Incorporating Prior Knowledge on Features into Learning

Eyal Krupka* **and** **Naftali Tishby**
School of Computer Science and Engineering,
Interdisciplinary Center for Neural Computation
The Hebrew University Jerusalem, 91904, Israel
{eyalkr,tishby}@cs.huji.ac.il

Abstract

In the standard formulation of supervised learning the input is represented as a vector of features. However, in most real-life problems, we also have additional information about each of the features. This information can be represented as a set of properties, referred to as *meta-features*. For instance, in an image recognition task, where the features are pixels, the meta-features can be the (x, y) position of each pixel. We propose a new learning framework that incorporates meta-features. In this framework we assume that a weight is assigned to each feature, as in linear discrimination, and we use the meta-features to define a prior on the weights. This prior is based on a Gaussian process and the weights are assumed to be a smooth *function* of the meta-features. Using this framework we derive a practical algorithm that improves generalization by using meta-features and discuss the theoretical advantages of incorporating them into the learning. We apply our framework to design a new kernel for handwritten digit recognition. We obtain higher accuracy with lower computational complexity in the primal representation. Finally, we discuss the applicability of this framework to biological neural networks.

1 Introduction

Incorporating prior knowledge about the representation of objects has long been known to profoundly influence the effectiveness of learning. This has been demonstrated by many authors using various heuristics such as specially engineered architectures or distance measures (see e.g., LeCun et al., 1998, Simard et al., 1993). In the context of support vector machines (SVM) a variety of successful methods to incorporate prior knowledge have been published over the last ten years (see e.g., Decoste and Scholkopf, 2002 and a recent review by Lauer and Bloch, 2006). Since the nature of prior knowledge is different from task to task, many of these methods are task-specific. For example, one of the methods applied in digit recognition is to increase the training set size by creating virtual examples, i.e. new instances are created using transformations on the original instances. Although this method works well for digit recognition, it is not clear how to use it in other tasks such as text classification. Since prior knowledge may be available in very different forms, it is not clear how to develop a general approach that is suitable to *any* learning problem.

In our setup, we focus on learning problems where prior knowledge on each feature can be represented directly by a vector of meta-features. For example, in collaborative filtering, where the features are movie ratings, the meta-features represent information about the movie, e.g. genre and year produced. We represent the possible mappings between meta-features and weights by a class of functions. In general, we expect these functions to be smooth, e.g. movies with similar meta-features (genres) tend to have similar weights. Therefore, we can define a prior on the mappings from the meta-features to the weight that "prefers" smooth functions. Such a prior induces a prior on the weights which leads to better generalization.

As a simple example, consider a two-class problem of handwritten digit recognition, where the instances are represented by vectors of pixel gray levels. We use a linear discriminator: $h_{\mathbf{w}}(x) = \text{sign}\left(\mathbf{w}^T\mathbf{x}\right)$. Assuming the data are linearly separable, the maximal-margin approach of linear SVM selects the weight vector with a minimal L2 norm, $\|\mathbf{w}\|^2$, under the constraint $\left(\mathbf{w}^T\mathbf{x}_i\right) y_i \geq 1$, for all instances in the training set $\{\mathbf{x}_i, y_i\}_{i=1}^m$, where $y_i \in \{\pm 1\}$ is the label. Here however, we want to exploit additional information -

* Intel Research Israel. eyal.krupka@intel.com

the position of the pixel. This position is represented as a 2-element vector of meta-features (row, column) for each of the pixels. In general, we expect adjacent pixels to have similar weights. In other words, we expect the weights to be a smooth function of the meta-features. One way to achieve this is to use a Gaussian prior on $\mathbf{w}$, defined by a covariance matrix, $\mathbf{C}$. To encourage smoothness, the covariance between a pair of weights is taken to be a decreasing *function* of the distance between their meta-features, i.e. the corresponding pixel positions. This *covariance function* defines a Gaussian process on the meta-feature space, which encourages shift invariance. Then, instead of selecting the minimal norm $\|\mathbf{w}\|^2$, we select $\mathbf{w}$ with a minimal weighted norm, $\mathbf{w}^T\mathbf{C}^{-1}\mathbf{w}$ (i.e. maximum likelihood) under the same constraint $\left(\mathbf{w}^T\mathbf{x}_i\right) y_i \geq 1$. Obviously, performance can be improved by using richer features and meta-features. In section 3 we show that using monomial features with meta-feature based feature selection and a prior on weights can improve accuracy compared to other monomial-based kernels. Moreover, although our classifier does not use the "kernel trick", it requires significantly less memory and has lower computational complexity when classifying test instances compared to other designed kernels (see e.g., Scholkopf et al., 1998).

Our key motivation for explicitly defining and using meta-features is as follows. By representing prior knowledge on features as vectors of meta-features, we can develop common tools and theoretical results to incorporate prior knowledge into a variety of learning tasks. In this paper we introduce one of these tools, which is based on kernel design by defining a Gaussian process on the meta-feature space. This tool is demonstrated on a digit recognition task, but can easily be adapted to other learning tasks. We also analyze the theoretical advantages of using meta-features.

In section 5 we discuss the possible relation of our framework to biological neural networks in the cortex, where meta-features are represented by the spatial position of the neurons in the cortex.

2 Kernel design by meta-features

In this section we use meta-features to define a Gaussian process prior on the meta-features to weights mapping. Consider a training set $\{\mathbf{x}_i, y_i\}_{i=1}^m$, where y_i is the label and $\mathbf{x}_i \in \mathbb{R}^d$ can be the raw input variable or features extracted from the inputs. Each of the d elements in the vector $\mathbf{x}$ is a feature. Let $\{\mathbf{u}_1, \ldots, \mathbf{u}_d\}$ be vectors of meta-features, where $\mathbf{u}_j \in \mathbb{R}^k$ are the meta-features of the jth feature. A key point is that the value of the meta-features is fixed per feature, i.e. it is not dependent on the instances. Therefore, we refer to the meta-features as prior knowledge.

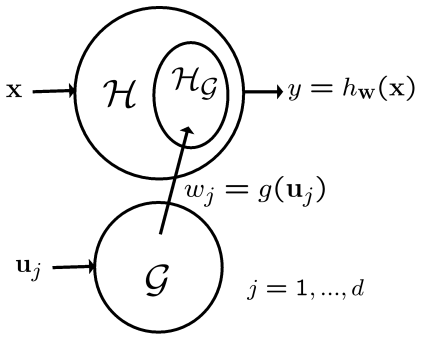

Figure 1: The hypothesis space of the basic setup. $\mathcal{H}$ is the hypothesis space of the standard learning problem, parametrized by $\mathbf{w}$. In our setup, we define a class of mappings, $\mathcal{G}$, from a vector of meta-features $\mathbf{u}$, to the weight of the corresponding feature. Given the values of the meta-features of all the features $\{\mathbf{u}_j\}_{j=1}^d$, the mapping $g(\mathbf{u}) \in \mathcal{G}$ indirectly defines a hypothesis $h \in \mathcal{H}$ by its weights $\{w_j\}_{j=1}^d$. Moreover, a prior on the mappings in $\mathcal{G}$, indirectly defines a prior on $\mathcal{H}$, i.e. on $\mathbf{w}$. As discussed in Section 4, $\mathcal{G}$ can also be used to define a subclass $\mathcal{H}_\mathcal{G} \in \mathcal{H}$, and hence reduce the hypothesis space.

Let $\mathcal{H}$ be the class of homogeneous linear discrimination functions, where each function in the class $h_\mathbf{w} \in \mathcal{H}$ is defined by the weight vector $\mathbf{w}$, i.e. $h_\mathbf{w}(\mathbf{x}) = \text{sign}\left(\mathbf{w}^T\mathbf{x}\right)$. In order to use the meta-features, we view the weights as a function of the meta-features. Let $\mathcal{G}$ be the class of meta-features to weights mapping, then for $g \in \mathcal{G}$, each weight satisfies $w_j = g(\mathbf{u}_j)$ $(j = 1, \ldots, d)$. The prior on $\mathbf{w}$ is defined by a prior on the functions in $\mathcal{G}$, i.e. the prior on meta-features to weight mapping. This is illustrated in Figure 1.

A learning algorithm uses the training set to select $\mathbf{w}$. Assuming that the training set is linearly separable, there exists an infinite set of $\mathbf{w}$ such that all instances in training set $\left(\mathbf{w}^T\mathbf{x}_i\right) y_i \geq 1$. In the standard approach of linear SVM, we solve the following optimization problem

$$\mathbf{w} = \underset{\mathbf{w}:(\mathbf{w}^T\mathbf{x}_i)y_i \geq 1,\, i=1,\ldots,m}{\text{argmin}} \|\mathbf{w}\|^2 \qquad (1)$$

This can be viewed as finding the maximum-likelihood hypothesis, under the above constraint, where we have a Gaussian prior on $\mathbf{w}$

$$P(\mathbf{w}) \propto e^{-\frac{1}{2}\mathbf{w}^T\mathbf{C}^{-1}\mathbf{w}} \qquad (2)$$

and the covariance matrix $\mathbf{C}$ equals the unit matrix, i.e. all weights are assumed to be independent and have the same variance. However, we can exploit the meta-features in order to create a better prior on

Table 1: Meta-features for various learning tasks.

Application	Features	Meta-features	Notes
Handwritten recognition	Monomials	Position, orientation, geometric shape descriptors	See Section 3
Collaborative filtering (movies)	Vector of movie ratings per use	Movie genres, year, actors	A detailed experiment is available in [Krupka and Tishby, 2007]
Text classification	Words (bag of words)	Stem, semantic meaning, synonym, co-occurrence, part of speech, is stop word	Covariance is determined by relation between words that is represented by meta-features. Variance by type (e.g. stop word)

$\mathbf{w}$. For this purpose we define the covariance between weight i and j as a function of the meta-features associated with features i and j, i.e.

$$\mathbf{C}_{ij} = C(\mathbf{u}_i, \mathbf{u}_j) \tag{3}$$

where C is a covariance *function*. In general, we expect weights of features with similar meta-features to be similar. Therefore, the covariance function should be some decreasing function of the distance between $\mathbf{u}_i$ and $\mathbf{u}_j$. Note that the prior defined by equations 2 and 3 is a *Gaussian process* prior on the function class $\mathcal{G}$, i.e. the weights can be viewed as a Gaussian process in the meta-feature space. Therefore, we can use known results from the literature of Gaussian processes (see e.g., Williams and Rasmussen, 1996) to define a valid covariance function, i.e. guarantee that the result covariance matrix is positive semi-definite.

Once the covariance function is defined, we need to solve the following equation

$$\mathbf{w} = \underset{\mathbf{w}:(\mathbf{w}^T\mathbf{x}_i)y_i \geq 1,\ i=1,\ldots,m}{\operatorname{argmin}} \mathbf{w}^T\mathbf{C}^{-1}\mathbf{w} \tag{4}$$

this can be done by mapping the original input by defining $\tilde{\mathbf{x}}_i = \mathbf{C}^{1/2}\mathbf{x}_i$ $(i = 1, ..., m)$, and solving the standard linear SVM optimization for $\tilde{\mathbf{w}}$

$$\tilde{\mathbf{w}} = \underset{\tilde{\mathbf{w}}:(\tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_i)y_i \geq 1,\ i=1,\ldots,m}{\operatorname{argmin}} \|\tilde{\mathbf{w}}\|^2 \tag{5}$$

where $\mathbf{w} = \mathbf{C}^{1/2}\tilde{\mathbf{w}}$. Equations 5 and 4 are equivalent since $\mathbf{w}^T\mathbf{x}_i = \tilde{\mathbf{w}}^T\mathbf{C}^{1/2}\mathbf{C}^{-1/2}\tilde{\mathbf{x}}_i = \tilde{\mathbf{w}}^T\tilde{\mathbf{x}}_i$ and $\mathbf{w}^T\mathbf{C}^{-1}\mathbf{w} = \|\tilde{\mathbf{w}}\|^2$. This means that our solution is also equivalent to using kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T\mathbf{C}\mathbf{x}_j$. Since this kernel is based on linear transformation we can save memory and computations at classification time. By transforming the result weights using $\mathbf{w} = \mathbf{C}^{1/2}\tilde{\mathbf{w}}$, we can directly calculate $\mathbf{w}^T\mathbf{x}$ for each new instance in the test set without having to keep the support vector.

We showed that the prior on the weights in eq. 2 can be achieved by multiplying $\mathbf{x}$ by $\mathbf{C}^{1/2}$. Using other considerations, Scholkopf et al. [1998] proposed multiplying the input by a matrix $\mathbf{B}$, where $\mathbf{B}$ is optimized to achieve invariance to local transformations. In our setup, the invariance is incorporated by the assumption of smoothness of weights in the meta-feature space. The advantage of this approach is that it can be applied to other applications, e.g. collaborative filtering, where "local transformations" are not defined, but meta-features that describe features can easily be defined (see Table 1).

The link between Gaussian process and SVM has been pointed out by a number of authors (see e.g., Sollich, 2002). In previous works, the Gaussian process is on the mapping $\mathbf{x} \rightarrow y$ and therefore reflects the smoothness of y (or $P(y|\mathbf{x})$) in the *feature* space. The key difference is that in our work, the Gaussian process is on the mapping $\mathbf{u} \rightarrow w$, i.e. the meta-features to weights mapping. The advantage of this approach is that we incorporate the prior knowledge available in the meta-features.

2.1 A toy example

We first apply this method to a toy problem of a binary classifier for handwritten digit recognition that learns from two instances (one for each label). For this purpose we use the MNIST dataset [LeCun et al., 1998]. Since the training set is extremely small, direct usage of the gray level of pixels as features is good enough, i.e. we do not need to use feature extraction. The meta-features, $\mathbf{u}$, are the (row, column) position of each feature. The covariance function we use is

$$C(\mathbf{u}_i, \mathbf{u}_j) = e^{-\frac{1}{2\sigma^2}\|\mathbf{u}_i - \mathbf{u}_j\|^2} \tag{6}$$

where $\mathbf{u}_i$, $\mathbf{u}_j$ are the meta-features of the weights related to features i and j. σ is a parameter. This prior reflects our expectation of smoothness of the weights as a function of the feature position, and indirectly encourages shift invariance. The accuracy improved from 82% of 88% relative to max-margin. Figure 2 shows the result weights per pixel after training with the standard linear max-margin approach, compared

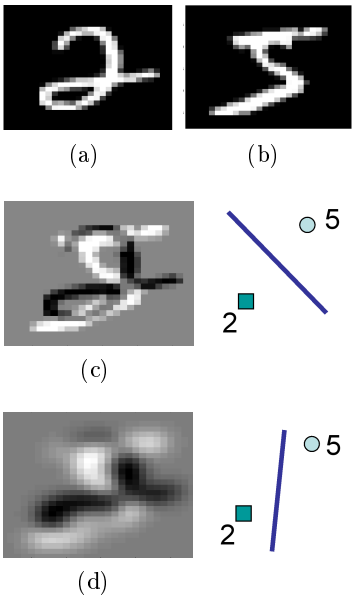

Figure 2: A toy example. Result weights of meta-features based prior vs. max-margin when training from two instances. (a,b) Gray level of the two digits used for training. (c) The weight per pixel of max-margin classifier (white is positive, black is negative). (d) The weights when using a Gaussian process prior on meta-features to weight mapping. The accuracy improved from 82% to 88% relative to max-margin.

to using a Gaussian process prior of meta-features to weights mapping. In the case of a larger training set we need to use richer features and meta-features, as done in Section 3.

2.2 Discussion

At this point one may wonder about the advantage of defining and using the meta-features explicitly. After all, there are other methods to incorporate knowledge of shift-invariance in an image. For example, we can find invariant hyperplanes [Scholkopf et al., 1998] or smooth the input images by some filter. The answer is two-fold. First, our framework is more general, and can be used for more sophisticated features, meta-features and priors. This is demonstrated in section 3 for a more realistic problem of handwritten digit recognition. Actually, in the above toy example, filtering the input image by 2-D convolution with a Gaussian kernel can be considered a *special case* of our framework, since it can be derived from the solution of $\tilde{\mathbf{x}}_i = \mathbf{C}^{1/2}\mathbf{x}_i$, where $\mathbf{C}$ is built using eq. 6. The second, and more important part of the answer is that our method can be successfully addressed to other types of learning problems where concepts such as filtering or transforming the instances do not apply but meta-features can be used (see Table 1). One key aim is to use similar mechanism of incorporating prior knowledge for different tasks.

3 Handwritten digit recognition aided by meta-features

In kernel methods, we map the input vectors into a high-dimensional feature space using a kernel function $K(\mathbf{x}_i, \mathbf{x}_j)$. The kernel trick (see e.g., Vapnik, 1998) enables us to implement a linear classifier in a high or even infinite-dimensional space of features, without explicitly calculating the features. The memory size and required number of operations at test time is in order of the number of support vectors times the number of raw inputs, instead of the number of features in the high-dimensional space. This method has proven itself to be very successful on a wide range of applications. However, using the kernel trick does not enable us to "freely" select or design the features we use. In many cases, this is not a problem, since "feature design" can be replaced by kernel design. For the task of handwritten recognition, Scholkopf et al. [1998] designed a kernel that improves generalization by using local correlations between pixels.

We propose a new way to apply SVM to handwritten digit recognition. First, we *explicitly* calculate a relatively small subset of the monomials of the polynomial kernel. This subset is selected by defining meta-features for the candidate monomials, and selecting a "good" subset by the value of their meta-features. Second, we apply our framework to build a prior on the weights, based on the meta-features of the monomial features. Perhaps surprisingly, in addition to achieving better accuracy relative to other specially-designed kernels, our classifier requires significantly less memory and computations per test instance relative to other kernels which are based on the dual representation. This is because our approach achieves good performance with a relatively small number of features (monomials), and we do not have to keep the support vectors.

Outline. The proposed method of incorporating meta-features to "standard" learning can be summarized as follows:

1. Define simple features and meta-features (Fig. 3). Meta-features should be informative on feature relevance or relations between features.
2. Select a subset of features by their meta-features.
3. Define a Gaussian process prior of meta-features to weights mapping, i.e. covariance function $C(\mathbf{u}_i, \mathbf{u}_j)$ (eq. 7).

Table 2: List of features used (for 3 inputs).

base len. (u_r)	Orientations (u_ϕ)	Pos. (u_x, u_y)
2	$0^o, 45^o, \ldots, 315^o$	20×20 area
4	$0^o, 22.5^o, \ldots, 337.5^o$	20×20 area
6	$0^o, 22.5^o, \ldots, 337.5^o$	20×20 area

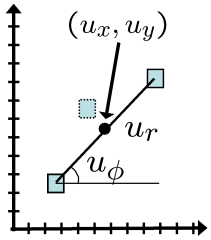

Figure 3: Features used in the experiments. Each feature is a monomial, multiplying the gray level of two or three pixels. The input pixels of a monomial form an isosceles triangle with its height equal to a quarter of its base (for two inputs, the middle input is omitted). Each feature is described by five meta-features $(u_t, u_r, u_\phi, u_x, u_y)$. u_t is the number of inputs (2 or 3). u_r is the distance between the two pixels at the base of the triangle. u_ϕ is the orientation of the base. (u_x, u_y) is the position of the feature in the 28×28 image.

4. Build the covariance matrix $\mathbf{C}$ and solve SVM by kernel $K(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{C} \mathbf{x}_j$ or linear transformation $\tilde{\mathbf{x}}_i = \mathbf{C}^{1/2}\mathbf{x}_i$ (eq. 4, 5). Calculate result weights ($\tilde{\mathbf{w}}$) and assign $\mathbf{w} = \mathbf{C}^{1/2}\tilde{\mathbf{w}}$. Only $\mathbf{w}$ is required for the classifier (test time).

5. Optimize selected features by tuning value of meta-features (e.g. using cross validation). The concept of feature selection by their meta-features appears in Krupka et al. [2006].

Experimental setup. We use the MNIST [LeCun et al., 1998] dataset which contains images of 28×28 pixels of centered digits. Each pixel was normalized to range $[-1, 1]$. We randomly selected 5000 out of 60000 instances from the MNIST training set, and checked the performance using the MNIST test set, which includes 10000 instances. We repeated the experiment with four different randomly selected training sets. The reported results are the average of these selections. As a general reference, we used multi-class SVM [Crammer and Singer, 2001] with a polynomial kernel of degree 4, which is the optimal degree, and obtained an error rate of 4.2%. The feature space of this kernel includes $O(10^{11})$ features, where each feature is a monomial (multiplication of up to 4 pixels).

Features and meta-features. We use the same type of features as the polynomial kernel uses indirectly. However, in order to build a prior on the weights, we need to explicitly calculate the features. Obviously, we cannot use all the $O(10^{11})$ features; therefore we selected a small subset of them. Since each feature is a monomial, it is defined uniquely by the position of its inputs. We used monomials with two or three inputs. The meta-features of two input monomials are the distance between the two inputs, the orientation and the center position. The three input monomials we used are defined by the same meta-features as the two input monomials, but with an additional pixel between the two pixels, forming an isosceles triangle . These features and meta-features are described in Figure 3.

The meta-features of the features we used are described in Table 2. Our initial selection was determined somewhat arbitrarily, and only the height of the triangle[1] and maximum length ($u_r \leq 6$) were tuned by a simple search, at coarse resolution (using cross-validation). The total number of features with 3 inputs is $40 \times 20 \times 20 = 16000$ (40 stands for the u_r and u_ϕ, and 20×20 for the center positions). Note that for two inputs, we get the same feature for a rotation of 180^o; therefore the number of features with two inputs is only 8000. We have a total of 24000 features.

The learning process. As previously, the prior on weights is determined by the covariance function C. First we use the simple covariance function

$$C(\mathbf{u}_i, \mathbf{u}_j) = \begin{cases} e^{-\frac{1}{2\sigma_P^2}\|\mathbf{u}_i - \mathbf{u}_j\|^2} & \text{if features } i,j \text{ have} \\ & \text{the same value of } u_t, u_r, u_\phi \\ 0 & \text{otherwise} \end{cases} \tag{7}$$

where σ_P is a parameter. Note that we assume that the weights across features with different sizes, orientations or number of inputs are uncorrelated. The correlation is determined by the distance between the center position of the features. Options for more sophisticated priors on weights are discussed below. The resulting covariance matrix $\mathbf{C}$ is a block diagonal, with 60 identical blocks of size 400×400. Hence, $\mathbf{C}^{1/2}$ can be calculated efficiently[2]. Note also that $\tilde{x}$ only needs to be calculated for training instances.

The hyper-parameter σ_P can be optimized using cross-

[1] The height of the triangle is also a meta-feature, but we use the same height for all features (can be considered "meta-features sharing").

[2] Since the function C is Gaussian and stationary it can be shown (using Fourier analysis) that the solution of $\mathbf{C}^{1/2}\mathbf{x}$ is equivalent to 2D convolution of each of the 20×20 "maps" in feature space, with a Gaussian kernel with a standard deviation equal to $\sqrt{\sigma_P}$. Hence, $\tilde{x}$ can be calculated efficiently without explicitly calculating $\mathbf{C}^{1/2}\mathbf{x}$.

validation. However, the performance is not sensitive to the exact value of this parameter. We achieved nearly the same performance for $4 \leq \sigma_P \leq 9$. The reported results are for $\sigma_P = 6$. After calculating $\tilde{x}$, the optimization was solved using a multi-class linear SVM [Crammer and Singer, 2001]. Since it is a linear SVM, it can also be solved online without calculating the kernel matrix [Shalev-Shwartz and Singer, 2006].

Results. The results are summarized in Table 3. The error range is about $\pm 0.1\%$ and includes the results of all four random selections of the training set. A significant performance improvement relative to the 4th degree polynomial SVM was achieved just by using the subset of monomials (4.2% to 2.9%). The overall improvement of our method (4.2% to 1.8%) is slightly better than what was reported by Scholkopf et al. [1998] for a combined method of virtual examples (but only with translations) and a specially designed kernel (4% to 2%)[3]. We believe that further research regarding the choice of features and the covariance function may lead to even better results. Since we calculate the features explicitly, we can choose any type of feature, and are not restricted to monomials. In addition, we can also incorporate methods of creating virtual examples.

For a training set of 5000 instances, our method needs about 80% less memory and computations to predict labels of new instances (after training) relative to polynomial SVM. The computational advantage is even greater when compared to specially designed kernels in the dual representation [Scholkopf et al., 1998] and use of virtual examples which may increase the number of support vectors. Moreover, a preliminary experiment shows that we can significantly reduce the number of features with only a slight drop in accuracy. However, in order to apply our method to larger training sets with significantly improved performance we need to use (and design) more features. This is a disadvantage of our approach compared to most other kernel methods which scale easier with the size of training set by increasing the number of support vectors. Further research is required to determine the number of required features vs. training set size. In particular, we need to compare it to the increase in the number of support vectors. An interesting question is whether the computational advantage of our approach relative to methods that work on the dual representation increases or decreases for large training sets.

Discussion. The covariance function in eq. 7 exploits only part of the prior knowledge available in the meta-features. We can incorporate small orientation invariance and the effect of base length and number of inputs on the variance by multiplying the covariance function in eq. 7 by

$$A_t(u_t)\, A_r(u_r) e^{-\frac{(\Delta\phi_{ij})^2}{2\sigma_\phi^2}}$$

where $\Delta\phi_{ij}$ is the orientation difference between the features i and j. $A_t(u_t)$ is a function of the number of inputs, that can control the balance between 2 and 3 input monomials. Similarly, $A_r(u_r)$ can control the prior variance as a function of distance between inputs (u_r in Figure 3); we expect large triangles to have smaller weights. We found that the contribution of adding these terms is not significant relative to the contribution of the prior in eq. 7. However, for small sizes of training sets, the improvement is significant.

[3] They used the same size of training and test sets as we did. However, they did not report which of the instances they used for training or for test.

Table 3: Test performance on MNIST (with 5000 training instances)

Classifier	Test Err
Multiclass SVM, 4th degree polynom	4.2%
Linear SVM on a subset of monomials	2.9%
Gaussian process on meta-feature space	1.8%
Local kernel and virtual SVM [Scholkopf et al., 1998]	2.0%

4 Towards a theory of meta-features

In this section we analyze the theoretical advantage of using meta-features. We focus on a simple case, where the meta-features are used to reduce the hypothesis space, and some special cases of mappings from meta-features to weights. For example, if we allow only smooth mappings from meta-features to weights, only a subset of the hypotheses in $\mathcal{H}$ can be used. We denote this subset by $\mathcal{H}_\mathcal{G}$ (see Figure 1). Although the proposed theorems are simple, they illustrate the advantages of using meta-features.

We compare the VC-dimension (VCD) [Vapnik, 1998] of $\mathcal{H}_\mathcal{G}$ relative to the VC-dim of $\mathcal{H}$. Without using the meta-features, VCD ($\mathcal{H}$) is the number of *features*. Our results show that for a large number of features, the complexity of mappings from meta-features to weights ($\mathcal{G}$) is more important than the number of features. This is especially valuable when one can measure or generate many features (feature extraction), and hence increase the complexity of $\mathcal{H}$, while keeping the complexity of $\mathcal{G}$ low. The next theorem states that when $\mathcal{G}$ is the class of linear regression functions, VCD($\mathcal{H}_\mathcal{G}$) is upper bounded by the number of *meta-features*.

Theorem 1 *Let $\mathcal{H}$ be the class of linear discrimination functions on $\mathbb{R}^d$. We assume that the hyperplane passes through the origin, i.e. $\forall h \in \mathcal{H}$, $h_{\mathbf{w}}(\mathbf{x}) = sgn\left(\mathbf{w}^T\mathbf{x}\right)$. Let $\mathcal{G}$ be the class of homogeneous linear regression functions from meta-features to weights $\mathbb{R}^k \rightarrow \mathbb{R}$, i.e. $\forall g \in \mathcal{G}$, $g(\mathbf{u};\mathbf{c}) = \mathbf{c}^T\mathbf{u}$, where $\mathbf{c}$ is a vector of k parameters. Then $VCD(\mathcal{H}_{\mathcal{G}}) \leq \min(k,d)$.*

Proof: Let $\mathbf{U}$ be a $k \times d$ matrix, where the jth column is $\mathbf{u}_j$, i.e. the vector of meta-features of the jth feature. By definition, $\mathbf{w} = \mathbf{U}^T\mathbf{c}$ and hence we get $h_{\mathbf{w}}(\mathbf{x}) = \text{sgn}\left(\mathbf{c}^T\tilde{\mathbf{x}}\right)$, where $\tilde{\mathbf{x}} = \mathbf{U}\mathbf{x}$. Since $\tilde{\mathbf{x}} \in \mathbb{R}^k$, we get an equivalent linear discrimination problem with k parameters. Therefore $\text{VCD}\left(\mathcal{H}_{\mathcal{G}}\right) \leq k$. Since $\mathcal{H}_{\mathcal{G}} \in \mathcal{H}$ we also get $\text{VCD}\left(\mathcal{H}_{\mathcal{G}}\right) \leq \text{VCD}\left(\mathcal{H}\right) = d$. □

From theorem 1 we can conclude that our theoretical gain from using meta-features is significant when the number of meta-features is small relative to the number of features. The next corollary generalizes Theorem 1 to a richer class of functions $\mathcal{G}$.

Corollary 1 *Using the above definition, but $\mathcal{G}$ here is the class of linear regression functions from a set of t fixed functions $\psi_1, ..., \psi_t$ of $\mathbf{u}$*

$$g(\mathbf{u}) = \sum_{j=1}^{t} c_j \psi_j(\mathbf{u})$$

where $c_1, ..., c_t$ are the parameters that define g. Then $VCD(\mathcal{H}_{\mathcal{G}}) \leq \min(t,d)$.

In the next Theorem, we use corollary 1 to relate the smoothness of the functions in $\mathcal{G}$ to $\text{VCD}\left(\mathcal{H}_{\mathcal{G}}\right)$. Our measure of smoothness is the bandwidth of the functions in $\mathcal{G}$. The idea is rather simple – consider the case where the number of features is very large, but the features are crowded in the meta-feature space. If the meta-features to weights mapping is smooth, then the weights depend strongly on each other. Therefore $\text{VCD}\left(\mathcal{H}_{\mathcal{G}}\right)$ is determined by the smoothness (bandwidth) of g and the volume in the meta-feature space where the features reside.

Theorem 2 *Let $\mathcal{H}$ be the class of linear discrimination functions on $\mathbb{R}^d$, where the hyperplane passes through the origin. Let $\mathcal{G}$ be the class of periodic functions of the meta-features, where the period in the rth dimension is T_r, i.e. $g(u_{j1}, ..., u_{jr}, ..., u_{jk}) = g(u_{j1}, ..., u_{jr} + T_r, ..., u_{jk})$ $(\forall g \in \mathcal{G}, \forall \mathbf{u}_j)$. We further assume that for all $g \in \mathcal{G}$, the bandwidth of g is limited as follows,*

$\hat{g}(f_1, f_2, ..., f_k) = 0$ *if there exists* r *such that* $|f_r| > B_r$

for some constants $B_1, ..., B_k$, where $\hat{g}$ is the multi-dimensional continuous Fourier transform of g and $f_1, ..., f_k$ are the frequencies. Then

$$VCD\left(\mathcal{H}_{\mathcal{G}}\right) \leq \prod_{r=1}^{k} \left(2B_rT_r + 1\right)$$

Proof: There are at most $t = \prod_{r=1}^{k} \lceil 2B_rT_r + 1 \rceil$ non-zero Fourier coefficients for all $g \in \mathcal{G}$. Let the set of fixed functions defined in theorem 1 $\{\psi_1, ..., \psi_t\}$ be the (sin / cos) functions corresponding to the Fourier basis functions that can have a non-zero coefficient. Any $g \in \mathcal{G}$ is a linear sum of these functions. Using corollary 1 we get theorem 2. □

Note that the requirement that g must be a periodic function is not a significant limit on the function class. Assuming the value of meta-features is bounded, we can choose T_r such that all our meta-feature vectors are within a half period of g. However, $\text{VCD}\left(\mathcal{H}_{\mathcal{G}}\right)$ increases with T_r.

The algorithmic application of Theorem 2 is that we can reduce the dimensionality by representing the features in the frequency domain of the meta-feature space, and discard features that represent high frequencies. This dimensionality reduction is different from principle component analysis (PCA), since it is not based on the joint statistics of the features (empirical covariance matrix) but on a fixed prior information (the meta-features).

5 Discussion

We introduce a new framework of incorporating knowledge about features into learning by defining a prior on the mapping from meta-feature space to weights. We show how the prior information can be incorporated using standard classification methods such as SVM for primary learning and Gaussian processes for the meta-features to weights mapping. We further show that a smoothness assumption in meta-feature space ($\mathbf{u} \rightarrow w$) captures domain knowledge such as shift invariance better than the standard assumption of smoothness in the feature space ($\mathbf{x} \rightarrow y$). In addition, the meta-features are also informative about the relevance of features, and can be used for defining a prior on the weight variance and for feature selection (see also Krupka et al., 2006). The theoretical advantage of using meta-features was shown by relating the smoothness in meta-feature space to the VC-dimension. We demonstrated the use of meta-features for handwritten digit recognition, and showed that we can achieve competitive accuracy and lower computational complexity with respect to other designed kernels.

The framework is applicable to other types of applications. We evaluated our framework on a dataset of col-

laborative filtering, where the goal is to predict movie ratings, and showed that we can improve accuracy of prediction. Due to lack of space, this experiment is not included here, but is available (see appendix in Krupka and Tishby, 2007). In Table 1 we list candidate meta-features for some learning tasks. Another potential application is gene-expression. In this case the gene-expression is the feature, and the meta-features represent our knowledge about each gene.

Our framework may also be related to learning in biological neural networks by using the following assumptions. First, feature values are represented by neuron activity. Second, neurons in the cortex are spatially organized by the value of *meta-features* related to each neuron, i.e. neurons which represent similar features are proximate, where similarity of features refers to proximity in meta-feature space. This assumption is supported by the existence of organization maps in the cortex, e.g. in the organization of the primary visual cortex the meta-features can be position, orientation and scale (see e.g., Miikkulainen et al., 2005). The third assumption is that the learning rule of updating weights of a neuron depends not only on the activity of the input neurons, but also on their relative spatial position, such that weight update to proximate input neurons tend to be correlated (this is the online implementation of the smoothness of weights prior). There is evidence for this type of learning in the brain, for example Engert and Bonhoeffer [1997] found that weights (synaptic strength) to input neurons in close spatial proximity tend to be potentiated together, even if some of the input neurons are not active.

Meta-features are used differently in the work of Raina et al. [2006] (although they do not use this term). They showed how to transfer knowledge about words from task to task by using the properties of the words (meta-features in our context). Another way of defining and using meta-features was proposed by Taskar et al. [2003]. In their setup they use meta-features to learn the role of words (features) that are unseen in the training set, but appear in the test set. In their work the features are words and the meta-features are words in the neighborhood of that word. Another framework of learning from observed features to unobserved features is described in Krupka and Tishby [2005] in the context of clustering. The commonality between this paper and the works above lies in extending learning from the standard instance-label framework to learning in the feature space, i.e. from feature to feature.

Acknowledgments

We thank Amir Navot, Ran Gilad-Bachrach, Idan Segev and Shai Shalev-Shwartz for helpful discussions. We also thank Koby Crammer for the use of code for Multiclass SVM [Crammer, 2003].

References

K. Crammer. MCSVM_1.0: C Code for Multiclass SVM. 2003. http://www.cis.upenn.edu/~crammer.

K. Crammer and Y. Singer. On the algorithmic implementation of multiclass kernel-based vector machines. *Journal of Machine Learning Research*, 2001.

D. Decoste and B. Scholkopf. Training invariant support vector machines. *Machine Learning*, 46(1-3):161–190, 2002. ISSN 0885-6125.

F. Engert and T. Bonhoeffer. Synapse specificity of long-term potentiation breaks down at short distances. *Nature*, 1997.

E. Krupka and N. Tishby. Generalization in clustering with unobserved features. In *NIPS*, 2005.

E. Krupka and N. Tishby. Incroporating prior knowledge on features into learnning. Technical report, Leibniz Center, the Hebrew University, 2007. http://leibniz.cs.huji.ac.il/tr/937.pdf.

E. Krupka, A. Navot, and N. Tishby. Learning to select features using their properties. Technical report, Leibniz Center, the Hebrew University, 2006. http://leibniz.cs.huji.ac.il/tr/926.ps.

F. Lauer and G. Bloch. Incorporating prior knowledge in support vector machines for classification: a review. *Submitted to Neurocomputing*, 2006.

Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, November 1998.

R. Miikkulainen, J. A. Bednar, Y. Choe, and J. Sirosh. *Computational Maps in the Visual Cortex*. Springer, Berlin, 2005.

R. Raina, A. Y. Ng, and D. Koller. Constructing informative priors using transfer learning. In *ICML*, 2006.

B. Scholkopf, P. Simard, A. Smola, and V. Vapnik. Prior knowledge in support vector kernels. In *NIPS*, 1998.

S. Shalev-Shwartz and Y. Singer. Efficient learning of label ranking by soft projections onto polyhedra. *Journal of Machine Learning Research*, 2006.

P. Y. Simard, Y. A. Le Cun, and Denker. Efficient pattern recognition using a new transformation distance. In *NIPS*. 1993.

P. Sollich. Bayesian methods for support vector machines: Evidence and predictive class probabilities. *Machine Learning*, 46, 2002.

B. Taskar, M. F. Wong, and D. Koller. Learning on the test data: Leveraging unseen features. In *ICML*, 2003.

V. N. Vapnik. *Statistical Learning Theory*. Wiley, 1998.

C. K. I. Williams and C. E. Rasmussen. Gaussian processes for regression. In *NIPS*, 1996.