Minimum Risk Feature Transformations

Shivani Agarwal SAGARWAL@CS.UIUC.EDU
Dan Roth DANR@CS.UIUC.EDU
Department of Computer Science, University of Illinois, Urbana, IL 61801 USA

Abstract

We develop an approach for automatically learning the optimal feature transformation for a given classification problem. The approach is based on extending the principle of risk minimization (RM), commonly used for learning classifiers, to learning feature transformations that admit classifiers with minimum risk. This allows feature extraction and classification to proceed in a unified framework, both guided by the RM principle. The framework is applied to derive new algorithms for learning feature transformations. Our experiments demonstrate the ability of the resulting algorithms to learn good features for a variety of classification problems.

1. Introduction

Most pattern classification systems consist of two main components: a feature extractor and a classifier. While the classifier is usually trained from data using machine learning techniques, the feature extractor is typically designed by hand, often based on detailed knowledge of the problem domain. Since different features are good for different classification problems, either considerable manual effort is expended in designing good features for each new problem, or there is a danger of using sub-optimal features.

In this paper, we formulate a framework that defines a general criterion for optimality of features for a given classification problem. The framework is based on extending the ideas of risk minimization, which have led to successful algorithms for learning classifiers, to learning feature transformations that admit classifiers with minimum risk. This gives a principled method for automatically learning the optimal features for a given classification problem, with respect to a given classification algorithm. Classical neural network learning methods, as well as some specialized methods for learning in specific situations, emerge as special cases of our framework. In addition, the framework allows the derivation of new feature learning algorithms.

We formulate our framework for learning minimum risk features in Section 2. Section 3 demonstrates how the framework can be applied to derive new algorithms by instantiating it for two specific cases. Specifically, we derive an algorithm for learning nonlinear features for logistic regression that generalizes the back-propagation algorithm for neural networks, and an algorithm for learning dimension-reducing features for SVMs that generalizes previous work on feature selection for SVMs. Section 4 describes applications of the two algorithms to various classification problems. Finally, we discuss relations to other work in Section 5, and conclude with a discussion in Section 6.

2. Learning Feature Transformations

We first briefly review the risk minimization (RM) principle used for learning classifiers, and then describe the extension to learning feature transformations.

2.1. Learning Classifiers via RM

In a supervised classification problem, we are given a finite set of labeled examples, $S = \{(\mathbf{x}_i, y_i)\}_{1 \leq i \leq m}$, where $\mathbf{x}_i$ are instances in some instance space X, y_i are corresponding labels in a set of class labels Y, and each example is assumed to be drawn independently from a fixed (but unknown) underlying distribution $P(\mathbf{x}, y)$. Using this training set, we are required to estimate a function $f : X \rightarrow Y$ that predicts the classification y of a new instance $\mathbf{x}$. New examples are assumed to be drawn from the same distribution $P(\mathbf{x}, y)$.

Given a loss function $\ell(f(\mathbf{x}), y)$ that measures the penalty incurred by a classifier f on example $(\mathbf{x}, y)$, the goal in RM, in accordance with Bayesian decision theory, is to find the classifier that minimizes the expected loss on a new example, or the Bayes risk:

$$R(f) = \int \ell(f(\mathbf{x}), y) dP(\mathbf{x}, y). \tag{1}$$

The true risk above cannot be computed as $P(\mathbf{x}, y)$ is unknown. Therefore, some estimate of the true risk is generally minimized instead. A common approach is to minimize the empirical risk over the training set S:

$$R_{emp}(f) = \frac{1}{m}\sum_{i=1}^{m} \ell(f(\mathbf{x}_i), y_i). \quad (2)$$

Other estimates of the true risk have also been proposed; for example, see (Chapelle et al., 2001). One can also minimize some suitable upper bound on the true risk. For simplicity, in our discussion we refer to the estimate or bound actually used in minimization as the *estimated risk*, denoted by $R_{est}(f)$.

Different classification algorithms may use different function classes for conducting the search for f. For a fixed classification algorithm $\mathcal{A}$ using a fixed function class $\mathcal{F}$, if the function f_{opt} that achieves the true minimum of $R_{est}(f)$ cannot be approximated well by any member of $\mathcal{F}$, then the classifier learned by the algorithm may have large (estimated) risk compared to f_{opt} even though it minimizes the estimated risk within $\mathcal{F}$. In such cases, to increase the expressivity of the hypothesis space effectively considered by the algorithm, it becomes necessary to first transform the input space X to a new representation $\boldsymbol{\phi}(X)$, and then learn a classifier in the new space. In other cases $\mathcal{F}$ may contain a classifier close to f_{opt}, but it may be desirable to find a transformation $\boldsymbol{\phi}$ that simplifies the input representation (e.g. to reduce dimensionality) without compromising on classification accuracy. Typically, in all such cases, feature transformations are either hand-crafted using prior domain knowledge, or selected using some measure which may or may not be related to the classification accuracy. As described below, the RM principle can be extended to give a principled method for learning a feature transformation that directly aims to minimize the (estimated) risk of the resulting classifier in the transformed space.

2.2. Learning Feature Transforms via RM

Given a classification algorithm $\mathcal{A}$ using function class $\mathcal{F}$, our goal is to find a feature transformation $\boldsymbol{\phi}$ such that the risk of the classifier learned in the transformed space $\boldsymbol{\phi}(X)$ is minimized.

To formalize this, we define the *transformation risk* of a feature transformation $\boldsymbol{\phi}$, with respect to the function class $\mathcal{F}$, to be the minimum risk that can be attained by a classifier in $\mathcal{F}$ learned in the transformed space $\boldsymbol{\phi}(X)$ (or, more generally, the infimum of the risk over classifiers in $\mathcal{F}$):[1]

$$\begin{aligned} R^T(\boldsymbol{\phi}) &= \inf_{f\in\mathcal{F}} \int \ell(f(\boldsymbol{\phi}(\mathbf{x})), y) dP(\mathbf{x}, y) \\ &= \inf_{f\in\mathcal{F}} R(f \circ \boldsymbol{\phi}). \end{aligned} \quad (3)$$

[1]Note that the functions in $\mathcal{F}$ now act on $\boldsymbol{\phi}(X)$ rather than on X; we continue to use $\mathcal{F}$ rather than $\mathcal{F}_{\boldsymbol{\phi}}$.

If the classification algorithm finds the minimum risk classifier in $\mathcal{F}$, then the optimal feature transformation is that which minimizes this transformation risk. As before, since $P(\mathbf{x}, y)$ is not known, the true transformation risk cannot be computed; some estimate must be minimized instead. Replacing the true risk in Eq. (3) with the empirical risk, we can minimize what we call the *empirical transformation risk* with respect to function class $\mathcal{F}$ and training set S:

$$\begin{aligned} R^T_{emp}(\boldsymbol{\phi}) &= \inf_{f\in\mathcal{F}} \frac{1}{m}\sum_{i=1}^{m} \ell(f(\boldsymbol{\phi}(\mathbf{x}_i)), y_i) \\ &= \inf_{f\in\mathcal{F}} R_{emp}(f \circ \boldsymbol{\phi}). \end{aligned} \quad (4)$$

Other estimates of $R^T(\boldsymbol{\phi})$ can be derived from other estimates of $R(f\circ\boldsymbol{\phi})$, and can be minimized instead; a suitable upper bound on $R^T(\boldsymbol{\phi})$ can also be minimized. Again, we refer to the estimate or bound actually used in minimization as the *estimated transformation risk*, denoted by $R^T_{est}(\boldsymbol{\phi})$.

By minimizing $R^T_{est}(\boldsymbol{\phi})$ in some class of feature transformations Φ, we thus enlarge the effective hypothesis space available to the algorithm from $\mathcal{F}$ to $\mathcal{F}\circ\Phi$, where $\mathcal{F}\circ\Phi = \{f\circ\boldsymbol{\phi} : f\in\mathcal{F}, \boldsymbol{\phi}\in\Phi\}$.[2] Both the feature transformation and the classifier can thus be learned together in the same framework as[3]

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi}\in\Phi} R^T_{est}(\boldsymbol{\phi}), \quad (5)$$

$$f^* = \arg\min_{f\in\mathcal{F}} R_{est}(f\circ\boldsymbol{\phi}^*), \quad (6)$$

the final classification function being given by $f^*\circ\boldsymbol{\phi}^*$.

In many cases, the classifier learned by the algorithm $\mathcal{A}$ may be different from the minimum risk classifier in $\mathcal{F}$. In such cases, the optimal feature transformation is that which minimizes the risk of the classifier $f^{\boldsymbol{\phi}}_{\mathcal{A}} \in \mathcal{F}$ that is actually learned by the algorithm in the transformed space $\boldsymbol{\phi}(X)$; we call this the *algorithm-specific transformation risk* with respect to $\mathcal{F}$ and S:

$$\begin{aligned} R^T_{\mathcal{A}}(\boldsymbol{\phi}) &= \int \ell(f^{\boldsymbol{\phi}}_{\mathcal{A}}(\boldsymbol{\phi}(\mathbf{x})), y) dP(\mathbf{x}, y) \\ &= R(f^{\boldsymbol{\phi}}_{\mathcal{A}} \circ \boldsymbol{\phi}). \end{aligned} \quad (7)$$

[2]It may appear that since we have simply replaced the restriction that the learned classifier lie in $\mathcal{F}$ with the restriction that it lie in $\mathcal{F}\circ\Phi$, the original problem remains unsolved. While it is true that the optimal function f_{opt} may still not lie in $\mathcal{F}\circ\Phi$, note that we are free to choose Φ, and therefore have some control over $\mathcal{F}\circ\Phi$, whereas $\mathcal{F}$ was fixed by the choice of classification algorithm. By choosing an appropriate Φ, we can therefore obtain a close approximation to f_{opt} with practically *any* classification algorithm, irrespective of the function class $\mathcal{F}$ it uses.

[3]When the infimum $\inf_{u\in U} g(u)$ is not attained in U, we use $\arg\min_{u\in U} g(u)$ to denote $u^* \in U$ such that $g(u^*) < \inf_{u\in U} g(u) + \epsilon$ for some (pre-specified) small $\epsilon > 0$.

Again, $R^T_{\mathcal{A}}(\boldsymbol{\phi})$ cannot be computed. However, if it is possible to derive an estimate or upper bound based on knowledge of the classification algorithm, then the feature transformation $\boldsymbol{\phi}^*$ can again be learned as in Eq. (5), where $R^T_{est}(\boldsymbol{\phi})$ is now an estimate or bound associated with $R^T_{\mathcal{A}}(\boldsymbol{\phi})$ rather than $R^T(\boldsymbol{\phi})$. The final classification function in this case is given by $f^{\boldsymbol{\phi}^*}_{\mathcal{A}} \circ \boldsymbol{\phi}^*$.

We discuss next how the above framework can be used to derive specific algorithms for learning features. To simplify our discussion, in the rest of the paper we consider mainly binary classification problems over real-valued data: $X = \mathbb{R}^n$, $Y = \{\pm 1\}$. We further assume that the function class $\mathcal{F}$ used by the classification algorithm is some class of real-valued functions, with the prediction made by $f \in \mathcal{F}$ on an instance $\mathbf{x}$ being given by $\text{sign}(f(\mathbf{x}))$. Feature transformations are assumed to be of the form $\boldsymbol{\phi} : \mathbb{R}^n \to \mathbb{R}^d$; in many of the cases we consider, the feature class is of the form $\Phi = \mathcal{T}^d$ for some class $\mathcal{T}$ of real-valued functions $\phi : \mathbb{R}^n \to \mathbb{R}$, i.e. for each $\mathbf{x} \in \mathbb{R}^n$, $\boldsymbol{\phi}(\mathbf{x}) = (\phi_1(\mathbf{x}), \ldots, \phi_d(\mathbf{x})) \in \mathbb{R}^d$, with $\phi_j \in \mathcal{T}\ \forall j \in \{1, \ldots, d\}$.

3. Algorithms

In this section we demonstrate how the framework formulated above can be applied to derive algorithms for learning feature transformations for different classification algorithms. We consider two types of feature learning problems: learning nonlinear features for a logistic regression classifier, and learning dimension-reducing features for an SVM classifier. The first algorithm results in a simple generalization of the back-propagation algorithm for neural networks; the second makes use of results from SVM theory to derive bounds on the SVM-specific transformation risk. Experiments with both algorithms are described in Section 4.

3.1. Nonlinear Features for Logistic Regression

Logistic regression learns a linear classifier by minimizing the empirical risk under the following loss function:

$$\ell(f(\mathbf{x}), y) = \ln(1 + \exp(-yf(\mathbf{x}))). \tag{8}$$

We consider learning a nonlinear transformation of the data that increases the expressivity of the effective hypothesis space available to the algorithm. For purposes of illustration, we choose simple features consisting of linear functions passed through a nonlinear hyperbolic tangent squashing function. The feature class is thus given by $\Phi = \mathcal{T}^d_{\tanh}$, where $\mathcal{T}_{\tanh}$ can be described as[4]

$$\{\phi : \mathbb{R}^{n+1} \to \mathbb{R} \mid \phi(\mathbf{x}) = \tanh(\textstyle\sum_{k=0}^{n} v_k x_k),\ v_k \in \mathbb{R}\}.$$

[4]We add a unit component ($x_0 = 1$) to the input representation to simplify notation for the additive bias.

We learn the optimal feature transformation (together with the optimal classifier) by minimizing the empirical transformation risk under the loss function given in Eq. (8):[5]

$$(f^*, \boldsymbol{\phi}^*) = \arg\min_{f \in \mathcal{F}, \boldsymbol{\phi} \in \Phi} R_{emp}(f \circ \boldsymbol{\phi}), \tag{9}$$

$$R_{emp}(f \circ \boldsymbol{\phi}) = \frac{1}{m} \sum_{i=1}^{m} \ln(1 + \exp(-y_i f(\boldsymbol{\phi}(\mathbf{x}_i)))). \tag{10}$$

Since $\mathcal{F}$ is the class of linear functions, we have

$$f(\boldsymbol{\phi}(\mathbf{x}_i)) = \sum_{j=1}^{d} w_j \phi_j(\mathbf{x}_i) + b, \tag{11}$$

$$\phi_j(\mathbf{x}_i) = \tanh(\sum_{k=0}^{n} v_{jk} x_{ik}), \tag{12}$$

where $w_j, b, v_{jk} \in \mathbb{R}$ are the parameters of the functions. To perform the minimization in Eq. (9) using a gradient-based method, we therefore need the following quantities, computed easily from Eqs. (10-12):

$$\frac{\partial R_{emp}(f \circ \boldsymbol{\phi})}{\partial w_j} = \frac{1}{m} \sum_{i=1}^{m} \gamma_i \phi_j(\mathbf{x}_i), \tag{13}$$

$$\frac{\partial R_{emp}(f \circ \boldsymbol{\phi})}{\partial b} = \frac{1}{m} \sum_{i=1}^{m} \gamma_i, \tag{14}$$

$$\frac{\partial R_{emp}(f \circ \boldsymbol{\phi})}{\partial v_{jk}} = \frac{1}{m} \sum_{i=1}^{m} \gamma_i w_j x_{ik} (1 - \phi_j^2(\mathbf{x}_i)), \tag{15}$$

where

$$\gamma_i = \frac{-y_i \exp(-y_i f(\boldsymbol{\phi}(\mathbf{x}_i)))}{1 + \exp(-y_i f(\boldsymbol{\phi}(\mathbf{x}_i)))}. \tag{16}$$

Using Eqs. (13-16), we can use gradient descent (or any other gradient method) to find the parameters corresponding to a local minimum in Eq. (9).

With Φ chosen as above, the effective hypothesis space in this case, $\mathcal{F} \circ \Phi$, is the class of two-layer neural networks. The above algorithm is therefore similar to the back-propagation algorithm for neural networks, with the squared loss being replaced by the logistic regression loss. However, the same framework can be instantiated with any other feature class Φ for which the above gradients can be computed.

Since the algorithm derived above performs optimization simultaneously over the parameters of the feature transformation and those of the classifier, it can also be viewed as learning a minimum risk classifier directly using the function class $\mathcal{F} \circ \Phi$. However, viewing $\mathcal{F}$

[5]Note that the optimization problem in Eq. (9) is equivalent to the optimization problem in Eqs. (5-6) (although the local minima reached by an algorithm in the two cases may be different).

and Φ separately allows Φ to be replaced with a different class of features without affecting the top-level classification algorithm. In addition, as the next section shows, the separation of $\mathcal{F}$ and Φ also allows the derivation of algorithms that do not have a direct interpretation when viewed as learning using $\mathcal{F} \circ \Phi$.

3.2. Dimension-Reducing Features for SVMs

In this section we apply the framework developed in Section 2 to derive an algorithm for learning dimension-reducing features for support vector machine (SVM) classifiers.

The function class $\mathcal{F}$ used by the SVM algorithm (Vapnik, 2000) is the class of linear functions in the $\boldsymbol{\psi}$-mapped space, $f(\mathbf{x}) = \mathbf{w} \cdot \boldsymbol{\psi}(\mathbf{x}) + b$, where $\boldsymbol{\psi}(\cdot)$ is the mapping performed by the kernel $K(\cdot, \cdot)$ being used with the SVM. (K defines an inner product in the $\boldsymbol{\psi}$-mapped space, $K(\mathbf{x}, \mathbf{x}') =< \boldsymbol{\psi}(\mathbf{x}), \boldsymbol{\psi}(\mathbf{x}') >$.) The function learned by the algorithm is parameterized as a kernel expansion on the training examples:

$$f(\mathbf{x}) = \sum_{i=1}^{m} \alpha_i y_i K(\mathbf{x}_i, \mathbf{x}) + b, \tag{17}$$

where $\alpha_i, b \in \mathbb{R}$. The algorithm finds the hyperplane function in $\mathcal{F}$ that maximizes the margin, i.e. the distance (in the $\boldsymbol{\psi}$-mapped space) of the hyperplane to the closest point in the training set.

As for many other algorithms, the performance of SVMs can degrade in the presence of a large number of irrelevant variables (Weston et al., 2001). In such a scenario, feature transformations that reduce the input dimensionality are often important. Here we consider learning features for (non-linear) SVMs that reduce the input dimensionality via a linear transformation; in other words, the feature class we consider is $\Phi = \mathcal{T}_{lin}^{d}$, where $d < n$ and

$$\mathcal{T}_{lin} = \{\phi : \mathbb{R}^n \to \mathbb{R} \mid \phi(\mathbf{x}) = \textstyle\sum_{k=1}^{n} v_k x_k,\ v_k \in \mathbb{R}\}.$$

Based on the theory of SVMs, one can define upper bounds on the expected error probability of the function $f_{\text{SVM}}^{\boldsymbol{\phi}}$ learned by the algorithm in the transformed space $\boldsymbol{\phi}(X)$. These can be used to derive an approximate upper bound on the SVM-specific transformation risk, $R_{\text{SVM}}^{T}(\boldsymbol{\phi})$ (see Eq. (7) for the definition of algorithm-specific transformation risk). The resulting upper bound can then be minimized to learn a transformation $\boldsymbol{\phi}^*$ that approximately minimizes $R_{\text{SVM}}^{T}(\boldsymbol{\phi})$. As we discuss in Section 5.2, Weston et al. (2001) use such a procedure for the special case of feature selection; we extend the procedure here to learning more general classes of transformations.

Given a transformation $\boldsymbol{\phi}$, we define the kernel function $K_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{x}') = K(\boldsymbol{\phi}(\mathbf{x}), \boldsymbol{\phi}(\mathbf{x}'))$; thus $K_{\boldsymbol{\phi}}$ defines an inner product in the $(\boldsymbol{\psi} \circ \boldsymbol{\phi})$-mapped space, $K_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{x}') =< \boldsymbol{\psi}(\boldsymbol{\phi}(\mathbf{x})), \boldsymbol{\psi}(\boldsymbol{\phi}(\mathbf{x}')) >$. The SVM algorithm finds the hyperplane with maximal margin in the $(\boldsymbol{\psi} \circ \boldsymbol{\phi})$-mapped space; the parameters $\boldsymbol{\alpha}^{\boldsymbol{\phi}}$ of the function $f_{\text{SVM}}^{\boldsymbol{\phi}}$ learned in the transformed space (which has the form given in Eq. (17) with kernel $K_{\boldsymbol{\phi}}$) are those that maximize the functional

$$W_{\boldsymbol{\phi}}(\boldsymbol{\alpha}) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,i'=1}^{m} \alpha_i \alpha_{i'} y_i y_{i'} K_{\boldsymbol{\phi}}(\mathbf{x}_i, \mathbf{x}_{i'}), \tag{18}$$

subject to the constraints $\sum_{i=1}^{m} y_i \alpha_i = 0, \alpha_i \geq 0\ \forall i$.

The error probability of the classifier $f_{\text{SVM}}^{\boldsymbol{\phi}}$, i.e. the SVM-specific transformation risk of $\boldsymbol{\phi}$ under the 0-1 loss function, is defined as follows:

$$R_{\text{SVM}}^{T}(\boldsymbol{\phi}) = \frac{1}{2} \int (1 - \text{sign}(y f_{\text{SVM}}^{\boldsymbol{\phi}}(\boldsymbol{\phi}(\mathbf{x})))) dP(\mathbf{x}, y).$$

If the images $(\boldsymbol{\psi} \circ \boldsymbol{\phi})(\mathbf{x}_i)$ of the training data lie within a sphere of radius $\rho_{\boldsymbol{\phi}}$ and are separable with margin $M_{\boldsymbol{\phi}}$, then the following result holds (Vapnik, 2000):

$$E[R_{\text{SVM}}^{T}(\boldsymbol{\phi})] \leq \frac{1}{m} E\left[\frac{\rho_{\boldsymbol{\phi}}^2}{M_{\boldsymbol{\phi}}^2}\right], \tag{19}$$

where the expectation is taken over all training sets of size m. The maximal margin $M_{\boldsymbol{\phi}}$ is given by

$$\frac{1}{M_{\boldsymbol{\phi}}^2} = 2W_{\boldsymbol{\phi}}(\boldsymbol{\alpha}^{\boldsymbol{\phi}}), \tag{20}$$

and the radius $\rho_{\boldsymbol{\phi}}$ can be computed (Vapnik, 1998) as

$$\rho_{\boldsymbol{\phi}}^2 = U_{\boldsymbol{\phi}}(\boldsymbol{\beta}^{\boldsymbol{\phi}}), \tag{21}$$

where

$$U_{\boldsymbol{\phi}}(\boldsymbol{\beta}) = \sum_{i=1}^{m} \beta_i K_{\boldsymbol{\phi}}(\mathbf{x}_i, \mathbf{x}_i) - \sum_{i,i'=1}^{m} \beta_i \beta_{i'} K_{\boldsymbol{\phi}}(\mathbf{x}_i, \mathbf{x}_{i'}),$$

and $\boldsymbol{\beta}^{\boldsymbol{\phi}}$ achieves the maximum of $U_{\boldsymbol{\phi}}(\boldsymbol{\beta})$ subject to the constraints $\sum_{i=1}^{m} \beta_i = 1, \beta_i \geq 0\ \forall i$. Substituting Eqs. (20-21) in Eq. (19) gives

$$E[R_{\text{SVM}}^{T}(\boldsymbol{\phi})] \leq \frac{2}{m} E\left[U_{\boldsymbol{\phi}}(\boldsymbol{\beta}^{\boldsymbol{\phi}}) W_{\boldsymbol{\phi}}(\boldsymbol{\alpha}^{\boldsymbol{\phi}})\right]. \tag{22}$$

Approximating the expectations above with their point estimates obtained from the given training set S, one can treat the following quantity as an approximate upper bound on $R_{\text{SVM}}^{T}(\boldsymbol{\phi})$:

$$R_{est}^{T}(\boldsymbol{\phi}) = \frac{2}{m} U_{\boldsymbol{\phi}}(\boldsymbol{\beta}^{\boldsymbol{\phi}}) W_{\boldsymbol{\phi}}(\boldsymbol{\alpha}^{\boldsymbol{\phi}}). \tag{23}$$

This can be minimized over $\boldsymbol{\phi}$ to learn features that approximately minimize the (SVM-specific) transformation risk:

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi} \in \Phi} R_{est}^{T}(\boldsymbol{\phi}). \tag{24}$$

Recall that $\boldsymbol{\phi}$ in our case has the form

$$\phi_j(\mathbf{x}) = \sum_{k=1}^{n} v_{jk} x_k, \quad 1 \le j \le d. \tag{25}$$

To perform the minimization in Eq. (24) using a gradient method, we therefore need to compute the gradient of $R^T_{est}(\boldsymbol{\phi})$ with respect to v_{jk}. This is given by

$$\frac{\partial R^T_{est}(\boldsymbol{\phi})}{\partial v_{jk}} = \frac{2}{m} U_{\boldsymbol{\phi}}(\boldsymbol{\beta}^{\boldsymbol{\phi}}) \frac{\partial W_{\boldsymbol{\phi}}(\boldsymbol{\alpha}^{\boldsymbol{\phi}})}{\partial v_{jk}} + \frac{2}{m} W_{\boldsymbol{\phi}}(\boldsymbol{\alpha}^{\boldsymbol{\phi}}) \frac{\partial U_{\boldsymbol{\phi}}(\boldsymbol{\beta}^{\boldsymbol{\phi}})}{\partial v_{jk}},$$

where (neglecting the terms involving gradients of $\boldsymbol{\alpha}^{\boldsymbol{\phi}}$ and $\boldsymbol{\beta}^{\boldsymbol{\phi}}$ with respect to v_{jk})

$$\frac{\partial W_{\boldsymbol{\phi}}(\boldsymbol{\alpha}^{\boldsymbol{\phi}})}{\partial v_{jk}} = -\frac{1}{2} \sum_{i,i'=1}^{m} \alpha_i^{\boldsymbol{\phi}} \alpha_{i'}^{\boldsymbol{\phi}} y_i y_{i'} \frac{\partial K_{\boldsymbol{\phi}}(\mathbf{x}_i, \mathbf{x}_{i'})}{\partial v_{jk}},$$

$$\frac{\partial U_{\boldsymbol{\phi}}(\boldsymbol{\beta}^{\boldsymbol{\phi}})}{\partial v_{jk}} = \sum_{i=1}^{m} \beta_i^{\boldsymbol{\phi}} \frac{\partial K_{\boldsymbol{\phi}}(\mathbf{x}_i, \mathbf{x}_i)}{\partial v_{jk}} - \sum_{i,i'=1}^{m} \beta_i^{\boldsymbol{\phi}} \beta_{i'}^{\boldsymbol{\phi}} \frac{\partial K_{\boldsymbol{\phi}}(\mathbf{x}_i, \mathbf{x}_{i'})}{\partial v_{jk}}.$$

It is easy to calculate the gradient of the kernel function in the above equations; for an SVM using a polynomial kernel of order n, this is

$$\frac{\partial K_{\boldsymbol{\phi}}(\mathbf{x}, \mathbf{x}')}{\partial v_{jk}} = n(1 + \boldsymbol{\phi}(\mathbf{x}) \cdot \boldsymbol{\phi}(\mathbf{x}'))^{n-1} (x_k \phi_j(\mathbf{x}') + x'_k \phi_j(\mathbf{x})).$$

Once $\boldsymbol{\phi}^*$ is found, the final classification function is given by $f^{\boldsymbol{\phi}^*}_{\mathrm{SVM}} \circ \boldsymbol{\phi}^*$.

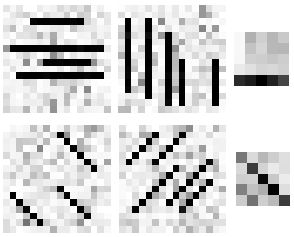
The above algorithm can also be viewed as defining a family of kernel functions $\{K_{\boldsymbol{\phi}}\}_{\boldsymbol{\phi} \in \Phi}$ parameterized by $\boldsymbol{\phi}$, and then learning the optimal kernel parameters. However, the criterion that drives the parameter learning (namely, minimizing the transformation risk) is different from the criterion that drives the classifier learning (maximizing the margin). Consequently, the algorithm derived above does not have a direct interpretation when viewed as an algorithm for learning directly using the function class $\mathcal{F} \circ \Phi$.

4. Applications

In this section we describe some applications of the feature learning algorithms derived above. Section 4.1 presents experiments with the algorithm for learning nonlinear features for logistic regression, while Section 4.2 presents experiments with the algorithm for learning dimension-reducing features for SVMs.

4.1. Logistic Regression Experiments

The algorithm of Section 3.1 can be useful for a wide range of applications. Here we show its application to two important problems: learning convolution kernels for image data, and learning features for binary data.



Representation	Test error
Raw input	20.50 %
Learned features	0.0 %

Representation	Test error
Raw input	8.50 %
Learned features	0.10 %

Figure 1. Examples of positive and negative images used in the two image classification problems, the convolution kernel learned in each case, and the performance of logistic regression in the original and transformed spaces (see text). **Top row:** Horizontal vs. vertical lines problem. **Bottom row:** Diagonal lines (NW-SE vs. NE-SW) problem.

4.1.1. Convolution Kernels for Image Data

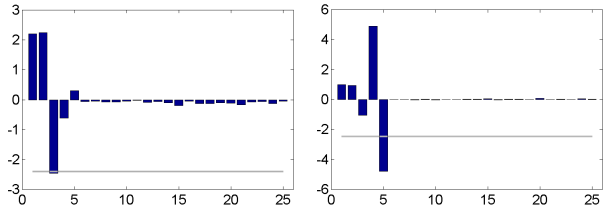
Convolution with small image kernels has proved to be an effective feature generation method for classification problems involving image data (Teow & Loe, 2000). However, in most cases, these kernels are selected manually through trial and error. We apply the algorithm of Section 3.1 to automatically learn such convolution kernels. Convolutional networks (LeCun et al., 1998) use a similar principle (RM under squared loss) to learn such kernels in their intermediate layers.

Given an image $\mathbf{x} \in \mathbb{R}^{n \times n}$, a convolution kernel $\boldsymbol{\theta} \in \mathbb{R}^{r \times r}$ (where $r < n$) defines a feature map $\mathbf{z} \in \mathbb{R}^{p \times p}$ ($p = n - r + 1$), where each element of the feature map is a dot product between the kernel and an $r \times r$ region of the image. Adding a bias term to the dot product and passing the result through a tanh squashing function, we get the following nonlinear features with $\boldsymbol{\theta}$:

$$z_{kl} = \tanh\left(\sum_{u=1}^{r} \sum_{v=1}^{r} \theta_{uv} x_{k+u, l+v} + \theta_0 \right), 1 \le k, l \le p$$

Using d such kernels $\boldsymbol{\theta}_j$ (together with bias terms θ_{j0}) thus defines dp^2 functions, ϕ_{jkl} ($1 \le j \le d, 1 \le k, l \le p$), in the class $\mathcal{T}_{\tanh}$ defined in Section 3.1. These can be learned using a simple modification of the algorithm of Section 3.1; since the p^2 functions corresponding to each kernel $\boldsymbol{\theta}_j$ share parameters, a simple modification is required in the computation of the gradient with respect to the feature parameters.

As a simple illustration, we consider the problem of classifying images containing lines of different orientations. We generated two synthetic data sets for this purpose; the first data set contains images with horizontal lines as positive examples and images with vertical lines as negative examples, while the second data set has diagonal lines in NW-SE orientation as positive examples and NE-SW orientation as negative examples. Examples of the images are shown in Figure 1; the images are 16×16 pixels in size, have small random



Representation	Test error
Raw input	5.7 %
Learned features	0.0 %

Figure 2. **Top:** Coefficients in the two features learned for the target function $(x_1 \wedge x_2 \wedge \neg x_3) \vee (x_4 \wedge \neg x_5)$. The grey lines represent the bias terms. **Bottom:** Performance of logistic regression in the original and transformed spaces.

noise in the background, and contain 3-5 lines of varying lengths and in varying positions. Each data set has 2000 training images and 1000 test images. Since the data sets are not linearly separable in the input space, a logistic regression classifier over the raw image data gives poor performance. However, on using the algorithm of Section 3.1 to learn a single 5×5 convolution kernel, logistic regression in the transformed space gives near-perfect classification in both cases (see Figure 1). The convolution kernels learned in each case are also shown in Figure 1; in both cases, the kernels were initialized with small random entries, and gradient descent was used to minimize the empirical risk. As can be seen, the algorithm successfully adapts the kernels to the specific classification problem.

4.1.2. Features for Binary Data

In a large number of domains, the data is represented by binary variables ($X = \{0,1\}^n$), and the target function is a Boolean function over the variables. Since the target function can be represented as a DNF (disjunctive normal form) expression, the ability to learn conjunctive features is important for such domains.

For any conjunction $c(x_1, \ldots, x_n)$, there exist $v_j \in \mathbb{R}$ such that $c(x_1, \ldots, x_n)$ is true if and only if $\text{sign}(\sum_{j=1}^{n} v_j x_j + v_0) = 1$. Using the continuous tanh function in place of the discrete sign function, it is therefore possible to approximate a set of d conjunctions over the variables with a feature transformation in $\mathcal{T}^d_{\tanh}$. The algorithm of Section 3.1 can be used to learn such a transformation.

As a simple experiment, we generated a synthetic data set in $\{0,1\}^{25}$ by independently choosing 0 or 1 from a uniform distribution for each of the 25 variables, and labeling the resulting instances according to the function $(x_1 \wedge x_2 \wedge \neg x_3) \vee (x_4 \wedge \neg x_5)$. We used 2000 examples for training and 1000 for testing. Figure 2 shows the results obtained with a typical run of the algorithm when $d = 2$; the learned features approximate well the conjunctions in the target function. Running the algorithm with $d = 5$ again gave perfect classification; interestingly, 2 of the 5 features learned in this case corresponded to the 2 conjunctions in the target function, while the remaining 3 features (which were more or less random) were eliminated at the classification stage by being assigned small weights in the classifier.

We should point out that the above method does not attempt to learn a DNF, but rather is presented as an example to show the potential of the framework in learning features that improve classification accuracy.

4.2. SVM Experiments

We applied the algorithm derived in Section 3.2 to learn dimension-reducing features for classification problems from two different domains: clsasification of diabetes patients, and classification of radar returns from the ionosphere. The data sets are both from the UCI Machine Learning Repository. In both cases we used an SVM with a second order polynomial kernel. The results are shown in Tables 1-2 (MTR refers to minimum transformation risk); we present the results obtained with principal component analysis (PCA) for comparison. For each dimension we initialized the gradient descent procedure with three different random transformations, and chose the transformation with minimum (estimated) transformation risk of the three. The minimization was done subject to the transformations having unit norm ($\|\mathbf{v}_j\| = 1$, $1 \leq j \leq d$).

Since the bound on the transformation risk used is only approximate (see Section 3.2), the error does not always exhibit a monotonic relation to it. More sophisticated bounds have been used for learning kernel parameters for SVMs (Chapelle et al., 2002); it may be possible to obtain better estimates using such bounds.

Table 1. Test error on the Pima Diabetes data set using a second order SVM. (Training size 500, test size 268.)

Reduction Method	Reduced space		Input space
	$d = 2$	$d = 4$	$n = 8$
PCA	23.88 %	22.76 %	19.40 %
MTR	24.25 %	20.52 %	

Table 2. Test error on the Ionosphere data set using a second order SVM. (Training size 240, test size 111.)

Reduction Method	Reduced space		Input space
	$d = 4$	$d = 12$	$n = 33$
PCA	13.51 %	8.11 %	7.21 %
MTR	9.91 %	9.91 %	

5. Related Work

In this section we discuss the relationship of our framework to existing methods. The framework clearly contains neural network learning methods, which can be seen as learning features in hidden layers, as a special case (Section 5.1). A method used for feature selection for SVMs can also be viewed as a special case of the framework (Section 5.2). Section 5.3 discusses relations with the model used in inductive bias learning. We discuss here only work related to the RM framework; for work on constructive induction methods, see, for example, (Markovitch & Rosenstein, 2002).

5.1. Neural Networks

Consider the back-propagation algorithm for learning a multilayer perceptron (MLP) with k layers, where the $(k-1)$ hidden layers use some nonlinear squashing function while the output unit[6] simply returns a weighted sum of the activations at the $(k-1)$th layer. If we let $\mathcal{F}$ be the class of linear functions $f(\mathbf{x}) = \mathbf{w} \cdot \mathbf{x} + b$, and take $\ell(f(\mathbf{x}), y)$ to be the squared loss $(f(\mathbf{x}) - y)^2$, then the back-propagation algorithm can be viewed as attempting to find $f^* \circ \boldsymbol{\phi}^*$ such that

$$(f^*, \boldsymbol{\phi}^*) = \arg\min_{f \in \mathcal{F}, \boldsymbol{\phi} \in \Phi} R_{emp}(f \circ \boldsymbol{\phi}), \qquad (26)$$

where $\Phi = \mathcal{T}^d_{\mathrm{MLP}}$, with d being the number of hidden units in the $(k-1)$th layer, and $\mathcal{T}_{\mathrm{MLP}}$ being the class of squashed linear functions if $k = 2$, and the class of $(k-1)$ layer MLPs (with squashed outputs) if $k > 2$.

Radial basis function (RBF) networks can also be viewed in this framework. An RBF network in which the hidden RBF units and linear output unit are trained together can be seen as implementing the minimization in Eq. (26), with the function class $\mathcal{F}$ and loss function ℓ being the same as for MLPs, and with feature class $\Phi = \mathcal{T}^d_{\mathrm{RBF}}$, where d is the number of RBF units in the hidden layer and $\mathcal{T}_{\mathrm{RBF}}$ is the class of radial basis functions chosen for the network.

5.2. Feature Selection for SVMs

A method for selecting features for SVMs that also falls into our framework is proposed in (Weston et al., 2001). Feature selection can be viewed as a restricted type of feature transformation, where the new features are simply a subset of the original input features; the feature class here is thus $\Phi = \mathcal{T}^d_{sel}$, where $d < n$ and

$$\mathcal{T}_{sel} = \{\phi : \mathbb{R}^n \to \mathbb{R} \mid \phi(\mathbf{x}) = x_k, 1 \leq k \leq n\}.$$

As discussed in Section 3.2, it is possible to define upper bounds on the expected error probability of the function $f^{\boldsymbol{\phi}}_{\mathrm{SVM}}$ learned by an SVM in the transformed space $\boldsymbol{\phi}(\mathbb{R}^n) = \mathbb{R}^d$; these can be used to derive approximate upper bounds on the SVM-specific transformation risk. Weston et al. (2001) minimize such bounds to find the optimal feature subset for a given problem.

5.3. Inductive Bias Learning

The inductive bias learning model developed in (Baxter, 2000) addresses the problem of learning bias in a scenario where the learner is embedded in some environment of related learning tasks, each task having its own distribution $P(\mathbf{x}, y)$ over the data, and a distribution $Q(P)$ governing the probability of seeing each task. The bias is represented by the choice of hypothesis space. The learner is given a family of hypothesis spaces together with data samples from some number of tasks in the environment, and the goal is to choose a hypothesis space that contains good solutions both to the tasks already seen and to novel tasks from the environment that could be seen in the future.

Baxter applies the bias learning model to the problem of learning a feature transformation that is appropriate for such an environment of learning tasks. This is done by representing the hypothesis space family as

$$\mathbb{H} = \mathcal{F} \circ \Phi = \{\mathcal{F} \circ \boldsymbol{\phi} : \boldsymbol{\phi} \in \Phi\},$$

where $\mathcal{F}$ is the (fixed) function class to be used by the classification algorithm and Φ is a class of feature transformations, so that choosing a hypothesis space from $\mathbb{H}$ corresponds to choosing a feature transformation from Φ. (In the example presented in (Baxter, 2000), $\mathcal{F}$ is the class of squashed linear functions, while Φ is the class of two-layer neural networks.) Using our terminology, the method proposed there corresponds to choosing the feature transformation that minimizes the *expected* transformation risk on a new task, where the expectation is according to the distribution $Q(P)$ over the tasks in the environment:[7]

$$E_Q[R^T(\boldsymbol{\phi}; P)] = \int R^T(\boldsymbol{\phi}; P) dQ(P).$$

Since the distributions are not known, the actual minimization is done using an estimate of the above quantity; given data samples from N tasks, the transformation that minimizes the average (estimated) transformation risk over the seen tasks is selected:[8]

$$\boldsymbol{\phi}^* = \arg\min_{\boldsymbol{\phi} \in \Phi} \frac{1}{N} \sum_{i=1}^{N} R^T_{est}(\boldsymbol{\phi}; S_i).$$

[6] The output layer here consists of a single unit since we are considering a binary classification problem; however, the discussion holds also for the multiple-output case.

[7] We use $R^T(\boldsymbol{\phi}; P)$ to denote the transformation risk of $\boldsymbol{\phi}$ under the distribution $P(\mathbf{x}, y)$. In Baxter's example the risk is computed using the squared loss $(f(\mathbf{x}) - y)^2$.

[8] We use $R^T_{est}(\boldsymbol{\phi}; S_i)$ to denote the estimated transformation risk of $\boldsymbol{\phi}$ computed using the data sample S_i corresponding to the i-th task.

6. Discussion

The question of how to learn good features for classification is an important one in machine learning. While there have been some attempts to answer this question in special cases, a general theory for learning optimal features has been missing. We have taken some steps in this direction by defining a criterion for optimality of features based on the principle of risk minimization. Our framework generalizes a number of earlier studies, including classical neural network learning methods, in addition to allowing the derivation of new feature learning algorithms. We have applied the framework to derive algorithms for two specific cases: learning nonlinear features for logistic regression, and learning dimension-reducing features for SVMs. Preliminary experiments demonstrate the potential of the method for learning good features for a variety of problems.

In the following sections, we discuss some issues that have not found place elsewhere in the paper, and then conclude with a discussion of some implications and directions for future work.

6.1. Choice of Feature Class

One issue that has not been addressed is the choice of feature class Φ. In some cases Φ may be determined by the problem itself (such as when a linear dimension-reducing transformation is sought). In other cases, when the goal is to increase the expressivity of the effective hypothesis space, the choice of Φ may be more difficult. Simple generic features, such as squashed linear functions, are often sufficient to improve upon the input representation; such feature classes can be chosen when no other knowledge of the domain is available. In more familiar domains, prior knowledge can be used to make a more intelligent choice of Φ, such as the class of convolution kernels we chose in Section 4.1.1 that capture spatial features for image data.

6.2. Capacity of Effective Hypothesis Space

When a complex feature class is used to increase the expressivity of the effective hypothesis space, the increase in capacity of the resulting space is a potential concern. In this case, an assumption about the availability of data is clearly needed for good generalization. On the other hand, when a simple feature class is used to reduce dimensionality, the capacity of the resulting space may actually be reduced, potentially allowing for better generalization from limited data.

6.3. Implications and Future Work

Algorithms for automatic feature learning clearly have important implications for applying learning techniques to new domains, in which prior knowledge is limited and therefore good features are difficult to design manually. In addition, automatic learning of features is desirable even for domains in which prior knowledge is available. For example, consider the problem of handwritten digit recognition; as shown in (Teow & Loe, 2000), features generated with convolution kernels give excellent performance for this task. However, the convolution kernels there are defined by hand; for a new problem from the same domain (e.g. character or digit recognition in a different language), considerable manual effort would be needed to design new kernels. On the other hand, an algorithm for learning features would be able to automatically learn the optimal kernels for a new problem.

The character recognition example also gives an avenue for further work, namely learning features for multi-class classification problems. It should be possible to achieve this with our framework using a multi-class loss function. So far we have considered only continuous feature transformations which are amenable to optimization using gradient methods. However, for many domains, the features of interest come from discrete transformation classes; therefore another interesting direction for future work is to develop methods for finding minimal risk features from a discrete class.

References

Baxter, J. (2000). A model of inductive bias learning. *Journal of Artificial Intelligence Research*, *12*, 149–198.

Chapelle, O., Vapnik, V., Bousquet, O., & Mukherjee, S. (2002). Choosing multiple parameters for support vector machines. *Machine Learning*, *46*, 131–159.

Chapelle, O., Weston, J., Bottou, L., & Vapnik, V. N. (2001). Vicinal risk minimization. *Advances in Neural Information Processing Systems 13*. MIT Press.

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

Markovitch, S., & Rosenstein, D. (2002). Feature generation using general constructor functions. *Machine Learning*, *49*, 59–98.

Teow, L.-N., & Loe, K.-F. (2000). Handwritten digit recognition with a novel vision model that extracts linearly separable features. *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*.

Vapnik, V. N. (1998). *Statistical learning theory*. New York: John Wiley and Sons.

Vapnik, V. N. (2000). *The nature of statistical learning theory*. New York: Springer. Second edition.

Weston, J., Mukherjee, S., Chapelle, O., Pontil, M., Poggio, T., & Vapnik, V. N. (2001). Feature selection for SVMs. *Advances in Neural Information Processing Systems 13*. MIT Press.