Unsupervised Classifier Selection Based on Two-Sample Test

Timo Aho, Tapio Elomaa, and Jussi Kujala

Department of Software Systems, Tampere University of Technology
P. O. Box 553 (Korkeakoulunkatu 1), FI-33101 Tampere, Finland
{timo.aho,tapio.elomaa,jussi.kujala}@tut.fi

Abstract. We propose a well-founded method of ranking a pool of m trained classifiers by their suitability for the current input of n instances. It can be used when dynamically selecting a single classifier as well as in weighting the base classifiers in an ensemble. No classifiers are executed during the process. Thus, the n instances, based on which we select the classifier, can as well be unlabeled. This is rare in previous work. The method works by comparing the training distributions of classifiers with the input distribution. Hence, the feasibility for unsupervised classification comes with a price of maintaining a small sample of the training data for each classifier in the pool.
In the general case our method takes time $O\big(m(t+n)^2\big)$ and space $O(mt+n)$, where t is the size of the stored sample from the training distribution for each classifier. However, for commonly used Gaussian and polynomial kernel functions we can execute the method more efficiently. In our experiments the proposed method was found to be accurate.

1 Introduction

The problem of *dynamic classifier selection* arises prominently in data stream classification [1], but it is also present in, e.g., tracking recurring drifting concepts [2,3], dynamical learning algorithm selection [4], and weighting or selecting the base classifiers in an ensemble [5,6]. All of these situations are dynamic in the sense that a classification algorithm has not been fixed beforehand in a separate training phase. Rather, the instances that are observed online affect our choice.

For example, in concept drifting the distribution underlying the data keeps changing over time. Often the states of the distribution reoccur after a while. Thus, it is useful to store and restore the data and classifiers of the past. Most of the practical restoring methods include choosing the classifier with the best fit for the current input. This can be the case with a single classifier [2] and with ensemble classifiers [6,7]. In the latter case the ranking information is used to weight the responses of classifiers depending on their suitability for the current input.

In either case, we have a set (pool) $\mathcal{H} = \{h_1, h_2, \ldots, h_m\}$ of varying kinds of classifiers $h\colon \mathcal{X} \to \mathcal{Y}$ available. The classifiers are trained on dissimilar instance distributions. Each classifier h_i maps any instance $\boldsymbol{x} \in \mathcal{X}$ to a class label $h_i(\boldsymbol{x}) \in$

$\mathcal{Y}$. Assume also that we can easily associate with each classifier a random sample drawn from its underlying training set. Moreover, we have a sequence of new instances $(\boldsymbol{x}) = \boldsymbol{x}_1, \boldsymbol{x}_2, \boldsymbol{x}_3, \ldots, \boldsymbol{x}_n$ available; they may be labeled (belong to $\mathcal{X} \times \mathcal{Y}$) or not (belong to $\mathcal{X}$). Keep in mind that this sequence may only be the first batch from a longer sequence.

How should we proceed when we want to obtain a good classifier for $(\boldsymbol{x})$? If the sequence is labeled the most straightforward approach, of course, is to employ the most suitable learning algorithm (see e.g. [8]), feed $(\boldsymbol{x})$ as training set, and take the resulting classifier as our choice. However, the set $(\boldsymbol{x})$ may be small when compared to the training sets on which the classifiers in $\mathcal{H}$ were trained. Also, the classifiers in the pool may, e.g., be more general than is possible to attain by training a new one.

Furthermore, if the classifiers are not of the same type, it is possible that different types of classifiers are chosen to best suit the underlying distributions. According to the "No Free Lunch" theorems [9] no single classifier type is superior in all situations. Experimental evidence also supports this [8]. Because of all these reasons, we concentrate on methods that choose the classifier from the pool $\mathcal{H}$.

The standard approach to choosing a classifier from $\mathcal{H}$ is to execute each of them on the input sequence $(\boldsymbol{x})$ and choose the most accurate one. This is the traditional solution to the dynamic classifier selection problem and has been used, e.g., by Watanabe [10]. Furthermore, the same method has generally been used for weighting the classifiers in an ensemble [6]. Assuming linear-time execution of classifiers, the time requirement of this procedure is $O(mn)$ for m classifiers and an example sequence of length n. In this case the sequence $(\boldsymbol{x})$ needs to be labeled. Thus, the method is infeasible for, e.g., uncategorized web pages. We are aiming at an efficient solution that would also let us use unsupervised learning.

Basically the method that we propose, MMDSel, compares sample distributions. This is done by maintaining a small sample of the training data for each classifier in the pool $\mathcal{H}$. However, it is not always necessary to store the samples explicitly. MMDSel gives the similarity of the samples drawn from the training data with the input sequence $(\boldsymbol{x})$. Thus, we can either rank the classifiers or select the one having a training distribution most similar to the input distribution. The main advantage of the method is that no classifier needs to be executed. Hence, it is useful in at least three different situations. First and most importantly, MMDSel can be used in unsupervised environments. Second, it may be more efficient when execution of classifiers is inefficient. There may also be other reasons — like privacy issues [11] — to prevent unnecessary classification. Third, our method does not need extensive preparation operation other than the sample storing (cf., e.g., meta-learning approaches [12]). Thus, the method is feasible for dynamic setting where the pool $\mathcal{H}$ may change online.

Most earlier approaches are usable only in the supervised setting [2,5]. However, Ali and Smith [8] aimed to find the most suitable classifier types for different kinds of distributions using fixed complexity measures. Some of the measures are

statistical and depend only on the distribution of the data. Thus, the approach is also feasible for unsupervised learning. Nevertheless, Ali and Smith did not really study dynamic selection of trained classifiers. Rather, they aimed to find a rule-based classifier type selection grounded on prior knowledge of the problem. Moreover, Zhu, Wu, and Yang [1] gave a method in data stream model that includes partitioning the whole instance space into subsets according to the feature values. The approach uses class labels of instances only in finding the classification accuracy of base classifiers for these subsets. Hence, the method could partly be used under unsupervised conditions.

A field with some similarity with classifier selection is choosing with expert advice [13]. Here the task is to minimize the amount of unsatisfactory decisions for a sequence of tasks. Traditionally the experts and the distribution underlying the decision are quite static. Similarly, we also aim to find the expert that is the best in hindsight. However, in dynamic classifier selection framework the distribution alters all the time and, thus, we are interested in single decision, not in the decision sequence. We do not necessarily have any optimal classifier; any single classifier would usually have been a poor choice for the whole sequence. Rather the best classifier depends entirely on the current distribution. Also, the set of classifiers $\mathcal{H}$ may alter.

The remainder of this paper is organized as follows. In Section 2 we briefly examine the theoretical background of MMDSel. Then, in Section 3, we present a way to compute our method efficiently with some kernels. Section 4 reports on an empirical evaluation of the proposed approach. After that we give the concluding remarks of this work.

2 Classifier Selection Using Samples of Training Examples

The basic idea of MMDSel is to compare a kind of a fingerprint of the training distribution of a classifier with the one of the input distribution. One could, of course, use the classifier itself as a fingerprint. That is, we could train a new classifier for the input examples ($\boldsymbol{x}$) and search for the nearest one in the set $\mathcal{H}$. If the classifiers have an explicit weight vector that includes all the information about the classifier, we can simply search for the most similar weight from the pool. Support vector machines (SVMs) with a linear kernel are an example of such classifiers. With the trivial search the best classifier can be found in $O(m + f(n))$ time and $O(m + n)$ space. Here the function $f(n)$ is the average time consumption of classifier learning a set of n elements. Nearest neighbor methods [14,15,16] could be used to reduce the time requirement. This method, though, is only usable for certain classifiers and only in the supervised setting. Also, in our experiments the method appeared quite unreliable.

Instead, MMDSel rather stores a sample of the training set of each classifier. The sample is in fact a fingerprint of the training distribution. If the input sequence ($\boldsymbol{x}$) is drawn from the same (or a very similar) distribution, we should choose the classifier trained for it — most likely it is the most suitable one. To

be exact, let the set $S = \{s_1, s_2, \ldots, s_m\}$ contain the training samples of size t for the m classifiers in $\mathcal{H}$. If we select the sample that is most similar with the sequence $(\boldsymbol{x})$ of length n, we should be able to find the classifier that best fits the current input without executing any of the classifiers.

The methods that compare distribution similarity via samples are called two-sample tests. For example, a two-sample test based on *maximum mean discrepancy* (MMD) proposed by Gretton et al. [17,18] and Smola et al. [19] uses $O\big((t+n)^2\big)$ time and $O(t+n)$ space. On the other hand, Borgwardt et al. [20] propose computing MMD in linear $O(t+n)$ time by randomized approximation. However, by our experiments, this leads to a significant reduction of accuracy.

MMD has previously been used for a somewhat similar application by Huang et al. [21]. They use MMD values to solve the sample selection bias problem where the test distribution differs from the training distribution.

There are several additional minor advantages in the two-sample approach adopted in MMDSel. Firstly, due to statistical insignificance of the labels, mislabeled examples have only a minor impact on the result. Also, in some cases this measure for the suitability of the classifier may be more appropriate than the error rate of classifiers used in the traditional solution [8].

2.1 Comparing Distributions with MMD

Let us now briefly introduce MMDSel and the measure MMD for executing it. The measure was originally designed for two-sample tests in which the problem is to find out whether two given samples come from different distributions. Formally we are given two samples X and Y drawn i.i.d. from distributions D_1 and D_2, respectively, and we want to know whether $D_1 \neq D_2$. In our application it would be useful to know also the "distance" or dissimilarity $d(D_1, D_2)$ between the distributions because the distribution may have changed only slightly.

In fact, MMD is a measure of dissimilarity — or more specifically, discrepancy — between the distributions. It is essentially the maximum difference between the mean of test function values on the distributions. Thus, informally, if the MMD between two samples is near zero the distributions are very similar. With the computed MMD value there are multiple methods to decide if we have enough evidence to reject the null hypothesis $D_1 = D_2$ or not [17]. Because we are only interested in rating the alternatives due to their similarity, the measure for the discrepancy itself is enough for our needs.

Formally MMD is defined by Gretton et al. [17] as follows.

Definition 1 *Let $\mathcal{F}$ be a class of test functions $f\colon \mathcal{X} \to \mathbb{R}$ and let D_1 and D_2 be distributions defined on the domain $\mathcal{X}$. Then the maximum mean discrepancy is*

$$\mathrm{MMD}[\mathcal{F}, D_1, D_2] := \sup_{f \in \mathcal{F}} \big(\boldsymbol{E}_{\boldsymbol{x} \sim D_1}[f(\boldsymbol{x})] - \boldsymbol{E}_{\boldsymbol{y} \sim D_2}[f(\boldsymbol{y})]\big) \ .$$

We select $\mathcal{F}$ to be a reproducing kernel Hilbert space (RKHS) with an associated kernel k. The kernel selection lets us control which properties of distributions are emphasized. Thus, we should choose the kernel according to the

features in which we have an interest. Gretton et al. [17] also proved that, in compact domains $\mathcal{X}$, with so-called *universal kernels* [22] MMD attains zero value if and only if D_1 equals D_2. In practice this means that, given that the samples are large enough, two different distributions can be separated with a universal kernel family. Steinwart [22] proved that, e.g., Gaussian and Laplacian kernels are universal. Even if we are not interested in the two-sample test itself, these kernels could be useful in some situations.

Gretton et al. [17] devised an easier way to calculate MMD owing to the fact that in RKHS the function f can be evaluated by the inner product $f(x) = \langle k(x, \cdot), f \rangle$. Hence, let x and x' be independent random variables with distribution D_1 and, similarly, $y, y' \sim D_2$. The square of MMD behaves identically as a measure for discrepancy. For it a more useful form can be derived [17]:

$$\begin{aligned} \mathrm{MMD}^2[\mathcal{F}, D_1, D_2] = \mathbf{E}_{\boldsymbol{x},\boldsymbol{x}' \sim D_1}[k(\boldsymbol{x}, \boldsymbol{x}')] - 2\mathbf{E}_{\boldsymbol{x} \sim D_1, \boldsymbol{y} \sim D_2}[k(\boldsymbol{x}, \boldsymbol{y})] \\ + \mathbf{E}_{\boldsymbol{y},\boldsymbol{y}' \sim D_2}[k(\boldsymbol{y}, \boldsymbol{y}')] \ . \end{aligned} \tag{1}$$

In practice we are dealing with samples X and Y drawn i.i.d. from D_1 and D_2, respectively. Let $(\boldsymbol{z}_1, ..., \boldsymbol{z}_n)$ be i.i.d. random variables, where $\boldsymbol{z}_i = (\boldsymbol{x}_i, \boldsymbol{y}_i)$ and $\boldsymbol{x}_i \in X, \boldsymbol{y}_i \in Y$. Gretton et al. [17] proved that an unbiased empirical estimate for MMD squared is

$$\mathrm{MMD}^2[\mathcal{F}, X, Y] = \frac{1}{n(n-1)} \sum_{i \neq j}^{n} h(\boldsymbol{z}_i, \boldsymbol{z}_j) \ , \tag{2}$$

where $h(\boldsymbol{z}_i, \boldsymbol{z}_j) = k(\boldsymbol{x}_i, \boldsymbol{x}_j) + k(\boldsymbol{y}_i, \boldsymbol{y}_j) - k(\boldsymbol{x}_i, \boldsymbol{y}_j) - k(\boldsymbol{x}_j, \boldsymbol{y}_i)$. Using this we can easily compute the MMD value.

Gretton et al. [17] also gave a biased estimate that can be computed under some restrictions[1] for the kernel function as

$$\begin{aligned} \mathrm{MMD}^2[\mathcal{F}, X, Y] = \frac{1}{n^2} \sum_{i,j=1}^{n} k(\boldsymbol{x}_i, \boldsymbol{x}_j) - \frac{2}{nt} \sum_{i,j=1}^{n,t} k(\boldsymbol{x}_i, \boldsymbol{y}_j) \\ + \frac{1}{t^2} \sum_{i,j=1}^{t} k(\boldsymbol{y}_i, \boldsymbol{y}_j) \ , \end{aligned} \tag{3}$$

where n and t are the sizes of samples X and Y. The estimate is biased, but there is an upper bound for the bias [17]. In our experiments we found this estimate to be slightly more accurate.

To execute MMDSel and rank the classifiers in a pool of size m we have to compute MMD for m pairs of samples. This results in the dissimilarity of each classifier with the current input distribution. In other words we have ranked the classifiers by their suitability for the current input. Thus we can easily either weight the classifiers according to the rank or choose the most suitable one.

[1] It is enough that for two i.i.d. random variables $\boldsymbol{x}, \boldsymbol{x}' \sim D$: $\mathbf{E}_{\boldsymbol{x},\boldsymbol{x}' \sim D}[k(\boldsymbol{x}, \boldsymbol{x}')] < \infty$ [17].

Because computing MMD for each sample pair takes $O\big((t+n)^2\big)$ time, MMDSel can be executed in time $O\big(m(t+n)^2\big)$ and space $O(mt+n)$. However, below we show that for some kernels the evaluation can be done much more efficiently.

3 Computing MMD Efficiently

We now discuss some ways to improve the efficiency of MMDSel. As mentioned the choice of a kernel affects the value of MMD. Thus, we should choose the kernel that is able to track the essential attributes of distributions — i.e., those that determine the similarity. For example, with a linear kernel MMD clearly indicates the difference between expected values.

3.1 Polynomial kernels

For often-used polynomial kernels (1) can be simplified significantly. This leads to efficient computation of MMDSel.

The main idea of the optimization is the following. For example, for a symmetric and linear (e.g., Euclidean inner product in $\mathbb{R}$) kernel (1) can be computed by

$$\begin{aligned}\mathrm{MMD}^2[\mathcal{F}, D_1, D_2] = k(\mathbf{E}_{\boldsymbol{x}\sim D_1}[\boldsymbol{x}], \mathbf{E}_{\boldsymbol{x}\sim D_1}[\boldsymbol{x}]) - 2\,k(\mathbf{E}_{\boldsymbol{x}\sim D_1}[\boldsymbol{x}], \mathbf{E}_{\boldsymbol{y}\sim D_2}[\boldsymbol{y}]) \\ + k(\mathbf{E}_{\boldsymbol{y}\sim D_2}[\boldsymbol{y}], \mathbf{E}_{\boldsymbol{y}\sim D_2}[\boldsymbol{y}])\ .\end{aligned} \tag{4}$$

Hence, we do not need to store the sample for each classifier explicitly. It suffices to store the expected values. Moreover, it is enough to compute the expected value of input examples only once during the test. The time and space requirement for these preparations is clearly $O(mtd)$ for d dimensional data. On the other hand, during the actual selection process only a constant amount of inner products are calculated, yielding a time requirement of $O((m+n)d)$.

The optimization can be generalized to all the polynomial kernels $(\langle\cdot,\cdot\rangle + c)^p$ of a finite integer degree p, where $c \geq 0$ is a constant. Using the technique introduced by Raykar et al. [23] polynomial kernels can be calculated in

$$r_{pd} = \binom{p+d}{d}$$

time and space. Hence, MMDSel can be executed in $O((m+n)r_{pd})$ time and space with $O(mtr_{pd})$ time and space preparation. With the natural assumption that $p \ll d$, the value r_{pd} can be upper bounded by $O(d^p)$.

Compared to the brute force solution in $O\big(md(t+n)^2\big)$ time and $O(d(mt+n))$ space, the optimized MMDSel is useful for low values of p (e.g., quadratic or linear kernel) and for very large values of t or n.

3.2 Gaussian kernels

Simple polynomial kernels may not be enough for our needs. Steinwart [22] defined kernel classes to be universal if they are dense on compact domains. These kernels can approximate any other kernel if the sample sizes are increased enough. If we want to use a universal kernel, e.g., a Gaussian one, a similar optimization method can be used to approximate them.

Yang et al. [24] and Raykar et al. [23] present an improved fast Gaussian transformation (IFGT). Their approach is based on calculating only the first terms of the Taylor series representation of Gaussian kernel. In our case the basic idea of IFGT is to approximate the Gauss transform at a chosen point $\boldsymbol{y}_*$

$$\begin{aligned} G\left(\boldsymbol{x}_j\right) &= \sum_{i=1}^{t} \exp\left(-\left\|\boldsymbol{x}_j - \boldsymbol{y}_i\right\|^2 / \sigma^2\right) \\ &= \sum_{i=1}^{t} \exp\left(-\left\|\boldsymbol{y}_i - \boldsymbol{y}_*\right\|^2 / \sigma^2\right) \exp\left(-\left\|\boldsymbol{x}_j - \boldsymbol{y}_*\right\|^2 / \sigma^2\right) \\ &\qquad \cdot \exp\left(2\left\langle \boldsymbol{x}_j - \boldsymbol{y}_*, \boldsymbol{y}_i - \boldsymbol{y}_*\right\rangle / \sigma^2\right) \quad . \end{aligned} \tag{5}$$

However to gain accuracy they cluster the space and replace the vector $\boldsymbol{y}_*$ with the centers of these clusters.

With IFGT the Gaussian kernel can be approximated in

$$O\big(nk' r_{(p'-1)d} + nk\big)$$

time with

$$O\big(t \log k + t r_{(p'-1)d}\big)$$

time preparation. Here $k < t$ is the number of clusters in the example space and $k' < k$ the maximum number of neighbor clusters and p' the number of Taylor series terms to be computed. Both the p' and k' depend on the desired error bound $\epsilon > 0$ and k' also depends on Gaussian kernel bandwidth. The space usage is $O\big(k r_{(p'-1)d} + t + n\big)$. Recall that $r_{pd} = O(d^p)$.

The procedure is interesting in our application, because if we cluster the whole instance space at once, we can reduce the running time significantly. If the values corresponding to $\boldsymbol{y}_*$ in (5) are same for every classifier on the pool $\mathcal{H}$ we have to compute the terms including $\boldsymbol{y}_j$ in equation only once during the computation of MMDSel.

Hence, when computing (1) we can calculate the term $\mathbf{E}_{\boldsymbol{y},\boldsymbol{y}' \sim D_2}[k(\boldsymbol{y}, \boldsymbol{y}')]$ completely for all the samples in S beforehand in the preparation phase. When $\mathbf{E}_{\boldsymbol{x} \sim D_1, \boldsymbol{y} \sim D_2}[k(\boldsymbol{x}, \boldsymbol{y})]$ is expressed using (5) this is also the case for parts including $\boldsymbol{y}$. Thus the preparation phase takes

$$O\Big(mt\left(\log k + d^{p'}\right)\Big)$$

time.

Table 1. Summary of asymptotic time and space requirement of MMDSel.

Kernel	Online		Preparation	
	Time	Space	Time	Space
Linear	$O((m+n)d)$	$O((m+n)d)$	$O(mtd)$	$O(mtd)$
Integer polynomial	$O((m+n)d^p)$	$O((m+n)d^p)$	$O(mtd^p)$	$O(m(td+d^p))$
Gaussian (IFGT)	$O\left((m+nk)\,d^{p'}\right)$	$O\left(mnd^{p'}\right)$	$O\left(mt\left(\log k + d^{p'}\right)\right)$	$O\left(mtd^{p'}\right)$
General	$O\left(md(t+n)^2\right)$	$O(d(mt+n))$	—	—
General randomized	$O(md(t+n))$	$O(d(mt+n))$	—	—
Traditional method	mnd	nd	—	—

Finally in the online phase we compute the term $\mathbf{E}_{\boldsymbol{x},\boldsymbol{x}'\sim D_1}[k(\boldsymbol{x},\boldsymbol{x}')]$ and the rest of the term $\mathbf{E}_{\boldsymbol{x}\sim D_1,\boldsymbol{y}\sim D_2}[k(\boldsymbol{x},\boldsymbol{y})]$. This takes time

$$O\Big(\left(m+nk'\right)d^{p'}+nk\Big) \quad .$$

For Gaussian kernels there are also other optimization approaches. For example Lee, Gray and Moore [25] give an approach based on space partitioning trees. Also Herbster [26] presents a simple way to compute additive Gaussian kernels even more efficiently.

A summary of the asymptotic time and space requirement of MMDSel in different settings is presented in Table 1. The randomized general version of MMDSel is based on the randomized linear time computation of MMD proposed by Borgwardt et al. [20].

4 Empirical Evaluation

Let us now evaluate MMDSel experimentally. As a reference approach we use the traditional solution of executing each classifier on the input and selecting the most accurate one. The accuracy of a classifier is computed by counting how many times it predicts the right label given by the data set. The experiments are executed on domains of the UCI machine learning repository and MNIST datasets. MNIST consists of images of handwritten digits from 0 to 9. From the UCI repository we choose the classification datasets with the greatest number of different class labels.

For every dataset with l different labels we create all the $\binom{l}{2}$ different binary classification *tasks*. The labels for each task are changed to ± 1 and a SVM classifier is trained for each task. A sample of size 20% from each of the training sets for MMDSel is stored.

The test examples are divided similarly and for each test the best classifier is chosen. Hence, for each test set there is exactly one trained classifier with the same training distribution. We report how many times different methods select this one correct classifier. The test set size is 20% of the training set size.

Table 2. Accuracy of different methods after at least 1 000 test tasks.

Data set			SVM Classification		MMDSel	
Name	Labels	Dimension	Gaussian	Linear	Gaussian	Linear
`mnist`	10	784	4.3	95.4	**100.0**	**100.0**
`abalone`	14	8	5.8	8.3	**18.2**	10.1
`ecoli`	5	7	11.3	56.3	**78.9**	75.0
`glass`	5	9	38.5	39.5	**46.2**	36.3
`letter`	26	16	77.5	84.2	**100.0**	97.6
`vowel-context`	10	12	38.4	**51.4**	21.6	18.8
`krkopt`	18	6	26.8	25.6	**32.9**	28.0
`led` with 10% noise	10	7	32.9	51.0	**93.1**	87.3
`mfeat`	10	649	9.8	64.6	**91.1**	74.1
`pendigits`	10	16	28.0	76.9	**100.0**	**100.0**
Mean	—	—	27.3	55.3	68.2	62.7

In UCI datasets some labels are removed due to small number of examples. Also some textual values are changed to numerical ones. In addition all the datasets are randomly permuted. Due to efficiency the size of training set is restricted to 1 000 instances.

We have implemented the introduced methods in Matlab with both linear and Gaussian kernel. The kernel width for Gaussian kernels is chosen with the rule-of-thumb

$$\sigma = \sqrt{\text{median distance between points}/2} \ .$$

In the experiments MMDSel chooses the classifier with the smallest distribution discrepancy, regardless of the value. For MMD computation [17] we use the biased estimate of (3), because it is slightly more accurate than the unbiased one of (2). The reported MMDSel versions are tried for unlabeled data because removing labels does not affect the accuracy.

The accuracy of the methods in at least 1 000 separate tasks are reported in Table 2. Each task results in either a right or a wrong answer for a method.

In the overall performance MMDSel prevails. In nearly all of the data sets MMDSel with a Gaussian kernel seems to take the lead with the linear kernel version being slightly worse. However, with the traditional SVM classification the linear kernel version surpasses the Gaussian one.

There are, however, some interesting exceptions. The `vowel` data set includes features of English pronounced vowels. The traditional method is better on this. The `glass` data set contains the chemical composition of different glass types. The traditional method overtakes linear MMDSel version also on this set.

It is worth noting that the setting is quite unfavorable for the classification approach. The behavior of a classifier depends essentially on the hyperplane that is computed to separate training classes. Thus, in the testing phase we are searching for the classifier with a hyperplane separating the test classes. However, because of the test setting there could be multiple hyperplanes that

give similar classification for the test set. Hence, many hypotheses would gain good classification accuracy. This could be partial explanation for the worse performance of the classification method. However, in reality the classification accuracies in our experiments seem to be quite diverse. Usually alone the chosen classifier has the best accuracy.

Gaussian kernel SVM performed surprisingly poorly in these evaluations. For example with the `ecoli` data set the result is only slightly better than a random guess. This is also emphasized with high dimensional data. A natural reason for this is the 'curse of dimensionality' that affects the Gaussian kernel [27]. This appears because of the small number of training examples compared to their dimension. Interestingly MMDSel does not seem to be affected by this. Maybe this is because MMDSel tracks down the underlying distribution instead of single points.

However, in additional experiments optimizing the Gaussian kernel width seemed to improve the SVM performance somewhat. Nevertheless, it did not reach the performance of linear SVM version on the high dimensional data sets. For example with `ecoli` data set the Gaussian SVM attained 44.0% accuracy with a smaller kernel width.

In summary, MMDSel would appear to offer a viable alternative to the traditional method. Only a small stored sample of the training distribution suffices to let us choose the correct classifier with high probability without even knowing the class labels of instances.

5 Conclusions

In this paper we introduced a method of ranking a pool of classifiers by their suitability for the current input. The MMDSel method is based on the similarities of classifier training distributions and the current input distribution. Thus, it is suitable for unsupervised learning. This advantage is, to the best of our knowledge, rare in previous work. Also, classification algorithm outputs are not used and thus the type of algorithms is entirely unlimited. Moreover, the pool may consist of different types of algorithms. We also showed that the test can be computed asymptotically efficiently with some optimization methods.

In our empirical evaluation the MMDSel with both linear and Gaussian kernels seems to be accurate enough to be a viable alternative for solving the given problem at least on some data.

There are some interesting open questions: Why the curse of dimensionality does not seem to affect MMDSel on Gaussian kernel as seen in our experiments? Also, surely different well-founded methods in addition to MMD for finding distribution similarities are possible.

Acknowledgments

This work was supported by Academy of Finland project "ALEA: Approximation and Learning Algorithms." Moreover, the work of T. Aho is financially

supported by Tampere Graduate School in Information Science and Engineering (TISE) and the work of T. Elomaa by Academy of Finland project "Machine Learning and Online Data Structures".

References

1. Zhu, X., Wu, X., Yang, Y.: Effective classification of noisy data streams with attribute-oriented dynamic classifier selection. Knowledge and Information Systems **9**(3) (2006) 339–363
2. Klinkenberg, R.: Learning drifting concepts: Example selection vs. example weighting. Intelligent Data Analysis **8**(3) (2004) 281–300
3. Gama, J., Medas, P., Castillo, G., Rodrigues, P.P.: Learning with drift detection. In Bazzan, A.L.C., Labidi, S., eds.: Advances in Artificial Intelligence — SBIA 2004, 17th Brazilian Symposium on Artificial Intelligence. Volume 3171 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer (2004) 286–295
4. Merz, C.J.: Dynamical selection of learning algorithms. In Fisher, D., Lenz, H.J., eds.: Learning from Data: Artificial Intelligence and Statistics. Volume 112 of Lecture Notes in Statistics. Springer, Berlin, Heidelberg (1996) 281–290
5. Ko, A.H., Sabourin, R., Britto, Jr., A.S.: From dynamic classifier selection to dynamic ensemble selection. Pattern Recognition **41**(5) (2008) 1735–1748
6. Kolter, J.Z., Maloof, M.A.: Dynamic weighted majority: An ensemble method for drifting concepts. Journal of Machine Learning Research **8** (2007) 2755–2790
7. Wang, H., Fan, W., Yu, P.S., Han, J.: Mining concept-drifting data streams using ensemble classifiers. In Getoor, L., Senator, T.E., Domingos, P., Faloutsos, C., eds.: Proceedings of the Ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, New York, NY, ACM Press (2003) 226–235
8. Ali, S., Smith, K.A.: On learning algorithm selection for classification. Applied Soft Computing **6**(2) (2006) 119–138
9. Wolpert, D.H., Macready, W.G.: No free lunch theorems for optimization. IEEE Transactions on Evolutionary Computation **1**(1) (1997) 67–82
10. Watanabe, O.: Sequential sampling techniques for algorithmic learning theory. Theoretical Computer Science **348**(1) (2005) 3–14
11. Wu, X., Chu, C.H., Wang, Y., Liu, F., Yue, D.: Privacy preserving data mining research: Current status and key issues. In Shi, Y., van Albada, G.D., Dongarra, J., Sloot, P.M.A., eds.: Computational Science — ICCS 2007, 7th International Conference. Volume 4489 of Lecture Notes on Computer Science., Berlin, Heidelberg, Springer (2007) 762–772
12. Janssen, F., Fürnkranz, J.: On meta-learning rule learning heuristics. In: Proceedings of the Seventh IEEE International Conference on Data Mining, Los Alamitos, CA, IEEE Computer Society (2007) 529–534
13. Cesa-Bianchi, N., Freund, Y., Haussler, D., Helmbold, D.P., Schapire, R.E., Warmuth, M.K.: How to use expert advice. Journal of the ACM **44**(3) (1997) 427–485
14. Andoni, A., Indyk, P.: Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. In: Proceedings of the 47th Annual IEEE Symposium on Foundations of Computer Science, Los Alamitos, CA, IEEE Computer Society (2006) 459–468
15. Charikar, M.S.: Similarity estimation techniques from rounding algorithms. In: Proceedings of the 34th Annual ACM Symposium on Theory of Computing, New York, NY, ACM Press (2002) 380–388

16. Chan, T.M.: Closest-point problems simplified on the RAM. In: Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms, Philadelphia, PA, SIAM (2002) 472–473
17. Gretton, A., Borgwardt, K.M., Rasch, M., Schölkopf, B., Smola, A.J.: A kernel method for the two-sample-problem. In Schölkopf, B., Platt, J.C., Hoffman, T., eds.: Advances in Neural Information Processing Systems. Volume 19., Cambridge, MA, MIT Press (2007) 513–520
18. Gretton, A., Borgwardt, K.M., Rasch, M., Schölkopf, B., Smola, A.J.: A kernel approach to comparing distributions. In: Proceedings of the Twenty-Second AAAI Conference on Artificial Intelligence, Menlo Park, CA, AAAI Press (2007) 1637–1641
19. Smola, A.J., Gretton, A., Song, L., Schölkopf, B.: A Hilbert space embedding for distributions. In Hutter, M., Servedio, R.A., Takimoto, E., eds.: Algorithmic Learning Theory, 18th International Conference, ALT 2007. Volume 4754 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer (2007) 13–31
20. Borgwardt, K.M., Gretton, A., Rasch, M.J., Kriegel, H.P., Schölkopf, B., Smola, A.J.: Integrating structured biological data by Kernel Maximum Mean Discrepancy. Bioinformatics **22**(14) (2006) 49–57
21. Huang, J., Smola, A.J., Gretton, A., Borgwardt, K.M., Schölkopf, B.: Correcting sample selection bias by unlabeled data. In Schölkopf, B., Platt, J.C., Hoffman, T., eds.: Advances in Neural Information Processing Systems. Volume 19., Cambridge, MA, MIT Press (2007) 601–608
22. Steinwart, I.: On the influence of the kernel on the consistency of support vector machines. Journal of Machine Learning Research **2** (2001) 67–93
23. Raykar, V.C., Duraiswami, R.: The improved fast Gauss transform with applications to machine learning. In Bottou, L., Chapelle, O., DeCoste, D., Weston, J., eds.: Large-Scale Kernel Machines. MIT Press, Cambridge, MA (2007) 175–201
24. Yang, C., Duraiswami, R., Davis, L.S.: Efficient kernel machines using the improved fast Gauss transform. In Saul, L.K., Weiss, Y., Bottou, L., eds.: Advances in Neural Information Processing Systems. Volume 17., Cambridge, MA, MIT Press (2004) 1561–1568
25. Lee, D., Gray, A.G., Moore, A.W.: Dual-tree fast gauss transforms. In Weiss, Y., Schölkopf, B., Platt, J., eds.: Advances in Neural Information Processing Systems. Volume 18., Cambridge, MA, MIT Press (2006) 747–754
26. Herbster, M.: Learning additive models online with fast evaluating kernels. In Helmbold, D.P., Williamson, B., eds.: Computational Learning Theory, 14th Annual Conference, COLT 2001. Volume 2111 of Lecture Notes in Computer Science., Berlin, Heidelberg, Springer (2001) 444–460
27. Bengio, Y., LeCun, Y.: Scaling learning algorithms towards AI. In Bottou, L., Chapelle, O., DeCoste, D., Weston, J., eds.: Large-Scale Kernel Machines. MIT Press, Cambridge, MA (2007) 321–388