A Scalable Learning Algorithm for Kernel Probabilistic Classifier

Mathieu Serrurier and Henri Prade

IRIT - 118 route de Narbonne 31062, Toulouse Cedex 9, France
{serrurie,prade}@irit.fr

Abstract. In this paper we propose a probabilistic classification algorithm that learns a set of kernel functions that associate a probability distribution over classes to an input vector. This model is obtained by maximizing a measure over the probability distributions through a local optimization process. This measure focuses on the faithfulness of the whole probability distribution induced rather than only considering the probabilities of the classes separately. We show that, thanks to a pre-processing computation, the complexity of the evaluation of this measure with respect to a model is no longer dependent on the size of the training set. This makes the local optimization of the whole set of kernel functions tractable, even for large databases. We experiment our method on five benchmark datasets and the KDD Cup 2012 dataset.

1 Introduction

It is well known that machine learning algorithms are constrained by some learning bias (language bias, hypothesis bias, algorithm bias, etc.). In that respect, learning a precise model may be illusionary. Moreover, in case where security issues are critical for instance, predicting one class only, without describing the uncertainty about this prediction, may be unsatisfactory. Probabilistic classification aims at learning models that associate to an input vector a probability distribution over classes rather than a single class. K-nearest-neighbor methods [1] compute this distribution by considering the neighborhood of the input vector. Probabilities are then computed from the frequency of the classes. The quality of the distribution highly depends on the density of the data. Some other types of algorithms such as naive Bayes classifiers [7] and Gaussian processes [9,10,15] are based on Bayesian inference. Gaussian processes assume that the attribute values follow a Gaussian distribution, it uses kernels for describing the co-variance between such variables. Thus, these approaches suppose strong assumptions (high density data, independent attributes, priors about the type of the probability distribution that underlies the data , ...). Logistic regression has been also proposed as a probabilistic classifier [2,5] since it can be used for a direct estimation of the probability of the classes. This approach has been extended for the non linear case by the use of kernels logistic functions (KLR [4]) or kernel functions [16]. These last methods are based on minimization of the squared distance between the value of the class (0 or 1) and the predicted value (between 0 and 1). Functions are learn independently for each class and a normalization post processing is needed. Moreover, these approaches are based on a costly optimization processes and are not tractable for large databases. Sugiyama [13] proposes an alternative reformulation of the calculus (still based on the minimization of squared distance) that partially overcomes this cost issue and skips the normalization step. Even if these methods are consistent with the maximum likelihood principle, there are based on the evaluation of the ability of the predicted distribution to identify the most probable class, but not on an evaluation of the faithfulness of the complete probability distribution with respect to the data.

In this paper we propose to learn a set of kernel functions as in this other kernel approaches. However, our method differs from them by many points. First, we constrain the parameters of the kernels in order to have the sum of the probabilities equal to 1. Second, the model is obtained through a local optimization process (we propose an implementation for two algorithms: the Nelder-Mead algorithm [8] and the particle swarm meta-heuristics [6]) by fixing the support vectors and maximizing a quality measure that estimates the faithfulness of a probability distribution with respect to a set of data. The kernel function are learned all together (rather than independently in classical kernel approaches). In this scope we have extended the loss function used in KLR to the whole distribution. Then, this measure relies on the squared distance between the optimal distribution (1 for the class, 0 for the other classes) and the proposed one. Moreover, we reformulate the computation of this measure for our set of kernel functions and make that the complexity of this computation no longer depend on the size of the dataset thanks to a pre-computation step. This allows us to handle very large databases, even when using costly meta-heuristics such as particle swarm.

The paper is structured as follows. Section 2 provides some definitions about probabilistic loss functions and a precise description of the proposed measure. In Section 3, we describe the model that we learn and we show how the measure can be reformulated in order to maximize performances. Section 4 is devoted to the description of the optimization process and the tuning of the parameters. Last, we validate our approach by experimentation on 5 benchmark datasets and on the KDD Cup 2012 dataset.

2 Probabilistic Loss Functions

Probabilistic loss functions are used for evaluating the adequateness of a probability distribution with respect to data. In this paper, we only consider the case of classification. A classification database is a set of n pairs $(\overrightarrow{x}_i, c_i)$, $1 \leq i \leq n$, where $\overrightarrow{x}_i$ is a vector of input variables in the feature space $\mathcal{X}$ and $c_i \in \{C_1, \ldots, C_q\}$ is the class variable. We note

$$\mathbb{1}_j(\overrightarrow{x}_i) = \begin{cases} 1 \text{ if } c_i = C_j \\ 0 \text{ otherwise.} \end{cases}$$

Given a probability distribution p on the discrete space $\Omega = \{C_1, \ldots, C_q\}$, we denote $p_1, \ldots, p_q$ the probability of being an element of Ω, i.e. $p(c_i = C_j) = p_i$. The values $p_1, \ldots, p_q$ entirely define p. The log-likelihood is a natural loss function for probability distributions. Formally the likelihood coincides with a probability value. The logarithmic-based likelihood is defined as follows (under the strict constraint $\sum_j^q p_j = 1$):

$$\mathcal{L}oss_{log}(p|\overrightarrow{x}_i) = -\sum_{j=1}^{q} \mathbb{1}_j(\overrightarrow{x}_i) log(p_j).$$

However, as a probabilistic loss function, the likelihood has some limitations. First, $\mathcal{L}oss_{log}$ is not defined when $p_j = 0$ and $\mathbb{1}_j = 1$. Second, it gives a very high weight to the error when probability is very low. These two issues are a strong limitation when the parameters are obtained through a local optimization process of a classifier. Indeed, avoiding the possibility to have $p_j = 0$ can be difficult when considering complex models. Moreover, exponential costs of error on low probability classes may have a too high effect on the whole model.

One common approach to overcome this problem is to turn the classification problem into a regression problem. The goal of kernel logistic regression and kernel regression is to minimize the least square between the value of the class (0 and 1) and the predicted probability. To this end, both methods independently learn a function (resp. kernel logistic function of kernel f_j for each class C_j). Then, for each x_i, f_j minimizes

$$LeastSquare(\overrightarrow{x}_i, f_j) = (\mathbb{1}_j(\overrightarrow{x}_i) - f_j(\overrightarrow{x}_i))^2.$$

It has been shown that minimizing this distance leads to an faithful estimation of the probability of being in the class C_j. However, since the function are obtained independently, the distribution encoded by the f_n' s does not necessarily satisfy $\sum_{j=1}^q f_j(\overrightarrow{x}_i) = 1$ for all $\overrightarrow{x}_i$. Thus, the predicted values have to be normalized in order to have a probability distribution and then, the faithfulness of the probability of being in the class C_j may be altered.

In this paper, we propose a method that learns the distribution directly. In order to achieve this goal we extend the previous expression in order to take into account the whole probability distribution p. Thus, we obtain:

$$\begin{aligned} LeastSquare(\overrightarrow{x}_i, p) &= \sum_{j=1}^{q} (\mathbb{1}_j(\overrightarrow{x}_i) - p_j)^2 \\ &= 1 + \sum_{j=1}^{q} p_j^2 - 2 * \sum_{j=1}^{q} \mathbb{1}_j(\overrightarrow{x}_i) * p_j. \end{aligned}$$

We build our loss function by removing the constant and normalizing the previous calculus. Then we have :

$$\mathcal{L}oss_{surf}(p|\overrightarrow{x}_i) = \frac{\sum_{j=1}^q \mathbb{1}_j(\overrightarrow{x}_i) * p_j - \frac{1}{2} * \sum_{i=1}^q p_i^2}{n}. \quad (1)$$

This loss function has been used in [11,12] for describing similar loss functions for possibility distributions and use it into regression process. We name it $\mathcal{L}oss_{surf}$ since maximizing this function is equivalent to minimize the square of the distance between p and the optimal probability distribution p^* (here we have for a given $\overrightarrow{x}_i$, $p_j^* = \mathbb{1}_j(\overrightarrow{x}_i)$. Then, contrarily to $(LeastSquare(\overrightarrow{x}_i, f_j)$, $\mathcal{L}oss_{surf}$ takes into account the whole distribution directly, and not the probability values independently.

3 Surface Probabilistic Kernel Classifier Learning

3.1 Definitions

We recall that a classification database is a set of n pairs, or examples, $(\overrightarrow{x}_i, c_i)$, $1 \leq i \leq n$, where $\overrightarrow{x}_i$ is a vector of input variables in the feature space $\mathcal{X}$ and $c_i \in \{C_1, \ldots, C_q\}$ is the class variable. A Surface Kernel probabilistic Classifier (Skc) associates a probability distribution over the classes to a vector of $\mathcal{X}$. A Skc is a set of q kernel functions $(Skc = \{f_1, \ldots, f_q\})$. A function f_j is a kernel function over r support vectors $\overrightarrow{s}_1, \ldots, \overrightarrow{s}_r$ (a support vector is a point in the feature space $\mathcal{X}$)), the same for all the functions, that encodes the probability of the example $\overrightarrow{x}_i$ pertaining to class C_j. Then we have:

$$f_j(\overrightarrow{x}) = \sum_{l=1}^{r} (\alpha_l^j * K(\overrightarrow{x}, \overrightarrow{s}_l)) + \alpha_{r+1}^j \tag{2}$$

where the α_l^j's and α_{r+1}^j are the parameters of the function and $K(.,.)$ is a kernel function. The probability of the example of pertaining to class C_i is then:

$$p_j(\overrightarrow{x}_i) = p(c_i = C_j) = f_j(\overrightarrow{x}_i). \tag{3}$$

We also have the following constraints:

1. $\forall j \in 1, \ldots, q, \forall l \in 1, \ldots, r, \sum_{j=1}^{q} \alpha_k^i = 0$
2. $\sum_{j=1}^{q} \alpha_{r+1}^j = 1.$
3. $\forall j \in 1, \ldots, q, \forall l \in 1, \ldots, r+1, -1 \leq \alpha_l^j \leq 1$

Constraints 1 and 2 guarantee that the probability distribution predicted for a vector $\overrightarrow{x}$ is normalized, i.e.:

$$\forall \overrightarrow{x} \in \mathcal{X}, \sum_{j=1}^{q} f_j(\overrightarrow{x}) = 1.$$

However, these constraints do not ensure that the distribution obtained is a genuine probability distribution. Indeed, we may have $f_j(\overrightarrow{x}) < 0$ or $f_j(\overrightarrow{x}) > 1$. This can be partially overcome with constraint 3, but it is not sufficient in general. This issue will be solved in the optimization process, as it will be explained in the following.

Once the probabilistic kernel functions defined, the goal is to find the Skc function that associate a probability distribution over classes that is as faithful as possible to each input vector of the training set. In the most favorable case, we will obtain a distribution that gives the probability value 1 to the right class. According to the previous definitions, the goal for learning Skc is to find the Skc that maximizes the surface loss function with respect to each example in the training set. This is formulated as follows: Find the α_l^j parameters of a Skc that maximizes the following expression:

$$\mathcal{L}oss_{surf}(Skc) = \sum_{i=1}^{n} \mathcal{L}oss_{surf}(Skc(\overrightarrow{x}_i)|\overrightarrow{x}_i). \tag{4}$$

The maximization of $\mathcal{L}oss_{surf}$ has several advantages:

- $\mathcal{L}oss_{surf}$ is defined even when the probability is equal to 0 and if the probability distribution is not normalized. $\mathcal{L}oss_{surf}$ is also defined for negative values or values greater than 1. Even if these values are not acceptable for probability prediction, it allows local optimizer to explore the whole feature space. Contrarily to the log-likelihood, it makes also the evaluation of the model possible when it performs well for a majority of examples and have aberrant values only for few examples. Moreover, $\mathcal{L}oss_{surf}$ always favors genuine probability functions. Thus, even if the definition of Skc permits such kind of abnormal distribution, this case never appears after the learning of a model in the experiments.
- $\mathcal{L}oss_{surf}$ evaluates the faithfulness of the probability distribution predicted and not only its ability to identify the most probable class. Moreover, even when only one example is considered, all the values of the probability distribution are taken into account (contrarily to log-likelihood) as it can be seen in Equation 1.

Then, this approach has some advantages with respect to the other Kernel approaches. First, the kernel functions have to be learned simultaneously and not one by one as it is done in Kernels approach (even for the binary case). Second, the combination of constraints and properties of the $\mathcal{L}oss_{surf}$ makes that the probability distribution predicted has not to be normalized. Finally, $\mathcal{L}oss_{surf}$ evaluates the faithfulness of the probability distribution without normalization when kernel approaches focus on maximizing the probability associated to the considered class (regardless of the values of the other classes). Even if these two approaches are acceptable in terms of accuracy maximization, the proposed approach seems best suited in terms of quality of the probability distributions learned. But on the contrary to other kernel approaches, finding the Skc function that maximizes $\mathcal{L}oss_{surf}(Skc)$ given a training set is a hard problem which has no simple analytical solution. Section 4 shows how this issue can be handled by local optimization algorithms. However, the computation time performance of these algorithms depends highly on the complexity cost of the evaluation of $\mathcal{L}oss_{surf}(Skc)$.

3.2 Complexity Evaluation and Reformulation of $\mathcal{L}oss_{surf}$

Given n examples, q classes, r support vectors and a Skc function, we have:

$$\begin{aligned}\mathcal{L}oss_{surf}(Skc) &= \sum_{i=1}^{n}(\sum_{j=1}^{q}(\mathbb{1}_j(\overrightarrow{x}_i) * f_j(\overrightarrow{x}_i) - \frac{1}{2}f_j(\overrightarrow{x}_i)^2))\\ &= \sum_{i=1}^{n}\sum_{j=1}^{q}(\mathbb{1}_j(\overrightarrow{x}_i) * \sum_{l=1}^{r}(\alpha_l^j * K(\overrightarrow{x}_i, \overrightarrow{s}_l)) + \alpha_{r+1}^j)\\ &\quad -\frac{1}{2}\sum_{i=1}^{n}\sum_{j=1}^{q}(\sum_{l=1}^{r}(\alpha_l^j * K(\overrightarrow{x}_i, \overrightarrow{s}_l)) + \alpha_{r+1}^j)^2.\end{aligned}$$

Under this form, the complexity of the calculus is $\mathcal{O}(m * q * r)$. This can be problematic since local optimization algorithms require to evaluate the target function frequently. In this case, the optimization process will rapidly become too costly when the size of the training set increases. Fortunately, it can be reformulated in order to lead to a more tractable computation. For the sake of readability, we split $\mathcal{L}oss_{surf}(Skc)$ into two parts, namely

$$\mathcal{L}oss_{surf}(Skc) = Part1 - \frac{1}{2}Part2$$

such that:

$$Part1 = \sum_{i=1}^{n}(\sum_{j=1}^{q}(\mathbb{1}_j(\overrightarrow{x}_i) * f_j(\overrightarrow{x}_i))$$

and

$$Part2 = \sum_{i=1}^{n}\sum_{j=1}^{q} f_j(\overrightarrow{x}_i)^2.$$

$Part1$ can be reformulated as follows:

$$\begin{aligned}Part1 &= \sum_{i=1}^{n}\sum_{j=1}^{q}(\mathbb{1}_j(\overrightarrow{x}_i) * \sum_{l=1}^{r}(\alpha_l^j * K(\overrightarrow{x}_i, \overrightarrow{s}_l)) + \alpha_{r+1}^j)\\ &= \sum_{i=1}^{n}\sum_{j=1}^{q}(\mathbb{1}_j(\overrightarrow{x}_i) * \alpha_{r+1}^j + \sum_{l=1}^{r}(\mathbb{1}_j(\overrightarrow{x}_i) * \alpha_l^j * K(\overrightarrow{x}_i, \overrightarrow{s}_l))\\ &= \sum_{j=1}^{q}\alpha_{r+1}^j * \sum_{i=1}^{n}\mathbb{1}_j(\overrightarrow{x}_i) + \sum_{j=1}^{q}\sum_{l=1}^{r}(\alpha_l^j * \sum_{i=1}^{n}(\mathbb{1}_j(\overrightarrow{x}_i) * K(\overrightarrow{x}_i, \overrightarrow{s}_l)))\\ &= \sum_{j=1}^{q}(\alpha_{r+1}^j * NB_j) + \sum_{j=1}^{q}\sum_{l=1}^{r}(\alpha_l^j * K_j^l)\end{aligned}$$

with $K_j^l = \sum_{i=1}^{n}(\mathbb{1}_j(\overrightarrow{x}_i) * K(\overrightarrow{x}_i, \overrightarrow{s}_l))$ and $NB_j = \sum_{i=1}^{n}\mathbb{1}_j(\overrightarrow{x}_i)$. It is interesting to remark that K_j^l and NB_j do not depend on the α_l^j's. Then, these values can be computed before the optimization process. During the optimization process, the complexity of the computation of $Part1$ goes down to $\mathcal{O}(q * r)$ which is independent from the size of the training set. In the same way, $Part_2$ can be reformulated as follows:

$$\begin{aligned}
Part2 &= \sum_{i=1}^{n}\sum_{j=1}^{q}(\sum_{l=1}^{r}(\alpha_l^j * K(\overrightarrow{x}_i, \overrightarrow{s}_l)) + \alpha_{r+1}^j)^2 \\
&= \sum_{i=1}^{n}\sum_{j=1}^{q}(\sum_{l=1}^{r}(\alpha_l^j * K(\overrightarrow{x}_i \overrightarrow{s}_l)))^2 \\
&\quad + \sum_{i=1}^{n}\sum_{j=1}^{q}((\alpha_{r+1}^j)^2 + 2 * \alpha_{r+1}^j * (\sum_{l=1}^{r}(\alpha_l^j * K(\overrightarrow{x}_i, \overrightarrow{s}_l))) \\
&= \sum_{i=1}^{n}\sum_{j=1}^{q}\sum_{l=1}^{r}\sum_{t=1}^{r}(\alpha_l^j * \alpha_t^j * K(\overrightarrow{x}_i \overrightarrow{s}_l) * K(\overrightarrow{x}_i, \overrightarrow{s}_t)) \\
&\quad + n * \sum_{j=1}^{q}(\alpha_{r+1}^j)^2 + 2 * \sum_{j=1}^{q}(\alpha_{r+1}^j * \sum_{l=1}^{r}(\alpha_l^j * \sum_{i=1}^{n} K(\overrightarrow{x}_i, \overrightarrow{s}_l))) \\
&= \sum_{j=1}^{q}\sum_{l=1}^{r}\sum_{t=1}^{r}(\alpha_l^j * \alpha_t^j * K_{s,l}) + n * \sum_{j=1}^{q}(\alpha_{r+1}^j)^2 + 2 * \sum_{j=1}^{q}(\alpha_{r+1}^j * \sum_{l=1}^{r}(\alpha_l^j * K_l))
\end{aligned}$$

where $K_{s,l} = \sum_{i=1}^{n}(K(\overrightarrow{x}_i, \overrightarrow{s}_l) * K(\overrightarrow{x}_i, vs_t))$ and $K_l = \sum_{i=1}^{n} K(\overrightarrow{x}_i, \overrightarrow{s}_l)$. As previously, $K_{s,l}$ are independent from the α_l^j's. Then if we pre-compute the values $K_{s,l}$, the complexity of the computation of $Part2$ goes down to $\mathcal{O}(q * r^2)$. We obtain a complexity of $\mathcal{O}(q*r^2)$ for the calculus of $\mathcal{L}oss_{surf}(Skc)$ if we compute the values K_j^l, NB_j, $K_{s,l}$ and K_l before the optimization process. Then, we can perform an optimization process that is independent from the size of the database.

4 Optimization Process

As pointed out in the previous section, the fact learning f_j functions has to be done simultaneously makes that there is no simple analytical solution. Thanks to the offline computation of the values that depends on the size of the database, the evaluation of the target function of a model is not costly. In this context, the use of local optimization algorithm is possible. However, it requires to previously choose the number of support vectors and their values. The number of support vectors is a parameter of the algorithm. The vectors are then obtained with the k-means clustering algorithm. We use two different optimization algorithms. The first one is the Nelder-Mead algorithm [8] which is very fast but converges to local optimum. The second one is the particle swarm meta-heuristics [6] which is more costly but has better optimization performances.

4.1 Nelder-Mead Implementation

The Nelder-Mead algorithm is a heuristics for maximizing of a function F in a N dimensions space. It is based on the deformation of a simplex until it converges to a local optima (Algorithm 1). The algorithm stops after a fixed number of loops without increasing $F(e_1)$. In addition to its efficiency, the Nelder-Mead algorithm is very simple and does not require to derive the function F. However, it can be easily trapped into local optima and it depends on the starting configuration. Results can be improved by restarting the algorithm with different starting configurations. In our case a state e corresponds to the vectors that describe the parameters α_l^j of a Skc function given a kernel and a set of r support vectors. Then the dimension of the state space is $N = q * (r + 1)$. $\mathcal{L}oss_{surf}(Skc)$ corresponds to the function F. The starting configurations are chosen randomly and have to respect the constraint described in section 3.1. The operations on the space states ensure that the constraints are not violated during the algorithm.

Algorithm 1. Nelder-Mead

```
Choose $N + 1$ points $e_1, \dots, e_{N+1}$
Order the points with respect to $F$
Compute $e_0$ the center of gravity of $e_1, \dots, e_N$
$e_r = 2 * e_0 - e_{N+1}$
if $F(e_r) > F(e_N)$ then
  $e_t = e_0 + 2 * (e_0 - e_{N+1})$
  if $F(e_t) > F(e_r)$ then
    $e_{N+1} = e_t$
  else
    $e_{N+1} = e_r$
  end if
else
  $e_c = e_{N+1} + \frac{1}{2} * (e_0 - e_{N+1})$
  if $F(e_c) \geq F(e_n)$ then
    $e_{N+1} = e_c$
  else
    forall $i \geq 2$ $e_i = e_1 + \frac{1}{2} * (e_i - e_1)$
  end if
end if
return to step 2
```

4.2 Particle Swarm Implementation

In order to overcome the problem of local optima, we propose to use the particle swarm optimization algorithm. One of the advantages of the particle swarm optimization with respect to the other meta-heuristics is that it is particularly suitable for continuous problems. Particle swarm works in the same settings than Nelder-Mead algorithm. Particle swarm with N particles (N is no longer the dimension of the state space) is described in Algorithm 2.

Algorithm 2. Particle Swarm Optimization

```
Choose randomly $N$ particles $e_1, \dots, e_N$
for all $i$ $be_i = e_i$
$eg = argmax_{e_i}(F(e_i)$ (best know position)
choose randomly $N$ velocity vectors $v_1, \dots, v_N$
repeat
  for $i = 1, \dots, N$ do
    choose randomly $r_p$ and $r_g$ in $[0, 1]$
    $v_i = \omega * v_i + \phi_p * r_p * (be_i - ei) + \phi_g * r_g * (eg - ei)$
    $e_i = e_i + v_i$
    if $F(e_i) > F(be_i)$ then
      $be_i = e_i$
      if $F(be_i) > F(eg)$ then
        $eg = be_i$
      end if
    end if
  end for
until a chosen number of times
```

Here, one particle represents the parameters of Skc function (α_l^j). At each step of the algorithm, each particle is moved along its velocity vector (randomly fixed at the beginning). The velocity vectors are updated at each step by considering the current vectors, the vector from the current particle position to particle best known position and the vector from the current particle position to global swarm's best known position. In order to maintain the constraint 1 and 2 over the parameters α_l^j the values v_l^j of the velocity vector have to satisfy the following constraints:

- $\forall j \in 1, \dots, q, \forall l \in 1, \dots, r+1, \sum_{j=1}^{q} v_k^i = 0$
- $\forall j \in 1, \dots, q, \forall l \in 1, \dots, r+1, -1 \leq v_l^j \leq 1$

If we have $-1 > \alpha_l^j$ (resp. $\alpha_l^j > 1$) after the application of the velocity vector, we fix $\alpha_l^j = -1$ (resp. $\alpha_l^j = 1$).

The particle swarm algorithm is easy to tune. The three parameters for the updating of the velocity ω, ϕ_p and ϕ_g correspond respectively to the coefficient for the current velocity, the coefficient for the velocity to the particle best known position and the coefficient for the velocity to the global swarm's best known position. Based on [14], we use generic values that perform well in most of the cases ($\omega = 0.72$, $\phi_p = 1.494$, $\phi_g = 1.494$ and 16 particles).

5 Experimentation

In this section, we compare our algorithms with naive Bayes classifier (NBC) and the kernel approaches (SVM) based on least square minimization described in [16] and implement in the java version of LibSVM. We note Skc_{NM} for the maximization of $\mathcal{L}oss_{surf}$ with the Nelder-Mead algorithm and Skc_{PSO} for the maximization of $\mathcal{L}oss_{surf}$ with the particle swarm optimization algorithm. We compare the results with respect to the accuracy, $\mathcal{L}oss_{log}$ (mind that here the lower the value, the better) and $\mathcal{L}oss_{surf}$. We also report time performance of SVM, Skc_{NM} and Skc_{PSO}. In the first experiments, we make 100000 steps of particle swarm movement and 5 restarts of Nelder-Mead algorithm. We empirically choose the number of support vectors with the formulas $r = 1 + 10log(n/3)$ where n is the number of examples. We use Gaussian kernels. All the experiments are done on a $3Ghz$ computer and all the algorithms are implemented with the JAVA language.

Table 1. Comparison of algorithms on 6 UCI dataset (10-cross validation)

db.	Alg.	Acc.	$\mathcal{L}oss_{log}$	$\mathcal{L}oss_{surf}$
Diab.	NBC	75.7[5.1]	0.57[0.15]	0.32[0.04]
	SVM	74.9[7.9]	0.52[0.09]	0.32[0.04]
	Skc_{NM}	76.2[8.1]	0.51[0.13]	**0.33[0.04]**
	Skc_{PSO}	**76.3[7.6]**	**0.49[0.1]**	**0.33[0.03]**
Breast.	NBC	95.7[2.1]	0.26[0.13]	0.45[0.02]
	SVM	95.1[2.4]	0.13[0.06]	0.46[0.01]
	Skc_{NM}	95.9[1.9]	0.12[0.04]	**0.46[0.01]**
	Skc_{PSO}	**95.9[1.8]**	**0.11[0.05]**	**0.46[0.01]**
Iono.	NBC	84.8[6.1]	0.7[0.34]	0.36[0.05]
	SVM	88.3[5.7]	0.29[0.06]	0.41[0.02]
	Skc_{NM}	90.8[2.7]	0.22[0.06]	**0.42[0.01]**
	Skc_{PSO}	**93.4[2.7]**	**0.2[0.06]**	**0.42[0.01]**
Mag. Tel.	NBC	72.7[0.8]	0.98[0.03]	0.26[0.0]
	SVM	**87.6[0.8]**	**0.3[0.01]**	**0.4[0.0]**
	Skc_{NM}	84.4[0.7]	0.38[0.01]	0.38[0.0]
	Skc_{PSO}	85.5[0.8]	0.38[0.01]	0.38[0.0]
Glass	NBC	50.3[15.4]	1.24[0.4]	0.22[0.05]
	SVM	**72.8[9.5]**	**0.81[0.18]**	**0.29[0.05]**
	Skc_{NM}	66.7[9.2]	0.86[0.19]	0.23[0.04]
	Skc_{PSO}	71.5[9.4]	0.79[0.13]	0.28[0.04]

5.1 Benchmark Dataset

In order to check the effectiveness of the algorithms, we used 5 benchmarks from UCI[1]. All the datasets have numerical attributes only. The Diabetes database describes 2 classes with 768 examples. The Breast cancer database contains 699 examples that describes 2 classes. The Ionosphere database describes 2 classes with 351 examples. The Magic telescope database contains 19020 examples that describes 2 classes. The Glass database describes 7 classes with 224 examples. The results presented in Table 1 are for 10-cross validation. Bold results correspond to the highest values. It shows that the Sks approaches outperform clearly NBC

[1] `http://www.ics.uci.edu/~mlearn/MLRepository.html`

Table 2. Computation time for the *Skc* algorithms on 6 UCI datasets

database	time		
	SVM	Skc_{NM}	Skc_{PSO}
Diabetes	0.2s	0.4s	7.9s
Breast c	0.1s.	0.2s	8s
Iono.	0.1s	1s	8s
Mag. Tel.	57s	1.9s	9.3s
Glass	0.1s	8s	23s

on all the databases (with a statistically significant difference for 3 databases) both for the accuracy and the $Loss_{surf}$. Skc_{PSO} outperforms SVM on 3 of the 5 databases (with a statistically significant difference for 1 database) and is outperformed on the two remaining ones (with a statistically significant difference for 1 database). This shows that our approaches compete with SVM probabilistic approach in terms of classification and have good performances for describing faithful probability distributions (even if we consider the log-likelihood). Skc_NM and Skc_PSO have close performances except when the number of classes increases. We can suppose that in this case Skc_NM is more easily trapped in local optima.

Table 2 gives the time in seconds for performing the optimization of *Skc*. As expected Skc_NM is around 10 times more efficient than Skc_PSO. Even if these times are larger than the SVM ones, they remain very low, and are not much sensitive to the size of the dataset (times for Ionosphere and Magic Telescope are closed for instance). The size of the database only matters for computing the support vectors and the pre-computed values (the number of support vector also increases slightly when the size increases). When the size of the database increases, as for magic telescope, our approaches become much faster than SVM approach.

5.2 KDD Cup 2012 Dataset

In order to check the scalability of our algorithms, we use our approaches on the KDD Cup 2012 database. This database describes a social network of microblogging. "Users" are people in the social network and "items" are famous people or objects that the users may follow. Users may be friend with other users. The task of this challenge is to predict if a user will accept or not to follow an item proposed by the system.

There are 10 millions of users described by their age, their genre, some keywords and their friends. There are 50000 items described by keywords and tags. The database contains 70 millions of propositions to follow an item with a label that indicates if the user has accepted the proposition or not. The problem is then a binary classification problem. We define 8 attributes based respectively on i)the percentage of users of the same genre as the target one, which follow the item, ii) the percentage of users of the same age category as the target one,

Table 3. Comparison of algorithms on KDD Cup 2012 dataset

database	Alg.	Acc.	$\mathcal{L}oss_{log}$	$\mathcal{L}oss_{surf}$	time
Size=1K	NBC	66.4	0	0.278	-
	SVM	70.8	0.580	0.302	0.4
	Skc_{NM} r=10	71.0	0.557	0.312	0.2s
	Skc_{NM} r=100	71.0	0.57	0.311	12s
	Skc_{PSO} r=10	71.4	0.557	0.313	8s
	Skc_{PSO} r=100	71.6	0.572	0.311	56s
Size=10K	NBC	68.0	0.663	0.285	-
	NBC	71.9	0.57	0.314	37s
	Skc_{NM} r=10	71.5	0.559	0.314	0.3s
	Skc_{NM} r=100	72.2	0.556	0.313	14s
	Skc_{PSO} r=10	71.6	0.56	0.313	8s
	Skc_{PSO} r=100	72.7	0.55	0.320	55s
Size=100K	NBC	69.0	0.662	0.292	-
	SVM	72	0.55	0.315	3.5 hours
	Skc_{NM} r=10	72.2	0.547	0.316	2s
	Skc_{NM} r=100	72.2	0.543	0.318	30s
	Skc_{PSO} r=10	72.2	0.547	0.316	10s
	Skc_{PSO} r=100	72.7	0.537	0.319	75s
Size=1M	NBC	68.3	0.665	0.29	-
	SVM	-	-	-	-
	Skc_{NM} r=10	72.2	0.551	0.315	19s
	Skc_{NM} r=100	72.5	0.543	0.318	97s
	Skc_{PSO} r=10	72.0	0.551	0.315	27s
	Skc_{PSO} r=100	73	0.534	0.321	119s

which follow the item, iii) the session time, iv) the number of friends of the target user that follow the item, v) the distance between the items followed by the user and the target item, vi) the number of users that follow the target items, vii) the number of items that are followed by the user, and viii) the number of times the item has been proposed to the user. We build a test dataset of around 1.9 millions of propositions.

Table 3 reports the result with different size of training sets (without any common tuple with the test dataset) and different number of support vectors. The results are computed on the test dataset. We can observe that performances increase when the size of the database increases, even if the dataset is summarized by the pre-computed values in the optimization process. Skc_PSO performs slightly better than Skc_NM and SVM approach. Last, time values confirm the efficiency of the approach and its low sensitivity with respect to the size of the dataset (less than 2 minutes for the Skc_PSO with r = 100 and 1 million examples in the training set). It shows that our approaches are usable on very large database while SVM would have difficulties for managing databases with more than 10000 examples (and is intractable for more than 100000 examples).

6 Conclusion and Future Works

In this paper we have proposed a probabilistic classification method based on the maximization of a loss probabilistic function that takes into account the whole probability distribution and not only the probability of the class. We propose two algorithms that simultaneously learn a set of kernel functions that encodes a probability distribution over classes without any post-normalization process. Last, we show that the computation time of the approach is very little sensitive to the size of the dataset. Our method competes with the other kernel approaches on the used benchmark datasets. Experiments on the KDD Cup 2012 dataset confirm that the approach is efficient on very large datasets when kernel methods are not tractable. Moreover, the parameters of the algorithm can be tuned automatically as it has been done the whole experimentation.

In the future, the way of choosing the number of kernels and computing the support vectors has to be more deeply investigated and alternatives to clustering approach have to be explored. We will also study how the approach can be embedded into a gradient boosting process [3] in order to increase the performance when the number of attributes and classes is large. Lastly, we have to compare our algorithm more deeply with the other probabilistic approaches.

References

1. Cover, T.M., Hart, P.E.: Nearest neighbour pattern classification. IEEE Transactions on Information Theory 13, 21–27 (1967)
2. Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., Lin, C.-J.: Liblinear: A library for large linear classification. Journal of Machine Learning Research 9, 1871–1874 (2008)
3. Friedman, J.H.: Greedy function approximation: A gradient boosting machine. Annals of Statistics 29, 1189–1232 (2000)
4. Jaakkola, T.S., Haussler, D.: Probabilistic kernel regression models. In: Proceedings of the 1999 Conference on AI and Statistics. Morgan Kaufmann (1999)
5. Jaakkola, T.S., Jordan, M.I.: A variational approach to bayesian logistic regression models and their extensions (1996)
6. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proceedings of the IEEE International Conference on Neural Networks 1995, pp. 1942–1948 (1995)
7. Langley, P., Iba, W., Thompson, K.: An analysis of bayesian classifiers. In: Proceedings of AAAI 1992, vol. 7, pp. 223–228 (1992)
8. Nelder, J.A., Mead, R.: A simplex method for function minimization. The Computer Journal 7(4), 308–313 (1965)
9. Nickisch, H., Rasmussen, C.E.: Approximations for binary gaussian process classification. Journal of Machine Learning Research 9, 2035–2078 (2008)
10. Opper, M., Winther, O.: Gaussian processes for classification: Mean field algorithms. Neural Computation 12, 2000 (1999)

11. Serrurier, M., Prade, H.: Imprecise regression based on possibilistic likelihood. In: Benferhat, S., Grant, J. (eds.) SUM 2011. LNCS, vol. 6929, pp. 447–459. Springer, Heidelberg (2011)
12. Serrurier, M., Prade, H.: Maximum-likelihood principle for possibility distributions viewed as families of probabilities (regular paper). In: IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), Taipei, Taiwan, pp. 2987–2993 (2011)
13. Sugiyama, M.: Superfast-trainable multi-class probabilistic classifier by least-squares posterior fitting. IEICE Transactions on Information and Systems 93-D(10), 2690–2701 (2010)
14. Trelea, I.C.: The particle swarm optimization algorithm: convergence analysis and parameter selection. Information Processing Letters 85(6), 317–325 (2003)
15. Williams, C.K.I., Barbe, D.: Bayesian classification with gaussian processes. IEEE Transactions on Pattern Analysis and Machine Intelligence 20(12), 1342–1351 (1998)
16. Wu, T.-F., Chih-Jen, C.-J., Weng, R.C.: Probability estimates for multi-class classification by pairwise coupling. Journal of Machine Learning Research 5, 975–1005 (2004)