagging (Breiman, 1999, 1996).

We have built 50 trees, each with one half of the features selected at random, and collected the features in the first five layers. Several configurations of number of trees, proportions of features and proportions of training examples were compared and the best one kept. This method is called "C4.5 feature selection" in the result sections.

2.5.4 FAST CORRELATION-BASED FILTER

Th FCBF method addresses explicitly the correlation between features. It first ranks the features according to their mutual information with the class to predict, and remove those which mutual information is lesser than a threshold δ.

In a second step, it iteratively removes any feature X_i if there exist a feature X_j such that $I(Y;X_j) \geq I(Y;X_i)$ and $I(X_i;X_j) \geq I(X_i;Y)$, i.e. X_j is better as a predictor of Y and X_i is more similar to X_j than to Y. The threshold δ can be adapted to get the expected number of features.

2.5.5 ADABOOST

A last method consists of keeping the features selected during boosting and is described precisely in §3.4, page 1538.

3. Classifiers

To evaluate the efficiency of the CMIM feature selection method, we compare the error rates of classifiers based on the features it selects to the error rates with the same classifiers build on features selected by other techniques.

We have implemented several classical type of classifiers, two linear (perceptron and naive Bayesian) and two non-linear (k-NN and SVM with a Gaussian kernel), to test the generality of CMIM. We also implemented a boosting method which avoids the feature selection process since it selects the features and combine them into a linear classifier simultaneously. This technique provides a baseline for classification score.

In this section, recall that we denote by $X_{v(1)}, \ldots, X_{v(K)}$ the selected features.

3.1 Linear Classifiers

A linear classifier depends on the sign of a function of the form

$$f(x_1, \ldots, x_N) \;=\; \sum_{k=1}^{K} \omega_k x_{v(k)} \,+\, b.$$

We have used two algorithms to estimate the $(\omega_1, \ldots, \omega_K)$ and b from the training set $\mathcal{L}$. The first one is the classical perceptron (Rosenblatt, 1958; Novikoff, 1962) and the second one is the naive Bayesian classifier (Duda and Hart, 1973; Langley et al., 1992).

3.1.1 PERCEPTRON

The perceptron learning scheme estimates iteratively the normal vector $(\omega_1, \ldots, \omega_K)$ by correcting it as long as training examples are misclassified. More precisely, as long as there exists a misclassified example, its feature vector is added to the normal vector if it is of class positive, and is otherwise subtracted. The bias term b is computed by considering a constant feature always "positive". If the training set is linearly separable in the feature space, the process is known to converge to a separating hyperplane and the number of iterations can be easily bounded (Christiani and Shawe-Taylor, 2000, page. 12–14). If the data are not linearly separable, the process is terminated after an a priori fixed number of iterations.

Compared to linear SVM for instance, the perceptron has a greater algorithmic simplicity, and suffers from different weaknesses (over-fitting in particular). It is an interesting candidate to estimate the quality of the feature selection methods as a way to control overfitting.

3.1.2 NAIVE BAYESIAN

The naive Bayesian classifier is a simple likelihood ratio test with an assumption of conditional independence among the features. The predicted class depends on the sign of

$$f(x_1, \ldots, x_N) = \log \frac{\hat{P}(Y = 1 \,|\, X_{\mathsf{v}(1)} = x_{\mathsf{v}(1)}, \ldots, X_{\mathsf{v}(K)} = x_{\mathsf{v}(K)})}{\hat{P}(Y = 0 \,|\, X_{\mathsf{v}(1)} = x_{\mathsf{v}(1)}, \ldots, X_{\mathsf{v}(K)} = x_{\mathsf{v}(K)})}.$$

Under the assumption that the $X_{\mathsf{v}(.)}$ are conditionally independent, given Y, and with $a = \log \frac{\hat{P}(Y=1)}{\hat{P}(Y=0)}$, we have

$$
\begin{aligned}
f(x_1, \ldots, x_N) &= \log \frac{\prod_{k=1}^{K} \hat{P}(X_{\mathsf{v}(k)} = x_{\mathsf{v}(k)} \,|\, Y = 1)}{\prod_{k=1}^{K} \hat{P}(X_{\mathsf{v}(k)} = x_{\mathsf{v}(k)} \,|\, Y = 0)} + a \\
&= \sum_{k=1}^{K} \log \frac{\hat{P}(X_{\mathsf{v}(k)} = x_{\mathsf{v}(k)} \,|\, Y = 1)}{\hat{P}(X_{\mathsf{v}(k)} = x_{\mathsf{v}(k)} \,|\, Y = 0)} + a \\
&= \sum_{k=1}^{K} \left\{ \log \frac{\hat{P}(X_{\mathsf{v}(k)} = 1 \,|\, Y = 1)}{\hat{P}(X_{\mathsf{v}(k)} = 1 \,|\, Y = 0)} \frac{\hat{P}(X_{\mathsf{v}(k)} = 0 \,|\, Y = 0)}{\hat{P}(X_{\mathsf{v}(k)} = 0 \,|\, Y = 1)} \right\} x_{\mathsf{v}(k)} + b.
\end{aligned}
$$

Thus we finally obtain a simple expression for the coefficients

$$\omega_k = \log \frac{\hat{P}(X_{\mathsf{v}(k)} = 1 \,|\, Y = 1)}{\hat{P}(X_{\mathsf{v}(k)} = 1 \,|\, Y = 0)} \frac{\hat{P}(X_{\mathsf{v}(k)} = 0 \,|\, Y = 0)}{\hat{P}(X_{\mathsf{v}(k)} = 0 \,|\, Y = 1)}.$$

The bias b can be estimated empirically given the ω_k to minimize the error rate on the training set.

3.2 Nearest Neighbors

We used the Nearest-Neighbors as a first non-linear classifier. Given a regularization parameter k and an example x, the k-NN technique considers the k training examples closest to x according to their distance in the feature space $\{0, 1\}^K$ (either L^1 or L^2, which are equivalent in that case), and gives as predicted class the dominant real class among those k examples. To deal with highly unbalanced population, such as in drug-design series of experiments, we have introduced a second parameter $\alpha \in [0, 1]$ used to weight the positive examples.

Both k and α are optimized by cross-validation during training.

3.3 SVM

As a second non-linear technique, we have used a SVM (Boser et al., 1992; Vapnik, 1998; Christiani and Shawe-Taylor, 2000) based on a Gaussian kernel. This technique is known to be robust to overfitting and has demonstrated excellent behavior on a very large spectrum of problems. The unbalanced population is dealt with by changing the soft-margin parameters C accordingly to the number of training samples of each class, and the choice of the σ parameter of the kernel is described in §6.1.

3.4 AdaBoost

The idea of boosting is to select and combine several classifiers (often referred to as *weak learners*, as they may have individually high error rate) into an accurate one with a voting procedure. In our case, the finite set of features is considered as the space of weak learners itself (i.e. each feature is considered as a boolean predictor). Thus, the training of a weak learner simply consist of picking the one with the minimum error rate. We allow negative weights, which is equivalent to adding for any feature X_n its anti-feature $1 - X_n$. Note that this classifier is not combined with a feature selection methods since it does both the feature selection and the estimation of the weights to combine them simultaneously.

The process maintains a distribution on the training examples which concentrates on the misclassified ones during training. At iteration k, the feature $X_{v(k)}$ which minimizes the weighted error rate is selected, and the distribution is refreshed to increase the weight of the misclassified samples and reduce the importance of the others. Boosting can be seen as a functional gradient descent (Breiman, 2000; Mason et al., 2000; Friedman et al., 2000) in which each added weak learner is a step in the space of classifiers. From that perspective, the weight of a given sample is proportional to the derivative of the functional to minimize with respect to the response of the result classifier on that sample: the more a correct prediction on that particular example helps to optimize the functional, the higher its weight.

In our comparisons, we have used the original AdaBoost procedure (Freund and Schapire, 1996a,b), which is known to suffer from overfitting. For noisy tasks, we have chosen a soft-margin version called AdaBoost$_{reg}$ (Ratsch et al., 1998), which regularizes the classical AdaBoost by penalizing samples which too heavily influence the training, as they are usually outliers.

To use boosting as a feature selector, we just keep the set of selected features $X_{v(1)}, \ldots, X_{v(K)}$, and combine them with another classification rule instead of aggregating them linearly with the weights $\omega_1, \ldots, \omega_K$ computed during the boosting process.

4. CMIM Implementations

We describe in this section how we compute efficiently entropy and mutual information and give both a naive and an efficient implementation of CMIM.

4.1 Mutual Information Estimation

For the clarity of the algorithmic description that follows, we describe here how mutual information and conditional mutual information are estimated from the training set. Recall that as said in §2.1 for any $1 \leq n \leq N$ we denote by $x_n \in \{0, 1\}^T$ the vector of responses of the nth feature on the T training samples.

The estimation of the conditional entropy, mutual information, or conditional mutual information can be done by summing and subtracting estimation of entropies of families of one to three variables. Let x, y, and z be three boolean vectors and u, v, and w, three boolean values. We denote by $||.||$ the cardinal of a set and define three counting functions

$$\eta_u(x) = ||\{t : x^{(t)} = u\}||$$
$$\eta_{u,v}(x, y) = ||\{t : x^{(t)} = u, y^{(t)} = v\}||$$

$$\eta_{u,v,w}(x, y, z) = ||\{t : x^{(t)} = u, y^{(t)} = v, z^{(t)} = w\}||.$$

From this, if we define $\forall x, \xi(x) = \frac{x}{T}\log(x)$, with the usual convention $\xi(0) = 0$, we have

$$
\begin{aligned}
\hat{H}(Y) &= \log(T) - \sum_{u \in \{0,1\}} \xi(\eta_u(y)) \\
\hat{H}(Y, X_n) &= \log(T) - \sum_{u,v \in \{0,1\}^2} \xi(\eta_{u,v}(y, x_n)) \\
\hat{H}(Y, X_n, X_m) &= \log(T) - \sum_{u,v,w \in \{0,1\}^3} \xi(\eta_{u,v,w}(y, x_n, x_m)).
\end{aligned}
$$

And by definition, we have

$$
\begin{aligned}
\hat{I}(Y ; X_n) &= \hat{H}(Y) + \hat{H}(X_n) - \hat{H}(Y, X_n) \\
\hat{I}(Y ; X_n | X_m) &= \hat{H}(Y | X_m) - \hat{H}(Y | X_n, X_m) \\
&= \hat{H}(Y, X_m) - \hat{H}(X_m) - \hat{H}(Y, X_n, X_m) + \hat{H}(X_n, X_m).
\end{aligned}
$$

Finally, those computations are based on counting the numbers of occurrences of certain patterns of bits in families of one to three vectors, and evaluations of ξ on integer values between 0 and T. The most expensive operation is the former: the evaluations of the η_u, $\eta_{u,v}$ and $\eta_{u,v,w}$, which can be decomposed into bit counting of conjunctions of binary vectors. The implementation can be optimized by using a look-up table to count the number of bits in couples of bytes and computing the conjunctions by block of 32 bits. Also, note that because the evaluation of ξ is restricted on integers smaller than T, a lookup table can be used for it too. In the pseudo-code, the function `mut_inf(n)` computes $\hat{I}(Y ; X_n)$ and `cond_mut_inf(n, m)` computes $\hat{I}(Y ; X_n | X_m)$.

Note that the naive Bayesian coefficients can be computed very efficiently with the same counting procedures

$$\omega_k = \log \eta_{0,0}(x_{v(k)}, y) + \log \eta_{1,1}(x_{v(k)}, y) - \log \eta_{1,0}(x_{v(k)}, y) - \log \eta_{0,1}(x_{v(k)}, y).$$

4.2 Standard Implementation

The most straight-forward implementation of CMIM keeps a score vector s which contains for every feature X_n, after the choice of $v(k)$, the score $s[n] = \min_{l \le k} \hat{I}\left(Y ; X_n | X_{v(l)}\right)$. This score table is initialized with the values $\hat{I}(Y ; X_n)$.

The algorithm picks at each iteration the feature $v(k)$ with the highest score, and then refreshes every score $s[n]$ by taking the minimum of $s[n]$ and $\hat{I}\left(Y ; X_n | X_{v(k)}\right)$. This implementation is given in pseudo-code on Algorithm 1 and has a cost of $O(K \times N \times T)$.

4.3 Fast Implementation

The most expensive part in the algorithm described above are the $K \times N$ calls to `cond_mut_inf`, each costing $O(T)$ operations. The fast implementation of CMIM relies on the fact that because the

---

Algorithm 1 Simple version of CMIM

for $n = 1 \ldots N$ **do**
  $s[n] \leftarrow \mathtt{mut\_inf}(n)$
for $k = 1 \ldots K$ **do**
  $nu[k] = \arg\max_n s[n]$
  for $n = 1 \ldots N$ **do**
    $s[n] \leftarrow \min\left(s[n], \mathtt{cond\_mut\_inf}(n, nu[k])\right)$

---

score vector can only decrease when the process goes on, bad scores may not need to be refreshed. This implementation does not rely on any approximation and produces the exact same results as the naive implementation described above.

Intuitively, consider a set of features containing several ones almost identical. Picking one of them makes all the other ones of this group useless during the rest of the computation. This can be spotted early because their scores are low, and will remain so because scores can only decrease.

The fast version of CMIM stores for every feature X_n a partial score $ps[n]$, which is the minimum over a few of the conditional mutual informations appearing in the min in equation (2) page 1534. Another vector $m[n]$ contains the index of the last picked feature taken into account in the computation of $ps[n]$. Thus, we have at any moment

$$ps[n] = \min_{l \leq m[n]} \hat{I}\left(Y ; X_n \mid X_{\nu(l)}\right).$$

At every iteration, the algorithm goes through all candidates and update its score only if the best one found so far in that iteration is not better, since scores can only go down when updated. For instance, if the *up-to-date score* of the first feature was 0.02 and the *non-already updated* score of the second feature was 0.005, it is not necessary to update the later, since it can only go down.

The pseudo-code on Algorithm 2 is an implementation of that idea. It goes through all the candidate features, but does not compute the conditional mutual information between a candidate and the class to predict, given the most recently picked features, if the score of that candidate is below the best up-to-date score $s^\star$ found so far in that iteration (see figure 1).

Algorithm 2 Fast version of CMIM

for $n = 1 \ldots N$ **do**
  $ps[n] \leftarrow \mathtt{mut\_inf}(n)$
  $m[n] \leftarrow 0$
for $k = 1 \ldots K$ **do**
  $s^\star \leftarrow 0$
  for $n = 1 \ldots N$ **do**
    while $ps[n] > s^\star$ **and** $m[n] < k-1$ **do**
      $m[n] \leftarrow m[n] + 1$
      $ps[n] \leftarrow \min\left(ps[n], \mathtt{cond\_mut\_inf}(n, nu[m[n]])\right)$
    if $ps[n] > s^\star$ **then**
      $s^\star \leftarrow ps[n]$
      $nu[k] \leftarrow n$

	1	2	3	4	5	6	7
1	6	3	1	5	4	2	5
2	3	5	?	5	3	?	3
3	5	2	?	4	?	?	?
4	4	?	?	6	?	?	?
5	3	?	?	4	?	?	?

| ps[n] | 3 | 2 | 1 | 4 | 3 | 2 | 3 |
| m[n] | 5 | 3 | 1 | 5 | 2 | 1 | 2 |

Figure 1: *The cell in column n and row l in the array contains the value* cond_mut_inf$(n, nu[l])$. *The score of feature X_n at step $k+1$ is the minimum over the k top-cells of column n. While the naive version evaluates the values of all cells in the first k rows, the fast version computes a partial score, which is the minimum over only the first $m[n]$ cells in column n. It does not update a feature score $ps[n]$ if its current value is already below the best score of a column found so far in that iteration.*

5. Experimental Settings

All the experiments have been done with softwares written in C++ on GNU/Linux computers. We have used free software tools (editor, compiler, debugger, word-processors, etc.), mainly from the Free Software Foundation.[1] We have also used the Libsvm[2] for the SVM.

5.1 Image Classification

This task is a classical pattern-recognition problem in which one tries to predict the real class of a picture. The input data are small grayscale patches and the two classes are face and background (non-face) pictures.

5.1.1 TRAINING AND TEST SETS

We have used training and test sets built from two large sets of scenes. Those original big sets were assembled by collecting a few thousand scenes from the web and marking by hand the locations of eyes and mouth on every visible frontal viewed face. Using two sets ensures that examples belonging to the same scene series will all be used either as training pictures or as test pictures. This prevents from trivial similarities between the training and test examples.

From every face of every scene we generate ten small grayscale face images by applying rotation, scaling and translation to randomize its pose in the image plan. We have also collected complex scenes (forests, buildings, furniture, etc.) from which we have automatically extracted tens of thousands of background (non-face) pictures. This leads to a total of $14,268$ faces and $14,800$ background pictures for training (respectively $5,202$ and $5,584$ for test).

All those images, faces and backgrounds, are of size 28×28 pixels, and with 256 grayscale levels. Faces have been registered roughly so that the center of the eyes is in a 2×2 central square, the distance between the eyes is between 10 to 12 pixels and the tilt is between -20 and $+20$ degrees (see figure 2 for a few examples of images).

For each experiment both the training and the test sets contain 500 images, roughly divided into faces and non-faces. Errors are averaged over 25 rounds with such training and test sets.

5.1.2 EDGE FEATURES

We use features similar to the edge fragment detectors in (Fleuret and Geman, 2001, 2002). They are easy to compute, robust to illumination variations, and do not require any tuning.

Each feature is a boolean function indexed by a location (x, y) in the 28×28 reference frame of the image, a direction d which can take 8 different values (see figures 3 and 4) and a tolerance t which is an integer value between 1 and 7 (this maximum value has been fixed empirically). The tolerance corresponds to the size of the neighborhood where the edge can "float", i.e. where it has to be present for the feature to be equal to 1 (see figure 3).

For every location in the reference frame (i.e. every pixel), we thus have 7×8 features, one for each couple direction / tolerance. For tolerance 1, those features are simple edge fragment detector (see figure 4). For tolerance 2, they are disjunction of those edge fragment detectors on two locations for each pixel, etc.

The total number of such features is $28 \times 28 \times 8 \times 7 = 43,904$.

---

1. http://www.fsf.org
2. http://www.csie.ntu.edu.tw/~cjlin/libsvm

Figure 2: The two upper rows show examples of background pictures, and the two lower rows show examples of face pictures. All images are grayscale of size 28×28 pixels, extracted from complete scenes found on the World Wide Web. Faces are roughly centered and standardized in size.
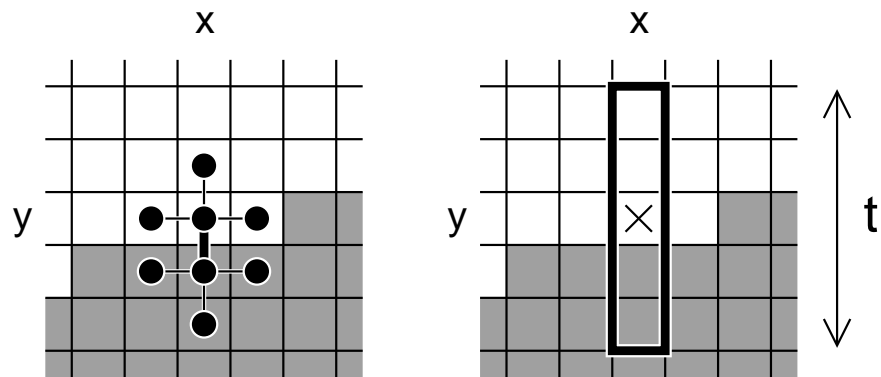

Figure 3: The boolean features we are using are crude edge detectors, invariant to changes in illumination and to small deformations of the image. The picture on the left shows the criterion for a horizontal edge located in (x, y). The detector responds positively if the six differences between pixels connected by a thin segment are lesser in absolute value than the difference between the pixels connected by the thick segment. The relative values of the two pixels connected by the thick line define the polarity of the edge (dark to light or light to dark). The picture on the right shows the strip where the edge can "float" for the feature to respond when the tolerance t is equal to 5.
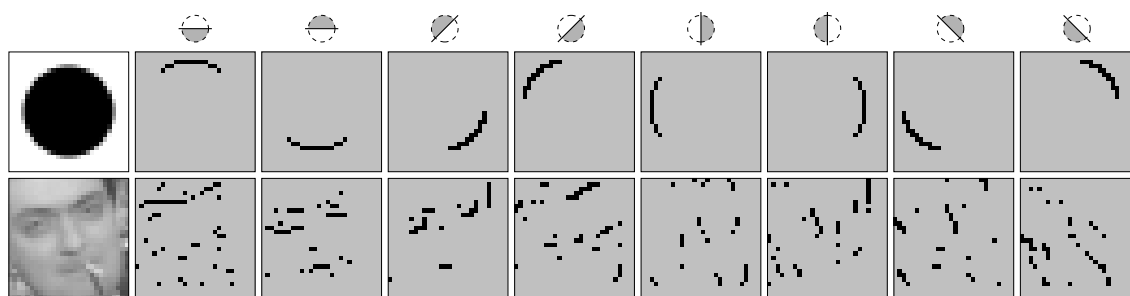
Figure 4: The original grayscale pictures are shown on the left. The eight binary maps on the right show the responses of the edge detectors at every locations in the 28×28 frame, for every one of the 8 possible directions and polarities. The binary features are disjunctions (ORings) of such edge detectors in small neighborhoods, which ensure their robustness to image deformations.

5.2 Prediction of Molecular Bio-activity

The second data set is based on $1,909$ compounds tested for their ability to bind to a target site on thrombin. This corresponds to a drug-design task in which one tries to predict which molecules will achieve the expected effect.

Each compound has a binary class (*active* or *inactive*) and $139,351$ binary features standing for as many three-dimensional properties. The exact semantic of those features remains unknown but is consistent among the samples. To be able to use many techniques in our comparisons, we restricted the number of binary features to $2,500$ by a rough random sampling, since the computation time would have been intractable with classical methods on such a large number of binary features.

All the experiments are done with 25 rounds of cross-validation. For each one of this round, 100 samples are randomly picked as test examples, and all the others used for training. Since the population are highly unbalanced (42 positive vs. 1867 negative examples), the balanced error rate (average of false positive and false negative error rates) was used both for training and testing.

The dataset was provided by DuPont Pharmaceutical for the KDD-Cup 2001 competition[3] and was used in 2003 for the NIPS feature selection challenge[4] under the name DOROTHEA.

6. Results

The experimental results we present in this section address both performance in term of error rates and speed.

In §6.1, we compare several associations of a feature selection method (CMIM, MIM, C4.5, random and AdaBoost as a feature selection method) and a classifier (naive Bayesian, k-NN, Gaussian SVM, perceptron). Also, AdaBoost and a regularized version of AdaBoost were tested on the same data.

For the image recognition task, since the error rates with 50 features correspond roughly to the asymptotic score of our main methods (CMIM + naive Bayesian and AdaBoost, see figure 5), we

---

3. `http://www.cs.wisc.edu/~dpage/kddcup2001/`
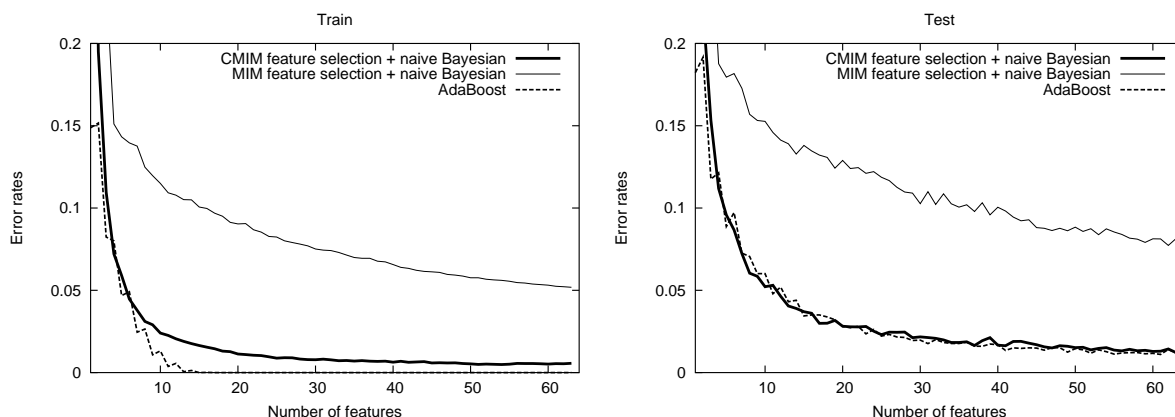4. `http://www.nipsfsc.ecs.soton.ac.uk`

Figure 5: The asymptotic error rates are reached with 50 features on the picture classification task.

have used this number of features for the extensive comparisons. Similarly, the number of features was 10 for the bio-activity prediction.

The σ parameter of the Gaussian kernel was chosen separately for every feature selection method by optimizing the test error with a first series of 25 rounds. The training and test errors reported in the results section are estimated by running 25 other rounds of cross-validation. The results may suffer slightly from over-fitting and over-estimate the score of the SVM. However the effect is likely to be negligible considering the large size of the complete sets.

In §6.2 we compare the fast implementation of CMIM to the naive one and provide experimental computation times in the task of image recognition.

6.1 Error Rates

To quantify the statistical significance in our comparisons, we estimate empirical error rates form the data sets, but also the empirical variance of those estimates. Those variances are computed under the assumption that the samples are independent and identically distributed.

We provide for every experiments in the tables 1, 2 and 3 both the estimated test error e and the score $\frac{e^\star - e}{\sqrt{\sigma_{e^\star} + \sigma_e}}$ where $e^\star$ is the score of our reference setting (CMIM + Naive Bayesian), and $\sigma_{e^\star}$ and σ_e are the empirical variances of the error rate estimates. This empirical variance are estimated simply as empirical variance of Bernoulli variables.

6.1.1 IMAGE CLASSIFICATION

The first round of experiments uses the dataset of pictures described in §5.1. The results on table 1 show that the best scores are obtained with CMIM + SVM, closely followed by AdaBoost features combined also with SVM. The Naive Bayesian with CMIM features performs pretty well, ranking fourth. CMIM as a feature selection method is always the best, for any given classification technique.

It is meaningful to note that the computational cost of SVM is few orders of magnitudes higher than those of AdaBoost alone or CMIM + Bayesian as it requires for training the computation of the optimal σ through cross-validation, and requires during classification the evaluation of hundreds of exponentials.

Classifier	Training error	Test error (e)	$\frac{e^\star-e}{\sqrt{\sigma_{e^\star}+\sigma_e}}$
CMIM + SVM	0.53%	1.12%	−2.77
AdaBoost feature selection + SVM	0%	1.21%	−2.11
AdaBoost	0%	1.45%	−0.45
CMIM feature selection + naive Bayesian	0.52%	1.52%	−
CMIM feature selection + k-NN	0%	1.69%	1.07
AdaBoost feature selection + k-NN	0%	1.71%	1.19
FCBF feature selection + SVM	0.75%	1.85%	2.02
FCBF feature selection + naive Bayesian	1.28%	2.13%	3.60
CMIM feature selection + perceptron	0%	2.28%	4.40
AdaBoost feature selection + perceptron	0%	2.46%	5.32
C4.5 feature selection + SVM	0.73%	2.58%	5.91
FCBF feature selection + k-NN	0%	2.75%	6.73
C4.5 feature selection + perceptron	0%	3.26%	9.02
C4.5 feature selection + naive Bayesian	1.4%	3.28%	9.11
FCBF feature selection + perceptron	0%	3.50%	10.03
AdaBoost feature selection + naive Bayesian	0.4%	3.51%	10.06
C4.5 feature selection + k-NN	0%	3.57%	10.31
MIM + SVM	3.26%	5.67%	17.73
MIM feature selection + perceptron	3.56%	8.28%	25.06
MIM feature selection + naive Bayesian	5.58%	8.54%	25.72
MIM feature selection + k-NN	0.23%	8.99%	26.84
Random feature selection + SVM	9.04%	11.86%	33.44
Random feature selection + perceptron	13.36%	17.45%	44.66
Random feature selection + k-NN	0.30%	21.54%	52.18
Random feature selection + naive Bayesian	21.69%	24.77%	57.93

Table 1: Error rates with 50 features on the accurate face vs. background data set. The right column shows the difference between the test error rate $e^\star$ of the CMIM + naive Bayesian method and the test error rate e in the given row, divided by the standard deviation of that difference.

To test the robustness of the combination of CMIM and naive Bayes, we have run a second round of experiments with noisy training data, known to be problematic for boosting schemes. We generated the new training set by flipping at random 5% of the training labels. This corresponds to a realistic situation in which some training examples have been mis-labelled.

It creates a difficult situation for learning methods which take care of outliers, since there are 5% of them, distributed uniformly among the training population. Results are summarized in table 2. Note that the performance of the regularized version of AdaBoost correspond to the optimal performance on the *test set*.

All methods based on perceptron or boosting have high error rates, since they are very sensitive to outliers. The best classification techniques are those protected from over-fitting, thus SVM, Naive Bayesian and regularized AdaBoost, which take the 8 first rankings. Again in this experiment, CMIM is the best feature selection method for any classification scheme.

The FCBF method, which is related to CMIM since it looks for features both highly correlated with the class to predict and two-by-two uncorrelated scores very well, better than in the non-noisy case. It may be due to the fact that in this noisy situation protection from overfitting matters more than picking optimal features on the training set.

6.1.2 MOLECULAR BIO-ACTIVITY

This third round of experiments is more difficult to analyze since the characteristics of the features we deal with are mainly unknown. Because of the highly unbalanced population, methods sensitive to overfitting perform badly.

Results for these experiments are given 3. Except in one case (SVM), CMIM leads to the lowest error rate for any classification method. Also, when combined with the naive Bayesian rule, it gets lower error rates than SVM or nearest-neighbors.

The same dataset was used in the NIPS 2003 challenge,[5] in which it was divided in three subsets for training, test and validation (respectively of size 800, 350 and 800). Our main method CMIM + Bayesian achieves 12.46% error rate on the validation set without any tunning, while the top-ranking method achieves 5.47% with a Bayesian Network, see (Guyon et al., 2004) for more details on the participants and results.

6.2 Speed

The image classification task requires the selection of 50 features among $43,904$ with a training set of 500 examples. The naive implementation of CMIM takes 18800ms to achieve this selection on a standard 1Ghz personal computer, while with the fast version this duration drops to 255ms for the exact same selection, thus a factor of 73. For the thrombin dataset (selecting 10 features out of $139,351$ based on $1,909$ examples) the computation times drops from $156,545$ms with the naive implementation to $1,401$ms with the fast one, corresponding to a factor 110.

The dramatic gain in performance can be explained by looking at the number of calls to `cond_mut_inf`, which drops for the faces by a factor 80 (from $4,346,496$ to $54,928$), and for the thrombin dataset by a factor 220 (from $13,795,749$ to $62,125$).

The proportion of calls to `cond_mut_inf` for the face dataset is depicted on figure 6. We have also looked at the number of calls required to sort out each feature. In the simple implementation that number is the same for all features and is equal to the number of selected features *K*. For the fast

---

5. http://www.nipsfsc.ecs.soton.ac.uk

Classifier	Training error	Test error (e)	$\frac{e^\star - e}{\sqrt{\sigma_{e^\star} + \sigma_e}}$
CMIM + SVM	5.68%	1.37%	-3.59
FCBF feature selection + SVM	6.02%	1.49%	-2.79
CMIM feature selection + naive Bayesian	5.06%	1.95%	–
FCBF feature selection + naive Bayesian	5.38%	2.39%	2.38
C4.5 + SVM	5.57%	2.99%	5.30
AdaBoost$_{reg}$ (optimized on test set)	3.80%	3.06%	5.61
C4.5 feature selection + naive Bayesian	6.14%	3.62%	8.03
AdaBoost feature selection + SVM	4.39%	4.18%	10.25
CMIM feature selection + k-NN	0.08%	5.36%	14.42
MIM + SVM	7.85%	5.87%	16.07
AdaBoost	0.58%	6.33%	17.48
C4.5 feature selection + k-NN	0.71%	6.34%	17.52
FCBF feature selection + k-NN	0.87%	6.50%	17.99
AdaBoost feature selection + k-NN	0.39%	7.20%	20.02
AdaBoost feature selection + perceptron	0.12%	8.23%	22.82
MIM feature selection + naive Bayesian	9.47%	8.59%	23.75
CMIM feature selection + perceptron	7.36%	9.32%	25.60
FCBF feature selection + naive Bayesian	8.20%	9.33%	25.62
AdaBoost feature selection + naive Bayesian	10.28%	9.46%	25.94
C4.5 feature selection + perceptron	7.58%	11.06%	29.71
Random + SVM	13.00%	12.19%	32.23
MIM feature selection + k-NN	2.92%	11.46%	30.61
MIM feature selection + perceptron	11.53%	13.12%	34.23
Random feature selection + perceptron	19.47%	20.58%	48.75
Random feature selection + k-NN	1.43%	24.77%	56.29
Random feature selection + naive Bayesian	24.13%	24.99%	56.68

Table 2: Error rates with 50 features on the face vs. background data set whose training labels have been flipped with probability 5%. The right column shows the difference between the test error rate $e^\star$ of the CMIM + naive Bayesian method and the test error rate e in the given row, divided by the standard deviation of that difference.

Classifier	Training error	Test error (e)	$\frac{e^\star - e}{\sqrt{\sigma_{e^\star} + \sigma_e}}$
CMIM feature selection + naive Bayesian	10.45%	11.72%	–
AdaBoost feature selection + SVM	9.35%	12.99%	1.36
AdaBoost feature selection + naive Bayesian	10.29%	13.60%	1.99
AdaBoost$_{reg}$ (optimized on test set)	9.48%	13.64%	2.04
CMIM + SVM	13.21%	13.65%	2.05
AdaBoost	9.49%	13.76%	2.16
C4.5 feature selection + naive Bayesian	9.22%	13.90%	2.31
C4.5 + SVM	8.72%	17.34%	5.65
CMIM feature selection + k-NN	17.17%	18.77%	6.97
FCBF feature selection + naive Bayesian	13.62%	19.22%	7.37
FCBF feature selection + SVM	13.39%	23.14%	10.76
MIM feature selection + naive Bayesian	21.53%	23.35%	10.94
CMIM feature selection + perceptron	20.31%	23.51%	11.08
C4.5 feature selection + perceptron	12.86%	23.88%	11.38
MIM + SVM	24.65%	25.75%	12.93
FCBF feature selection + perceptron	21.98%	27.06%	13.98
FCBF feature selection + k-NN	19.28%	27.94%	14.68
Random + SVM	30.10%	30.92%	17.05
C4.5 feature selection + k-NN	24.18%	34.11%	19.54
Random feature selection + naive Bayesian	39.32%	40.13%	24.23
MIM feature selection + perceptron	32.70%	40.27%	24.34
Random feature selection + perceptron	43.61%	45.68%	28.63
Random feature selection + k-NN	45.09%	47.29%	29.94
MIM feature selection + k-NN	50.00%	50.00%	32.19

Table 3: Error rates with 10 features on the Thrombin dataset. The right column shows the difference between the test error rate $e^\star$ of the CMIM + naive Bayesian method and the test error rate e in the given row, divided by the standard deviation of that difference.
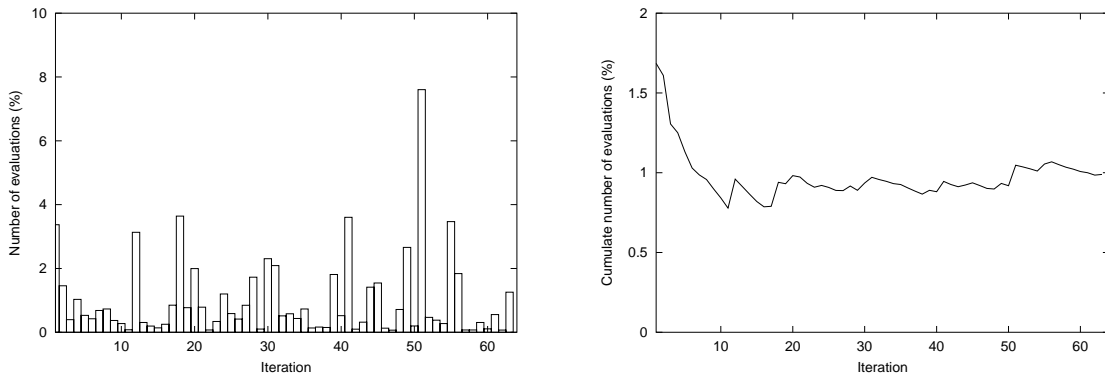
Figure 6: *Those curves show the proportion of calls to `cond_mut_inf` actually done in the fast version compared to the standard version for each iteration. The curve on the left shows the proportion for each step of the selection process, while the curve on the right shows the proportion of cumulate evaluations since the beginning. As it can be seen, this proportion is around 1% on the average.*
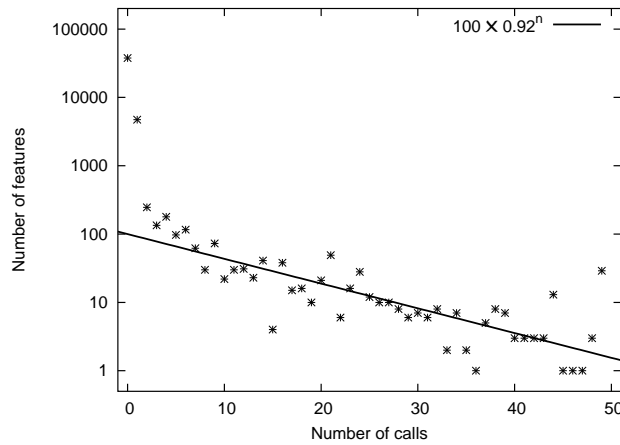

Figure 7: *This curve shows on an logarithmic scale how many features (y axis) require a certain number of calls to `cond_mut_inf` (x axis). The peak on the right corresponds to the 50 features actually selected, which had to be compared with all the other features, thus requiring 49 comparisons.*

version, this number depends on the feature, as very inefficient ones will probably require only one of them. The distribution of the number of evaluations is represented on figure 7 on a logarithmic scale, and fits roughly 100×0.92^n. This means that there are roughly 8% fewer features which require $n + 1$ evaluations than features which require n evaluations.

7. Discussion

The experimental results we provide show the strength of the CMIM, even when combined with a simple naive Bayesian rule, since it ranks 4th, 3rd and 1st in the three experiments in which it is compared with 26 other combination feature selection + classifier.

Classification Power

It is easy to build a task CMIM can not deal with. Consider a situation where the positive population is a mixture of two sub-populations, and where half of the features provide information about the first population, while the other features provide information about the second population. This can happen in an image context by considering two different objects which do not share informative edges.

In such a situation, if one sub-population dominates statistically, CMIM does not pick feature providing information about the second sub-population. It would go on picking feature informative about the domineering sub-population as long as independent features remain.

A feature selection based on C4.5 would be able to catch informative features since the minority class would quickly be revealed as the source of uncertainty, and features dedicated to them would be selected. Similarly, AdaBoost can handle such a challenge because the error concentrates quickly on the second sub-population, which eventually drives the choice of features. In fact, both can be ween as wrappers since they take into account the classification outcome to select the features.

We could fix this weakness of CMIM by weighting challenging examples as well, forcing the algorithm to care about problematic minorities and pick features related to them. This would be the dual solution to AdaBoost regularization techniques which on the contrary reduce the influence of outliers.

From that point of view, CMIM and AdaBoost are examples of two families of learning methods. The first one is able to cope with overfitting by making a strong assumption of homogeneity of the informative power of features, while the second one is able to deal with a composed population by sequentially focusing on sub populations as soon as they become the source of error.

In both cases, the optimal tradeoff has to be specified on a per-problem basis, as there is no absolute way to know if the training examples are reliable examples of a complex mixture or noisy examples of an homogeneous population.

Speed

CMIM and boosting share many similarities from an algorithmic point of view. Both of them pick features one after another and compute at each iteration a score for every single candidate which requires to go through every training example.

Despite this similarity the lazy evaluation idea can not be applied directly to boosting. One could try to estimate a bound of the score of a weak learner at iteration $k + 1$ (which is a weighted error rate), given its value at iteration k, using a bound on the weight variation. Practically, this idea gives very bad results because the variation can not be controlled efficiently and turns out to be very pessimistic. It leads to a negligible rate of rejection of the candidate features to check.

The feature selection based on C4.5 is even more difficult to optimize. Because of the complex interactions between features selected in previous nodes and the remaining candidates at a given

node, there is no simple way to predict that a feature can be ignored without reducing the performance of the method.

Usability

The CMIM algorithm does not require the tuning of any regularizing parameter, and since the implementation is an exact exhaustive search it also avoids the tuning of an optimization scheme.

Also, compared to methods like SVM or AdaBoost, both the feature selection criterion and the naive Bayesian classifier have a very clear statistical semantic. Since the naive Bayesian is an approximation of a likelihood ratio test, it can be easily combined with other techniques such as HMM, Bayesian Inference and more generally with other statistical methods.

Multi-class, Continuous Valued Features and Regression

Extension to the multi-class problem can be addressed either with a classifier-agnostic technique (for instance training several classifiers dedicated to different binary problems which can be combined into a multi-class predictor (Hastie and Tibshirani, 1998) or by extending CMIM and the Bayesian classifier directly to the multi-class context.

In that case, the price to pay is both in term of accuracy and computation cost. The estimation of the conditional mutual information requires the estimation of the empirical distribution of triples of variables, or N_c^3 empirical probabilities in a N_c class problem. Thus, accurate estimation requires as many more training samples. From the implementation perspective the fast version can be kept as-is but the computation of a conditional mutual information is $O(N_c^3)$, and the boolean computations by block require a $O(N_c)$ memory usage.

Extension to the case of continuous valued features and to regression (continuous valued class) is the center of interest of our current works. It is natural as soon as parametric density models are provided for any variable, couple of variables and triplet of variables. For any couple of features X_i, X_j, the estimation of the conditional mutual information given Y requires first an estimation of the model parameter α according to the training data.

The most naive form of multi-variable density would be piece-wise constant, thus discretisation with features of the form $F = 1_{X \geq t}$ where X is one of the original continuous feature. Such a model would lead to the same weakness as those described above for the multi-class situation.

If a more sophisticated model can legitimately be used – for instance multi-dimensional Gaussian – the only difficulty is the computation of the conditional mutual information itself, requiring sums over the space of values of products and ratios of such expressions. Depending on the existence of analytical form of this sum, the algorithm may require numerical integration and heavy computations. Nevertheless, even if the computation of the conditional mutual information is expensive, the lazy evaluation trick presented in §2 can still be used, reducing the cost by the same amount as in the provided results.

8. Conclusion

We have presented a simple and very efficient scheme for feature selection in a context of classification. On the experiments we have done CMIM is the best feature selection method except in one case (SVM for the thrombin experiment). Combined with a naive Bayesian classifier the scores

we obtained are comparable or better than those of state-of-the-art techniques such as boosting or Support Vector Machines, while requiring a training time of a few tenth of a second.

Because of its high speed, this learning method could be used to tune learnt structures on the fly, to adapt them to the specific difficulties of the populations they have to deal with. In the context of face detection, such an on-line training could exploit the specificities of the background population and reduce the false-positive error rate. Also, it could be used in applications requiring the training of a very large number of classifiers. Our current works in object recognition are based on several thousands of classifiers which are built in a few minutes.

References

Y. Amit, D. Geman, and K. Wilder. Joint induction of shape features and tree classifiers. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(11):1300–1305, November 1997.

R. Battiti. Using mutual information for selecting features in supervised neural net learning. In *IEEE Transactions on Neural Networks*, volume 5, 1994.

B. V. Bonnlander and A. S. Weigend. Selecting input variables using mutual information and non-parametric density estimation. In *Proceedings of ISANN*, 1996.

B. E. Boser, I. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. *in Proceedings of the Fifth Annual Workshop on Computational Learning Theory*, 5:144–152, 1992.

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

L. Breiman. Random forests – random features. Technical Report 567, Department of Statistics, University of California, Berkeley, 1999.

L. Breiman. Some infinity theory for predictors ensembles. Technical Report 579, Department of Statistics, University of California, Berkeley, 2000.

N. Christiani and J. Shawe-Taylor. *An Introduction to Support Vector Machines and other kernel-based learning methods*. Cambridge University Press, 2000.

T. M. Cover and J. A. Thomas. *Elements of Information Theory*. Wiley-Interscience, 1991.

S. Das. Filters, wrappers and a boosting-based hybrid for feature selection. In *Proceedings of the International Conference on Machine Learning 2001*, pages 74–81, 2001.

R. Duda and P. Hart. *Pattern classification and scene analysis*. John Wiley & Sons, 1973.

F. Fleuret. Binary feature selection with conditional mutual information. Technical Report RR-4941, INRIA, October 2003.

F. Fleuret and D. Geman. Coarse-to-fine visual selection. *International Journal of Computer Vision*, 41(1/2):85–107, 2001.

F. Fleuret and D. Geman. Fast face detection with precise pose estimation. In *Proceedings of ICPR2002*, volume 1, pages 235–238, 2002.

Y. Freund and R.E. Schapire. Experiments with a new boosting algorithm. In *Proc. 13th International Conference on Machine Learning*, pages 148–156. Morgan Kaufmann, 1996a.

Y. Freund and R.E. Schapire. Game theory, on-line prediction and boosting. In *Proc. 9th Annu. Conf. on Comput. Learning Theory*, pages 325–332. ACM Press, New York, NY, 1996b.

J. H. Friedman, T. Hastie, and R. Tibshirani. Additive logistic regression: a statistical view of boosting. *Annal of Statistics*, 28:337–407, 2000.

T. S. Furey, N. Cristianini, N. Duffy, D. W. Bednarski, M. Schummer, and D. Haussler. Support vector machine classification and validation of cancer tissue samples using microarray expression data. *Bioinformatics*, 16(10), 2000.

I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

I. Guyon, S. Gunn, S. Ben Hur, and G. Dror. Result analysis of the NIPS2003 feature selection challenge. In *Proceedings of the NIPS2004 conference*, 2004.

T. Hastie and R. Tibshirani. Classification by pairwise coupling. In Michael I. Jordan, Michael J. Kearns, and Sara A. Solla, editors, *Advances in Neural Information Processing Systems*, volume 10. The MIT Press, 1998.

R. Kohavi and G. John. Wrappers for feature subset selection. *Artificial Intelligence*, pages 273–324, 1997.

D. Koller and M. Sahami. Toward optimal feature selection. In *Proceedings of the 13th International Conference on Machine Learning 1996*, pages 284–292, 1996.

P. Langley, W. Iba, and K. Thompson. An analysis of bayesian classifiers. In *Proceedings of AAAI-92*, pages 223–228, 1992.

L. Mason, J. Baxter, P. Bartlett, and M. Frean. Boosting algorithms as gradient descent. In *Neural Information Processing Systems*, pages 512–518. MIT Press, 2000.

K. Miyahara and M. J. Pazzani. Collaborative filtering with the simple bayesian classifier. In *Pacific Rim International Conference on Artificial Intelligence*, pages 679–689, 2000.

A. Novikoff. On convergence proofs for perceptrons. In *In Symposium on Mathematical Theory of Automata*, pages 615–622, 1962.

C. A. Ratanamahatana and D. Gunopulos. Feature selection for the naive bayesian classifier using decision trees. *Applied Artificial Intelligence*, 17(5-6):475–487, 2003.

G. Ratsch, T. Onoda, and K.-R. Muller. Regularizing AdaBoost. In *Proceedings of NIPS*, pages 564–570, 1998.

F. Rosenblatt. The perceptron: A probabilistic model for information storage and organization in the brain. *Psychological Review*, 65:386–408, 1958.

K. Torkkola. Feature extraction by non-parametric mutual information maximization. *Journal of Machine Learning Research*, 3:1415–1438, 2003.

V. N. Vapnik. *The Nature of Statistical Learning Theory*. Springer-Verlag, 1998.

M. Vidal-Naquet and S. Ullman. Object recognition with informative features and linear classification. In *Proceedings of International Conference on Computer Vision 2003*, pages 281–288, 2003.

L. Yu and H. Liu. Feature selection for high-dimensional data: A fast correlation-based filter solution. In *Proceedings of the International Conference on Machine Learning 2003*, pages 856–863, 2003.