9 Linear Discriminant and Support Vector Classifiers

Isabelle Guyon
Clopinet
955 Creston Road
Berkeley, CA 94708-1501
isabelle@clopinet.com
http://www.clopinet.com

David G. Stork
Ricoh Silicon Valley
2882 Sand Hill Road, Suite 115
Menlo Park, CA 94025-7022
stork@rsv.ricoh.com
http://rsv.ricoh.com/˜ stork

Support Vector Machines were introduced and first applied to classification problems as alternatives to multi-layer neural networks. The high generalization ability provided by Support Vector Classifiers (SVCs) has inspired recent work on computational speedups as well as the fundamental theory of model complexity and generalization. At first glance, a Support Vector Classifier appears to be nothing more than a generalized linear discriminant in a high dimensional transformed feature space; indeed, many aspects of SVCs can best be understood in relation to traditional linear discriminant techniques.

This chapter explores interconnections between many linear discriminant techniques, including Perceptron, Radial Basis Functions (RBFs) and SVCs. The principle of duality between learning- or feature-based techniques (such as Perceptrons) and memory- or example-based methods (such as RBFs) is central to the development of SVCs. We provide several other examples of duality in linear discriminant learning algorithms.

9.1 Introduction

Support Vector Classifiers are linear classifiers in a high dimensional space. Recently, a large number of SVC training algorithms have been proposed [Krauth and Mézard, 1987, Frieß et al., 1998, Freund and Schapire, 1998, Schölkopf et al., 1999a], but there seems to be a lack of appreciation of the close relationships among SVCs and many other classification techniques, in particular older "classical" methods. Our goal is to clarify key relationships among several important methods. While we cannot address all relevant techniques, we shall discuss methods that are of historical importance, practical use, and illustrative of key concepts. In doing so, we shall emphasize the complementary nature of distributed *learning based methods* in a direct weight space, and local, *memory based methods* in a dual parameter space. We shall not consider questions of regularization, generalization, structural risk minimization, or convergence rates.

There are three interrelated reasons for the long and extensive literature on linear classifiers. First, such classifiers are computationally quite simple, both during learning and during classification. Second, despite this simplicity, linear classifiers have had great success in a large number of application areas. Finally, they are amenable to theoretical analysis. For instance, the proof of the Perceptron convergence theorem greatly encouraged practitioners [Rosenblatt, 1958]. These reasons continue to underlay the interest in current linear methods such as SVCs.

Our chapter is organized as follows. In Section 9.2 we review the basic concepts of classifiers based on linear discriminants and highlight the key role of duality between learning methods in the direct space and those in a dual space. We turn, in Section 9.3, to several settings or formulations of the learning problem for classification, including those based on criteria of training error, Perceptron objective function and mean squared error objective function. Then in Section 9.4 we describe several training algorithms for linear classifiers in general and SVCs in particular. Then in Section 9.5 we consider the properties of the solutions, such as robustness, sensitivity to outliers, and so on. We conclude in Section 9.6 with a summary of our main points and a set of outstanding problems.

9.2 What is a Linear Discriminant?

9.2.1 Classification Problem

We consider the two class classification problem, in which patterns, represented as n-dimensional vectors $\mathbf{x}$, are labeled with binary values $y \in \{-1, +1\}$. For example, assume that images of handwritten digits `0` and `1` must be classified. An optical scanner yields a gray level pixel image; each pixel can be encoded as a number corresponding to one component of $\mathbf{x}$. Images of the digit `0` are labeled -1 and images of the digit `1` are labeled $+1$.

Classification problems are usually characterized by an unknown probability distribution $p(\mathbf{x}, y)$ on $\mathbb{R}^n \times \{-1, +1\}$, but training examples $\mathbf{x}_i$ and their corresponding class labels y_i are provided:

$$X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subseteq \mathbb{R}^n \tag{9.1}$$

$$Y = \{y_1, \ldots, y_m\} \subseteq \{-1, 1\}. \tag{9.2}$$

The problem is to find a decision function $g(\mathbf{x})$, which predicts accurately the class label y of any example $\mathbf{x}$ that may or may not belong to the training set.

9.2.2 Discriminant Function

discriminant function

The discriminant function approach uses a real valued function $f(\mathbf{x})$, called *discriminant function*, the sign of which determines the class label prediction: $g(\mathbf{x}) = \mathrm{sgn}\,(f(\mathbf{x}))$. The discriminant function $f(\mathbf{x})$ may be parametrized with some parameters $\mathbf{a} = \{a_1, \ldots, a_p\}$ that are determined from the training examples by means of a learning algorithm. To be explicit, we should write $f(\mathbf{x}; \mathbf{a})$, but we shall generally omit this in the following. A great many pattern classification methods employ discriminant functions, either implicitly or explicitly. For instance, when an input pattern $\mathbf{x}$ is presented to the input layer in a standard three layer neural network having a 1-of-c representation at the output layer, each output unit computes, effectively, a discriminant function. The pattern is assigned the category label corresponding to the output unit with the largest discriminant value.

9.2.3 Linear Discriminant

Linear discriminant functions are discriminant functions that are linear in their parameters. Two kinds of linear discriminant functions have been studied in depth:

$$f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x}) \qquad \text{linear classifier or “Perceptron,” and} \tag{9.3}$$

$$f(\mathbf{x}) = \sum_{i=1}^{m} y_i \alpha_i k(\mathbf{x}, \mathbf{x}_i) \qquad \text{kernel classifier,} \tag{9.4}$$

with parameters $\mathbf{w} \in \mathbb{R}^N$, $\alpha_i \in \mathbb{R}$, basis functions $\Phi : \mathbb{R}^n \to \mathbb{R}^N$ and symmetric kernel functions $k : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$.[1] The basis and kernel functions may also be parametrized, but the parameters are not subject to training with the examples in the framework that we consider here.

For certain sets of basis functions, linear discriminant functions are universal approximators, that is, for well chosen parameters, the decision boundary $f(\mathbf{x}) = 0$

1. We simplify the parametrization of linear classifiers by integrating an eventual bias value b as an additional component of the weight vector. This corresponds to assuming that one of the components of $\Phi(\mathbf{x})$ is a constant.

can arbitrarily well approximate any decision boundary, including the ideal decision boundary. Such is the case, for instance, of polynomial classifiers. The basis functions $\Phi_k(\mathbf{x})$ of a polynomial classifier are monomials that are products of a number of components of $\mathbf{x}$.

9.2.4 Feature-based Classifiers

Perceptron

The basis functions $\Phi_k(\mathbf{x})$ of linear classifiers or "Perceptrons" of Equation 9.4 can be understood as feature detectors. For example, in a handwritten digit recognition task where 0s and 1s have to be classified, algorithms can be derived that detect the presence or absence of a straight line or of a loop. In the absence of domain knowledge about the task necessary to design such *ad hoc* feature detectors, general purpose basis functions can be used, such as monomials in a polynomial classifier.

9.2.5 Example-based Classifiers

RBF

The symmetric kernel functions $k(\mathbf{x}, \mathbf{x}')$ of kernel classifiers of Equation 9.4 are often (although not necessarily) radial basis functions. For example $k(\mathbf{x}, \mathbf{x}') = \exp(-\gamma\|\mathbf{x} - \mathbf{x}'\|^2)$ is a bell shaped or "Gaussian" kernel. Each contribution to the decision function $k(\mathbf{x}, \mathbf{x}_i)$ is centered around one example $\mathbf{x}_i$, where it is maximum, and vanishes as the Euclidean distance between $\mathbf{x}$ and $\mathbf{x}_i$ increases. Several examples of general purpose kernel functions are given in Table 9.1. Note that the polynomial kernel is not, in general, a Radial Basis Function. The computation of the linear discriminant of kernel classifiers requires storing in memory the training examples, thus the name "example-based" or "memory-based" classifier.

Parzen windows

There are close relationships between kernel classifiers and other non-parametric classification methods such as Parzen windows [Parzen, 1962b] and nearest neighbors [Cover and Hart, 1967]. Parzen windows are essentially kernel classifiers normalized such that $f(\mathbf{x})$ can be used as an estimator of the posterior probability $P(y = 1|\mathbf{x})$. When rectangular windows are used, the classification decision is done according to the largest number of examples of each class falling within a window of a given size that is centered around $\mathbf{x}$.

nearest neighbors

Similarly, in a k nearest neighbor classifier, the decision is made according to the majority label among the k training examples that are nearest to the test point $\mathbf{x}$. One of the benefits of k nearest neighbor classifiers is that decision boundaries are smooth where the data is sparse, but sharp and curvy where the data is dense, as we might wish.

tangent distance

Kernels can also be thought of as similarity measures and thus can often be designed to incorporate domain knowledge about the task. For example, in a handwriting digit recognition task, a similarity measure that incorporates invariance with respect to rotations, translations and other distortions is better suited than a simple dot product (see for instance the "tangent distance" [Simard et al., 1993, Schölkopf et al., 1998a]).

9.2.6 Feature and Example Selection

In general, the more complex the classification problem (i.e., the more complex the optimal decision boundary), the larger the number of features that should be included in the classifier, for example all the monomials of a polynomial classifier of order d. However the number of parameters to estimate rapidly becomes prohibitive, e.g., $O(n^d)$. Similarly, training a large number of parameters may be infeasible, either because of high computation complexity or, more seriously, if there is insufficient training data, because the solution is underdetermined.

principal component analysis

The classical approach to these general problems is to select a subset of features, itself often a subtle and challenging task [Schürmann, 1996]. One of the most popular methods is principal component analysis, in which the n-dimensional features are projected to a subspace so as to minimize an MSE measure [Oja, 1983]. Likewise, the computational load imposed by kernel classifiers can be reduced by selecting a subset of the training examples or by creating prototypes out of linear combinations of examples.

9.2.7 Locality

Feature-based linear classifiers are considered "global" decision functions because they are globally defined by *all* the training examples. Conversely, kernel classifiers, which rely on local radial kernels, each of which determined by a local subset of the training examples, are often considered "local" classifiers. When radial kernels are used, the sign of the decision function $f(\mathbf{x})$ is influenced by many local soft decisions of presence or absence of $\mathbf{x}$ in the neighborhood of $\mathbf{x}_i$.

9.2.8 Duality

Under certain conditions linear classifiers, Equation 9.4, and kernel classifiers, Equation 9.4, are two different representations of the same discriminant function. This invalidates the usual distinction between "global" and "local" linear discriminant.

For example, a wide variety of training algorithms of linear classifiers, including gradient descent algorithms, yield a weight vector $\mathbf{w}$, which is a weighted sum of the training examples in Φ-space:

$$\mathbf{w} = \sum_{i=1}^{m} y_i \alpha_i \Phi(\mathbf{x}_i). \tag{9.5}$$

By substituting $\mathbf{w}$ in the expression of the linear classifier (Equation 9.4): $f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x})$ one obtains a kernel classifier:

$$f(\mathbf{x}) = \sum_{i=1}^{m} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x}) \tag{9.6}$$

with kernel $k(\mathbf{x}_i, \mathbf{x}) = \Phi(\mathbf{x}_i) \cdot \Phi(\mathbf{x})$.

$k(\mathbf{x}, \mathbf{x}')$	**Name**
$(\mathbf{x} \cdot \mathbf{x}' + 1)^d$	Polynomial of order d
$\exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|^2)$	Gaussian radial basis function
$\exp(-\gamma \|\mathbf{x} - \mathbf{x}'\|)$	Exponential radial basis function

Table 9.1 Examples of kernels. Increasing parameters $d \in \mathbb{N}$ or $\gamma \in \mathbb{R}$ increases the capacity of the classifier.

Mercer conditions

Reciprocally, many kernels admit series expansions (eventually infinite), and can therefore be written as a dot product (see Mercer's conditions in Theorem 1.16). Such is the case, for instance, of the polynomial and the Gaussian kernel (Table 9.1).

dual parameters

In the following, we will refer to the linear classifier (Perceptron) as the primal representation of the linear discriminant and to the corresponding kernel classifier as its dual. Therefore, $\mathbf{w}$ will be the primal parameters and $\alpha_i, 1 \le i \le m$ the dual parameters.

Depending on the number of features and the number of training examples, it may be beneficial to carry out the computations in one or the other representation. For instance, the polynomial kernel: $k(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + 1)^d$ expands into the dot product between Φ vectors whose components contain all possible products of up to d components of $\mathbf{x}$ vectors. The number of primal parameters to estimate $N \simeq n^d$ (the dimension of feature space) is often much larger than the number of dual parameters m (the number of training examples).

9.2.9 Linear Separability

Besides being linear in its parameters, the linear classifier is linear in its input components in the N-dimensional Φ-space. Thus it defines a linear decision boundary (an $(N-1)$-dimensional hyperplane). A classification problem is said to be linearly separable, for given basis functions $\Phi(\mathbf{x})$, if there exists a parameter vector $\mathbf{w}$ such that $g(\mathbf{x}) = \operatorname{sgn}(\mathbf{w} \cdot \Phi(\mathbf{x}))$ classifies all vectors $\mathbf{x}$ without error.

Non-linear separability can arise from a number of causes (Figure 9.2.10):

- Classes may overlap in feature space (case (a)), due to the existence of inherently *ambiguous* patterns, e.g., a vertical bar can either represent a digit `1` or an lowercase `l`). Separability can also be due to the presence of "noisy data," including *meaningless* patterns and *mislabeled* valid examples. In the case of overlapping classes, the ideal decision boundary may nevertheless be linear. For instance, two classes generated by Gaussian distributions of identical covariance matrices have a Bayes optimum decision boundary $g(\mathbf{x}) = \operatorname{sgn}(p(\mathbf{x}, y = 1) - p(\mathbf{x}, y = -1))$ that is linear.
- Classes may have an ideal decision boundary which is not linear (case (b)), as might arise in the case of complex multimodal distributions.

From the point of view of finding an optimum linear discriminant function, these two cases are quite different. In case (a), one must allow that some examples will be misclassified, i.e., that the asymptotic error rate of Bayes optimum decision function (Bayes error) is non zero. In case (b), the Bayes error is zero, and a better choice of representation may render the problem separable.

9.2.10 Linearly Separable Training Set

A linearly separable training set is not by itself an indication that the problem is linearly separable. Any sparse training set is likely to be linearly separable. A non linearly separable training set indicates that the problem is non linearly separable (provided that the training data does not contain any error). However, it does not provide a clue as to whether classes overlap or not.

Given a universal set of basis functions, it is always possible to separate the training set without error, for some value of the parameter of the associated kernel assuming no two identical training points have been assigned to different classes. For instance, one can always reduce the width (increase γ) of the Gaussian kernels centered on the training points. However, this approach tends to overfit the data and give poor classification error rate on unseen test data. Such overfitting can easily be understood since, in the limit of infinitely narrow kernels, any training set can be learned without error, but no prediction can be made on unseen data. The training examples are merely "memorized."

A general principle from learning theory predicts that for two classifiers having the same training error, the classifier with smaller capacity is more likely to perform better on unseen data [Vapnik, 1979]. Loosely speaking, the capacity is related to the number of training examples that can be separated without error. Therefore, increasing the kernel parameter allows to learn easily more training examples, but simultaneously increases capacity. Introducing some errors on the training set by reducing the capacity may yield a better classifier.

Note that this is completely independent on whether the data is "noisy" or not.

9.3 Formulations of the Linear Discriminant Training Problem

The goal of linear discriminant function training is to adjust its parameters such that the expected value of the classification error on unseen patterns (referred to as expected risk or prediction error) is minimized. Note that only the parameters $\mathbf{w}$ or $\boldsymbol{\alpha}$ are being trained in this framework. The additional parameters of the basis functions or kernel functions are not subject to training.

Under the assumption that the data is generated by a probability distribution $p(\mathbf{x}, y)$, the prediction error is given by:

$$R(f) = \int_{\mathbb{R}^N \times \{-1,1\}} 1_{\{g(\mathbf{x}) \neq y\}} \; p(\mathbf{x}, y) d\mathbf{x} dy. \tag{9.7}$$

where $g(\mathbf{x}) = \operatorname{sgn}(f(\mathbf{x}))$ and $f(\mathbf{x})$ is a linear discriminant function.

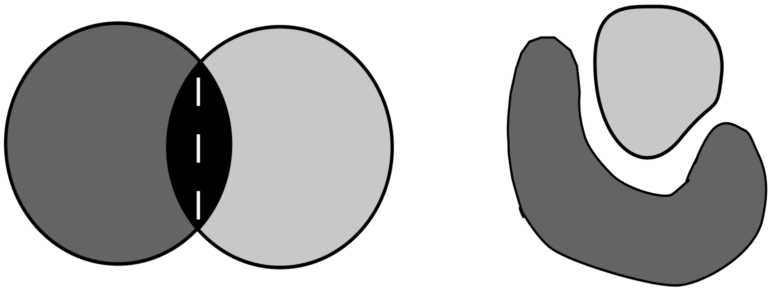

Figure 9.1 Non-linear separability. (a) Overlapping classes. The optimum decision boundary may still be linear. (b) Non overlapping classes. In the case shown, the optimum decision boundary is not linear.

The prediction $R(f)$ cannot be computed when $p(\mathbf{x}, y)$ is unknown, it can only be approximated. The various training methods that have been proposed all use a particular way of approximating $R(f)$, which is always optimum in some sense. We have selected a subset of methods that have some interesting connections to Support Vector Classifiers.

9.3.1 Minimizing the Number of Training Errors

The first method consists in estimating $R(f)$ by its discrete approximation computed with the training examples (the empirical risk or training error):

$$R_{emp}(f) = \sum_{i=1}^{m} 1_{\{g(\mathbf{x}_i) \neq y_i\}}. \tag{9.8}$$

The empirical risk functional $R_{emp}(f)$ is piecewise constant, which is typically difficult to optimize by standard techniques such as gradient descent.

9.3.1.1 Perceptron Objective Function

Perceptron objective function

One of the most popular substitute objective functions is the Perceptron objective function. The Perceptron objective function is equal to the sum of the "margin values" of the misclassified examples:

$$J_{\text{Perceptron}}(f) = -\sum_{i \in \mathcal{M}} y_i f(\mathbf{x}_i) = -\sum_{i \in \mathcal{M}} \rho_f(\mathbf{x}_i, y_i) \tag{9.9}$$

where $\mathcal{M} = \{i, i = 1, \ldots, m : g(\mathbf{x}_i) = \operatorname{sgn}(f(\mathbf{x}_i)) \neq y_i\}$ is the set of misclassified examples and $\rho_f(\mathbf{x}_i, y_i) = y_i f(\mathbf{x}_i) = y_i \mathbf{w} \cdot \Phi(\mathbf{x})$ are the margin values. The margin values are proportional to the distances from the misclassified examples to the decision boundary in Φ-space. The misclassified examples have negative margin values: $\rho_f(\mathbf{x}_i, y_i) < 0$.

If the training set is linearly separable, a set of parameters that minimizes $J_{\text{Perceptron}}(f)$ also minimizes $R_{emp}(f)$ of Equation 9.8. If the training set is not linearly separable, the minimization of $R_{emp}(f)$ is not guaranteed, but a set of parameters that minimizes $J_{\text{Perceptron}}(f)$ may nevertheless yield an acceptable practical solution.

9.3.2 Approximating the Bayes Optimum Discriminant Function

Bayes optimum discriminant

A different approach is to approximate the Bayes optimum discriminant function by a linear discriminant. By minimizing an approximation of $R(f)$, we were trying to find a decision function $g(\mathbf{x}) = \operatorname{sgn}(f(\mathbf{x}))$, where $f(\mathbf{x})$ is a linear discriminant, that minimizes the number of classification errors. We now consider the decision function $g_{\text{Bayes}}(\mathbf{x}) = \operatorname{sgn}(f_{\text{Bayes}}(\mathbf{x}))$, which gives the theoretical minimum number of classification errors. It is based on a discriminant function, e.g., $f_{\text{Bayes0}}(\mathbf{x}) = p(\mathbf{x}, y = 1) - p(\mathbf{x}, y = -1)$, which we approximate with a linear discriminant function. We have therefore replaced the problem of training a classifier by that of estimating a probability density.

Any discriminant function $f(\mathbf{x})$ can be transformed by a monotonically increasing function without changing the outcome of the decision function $g(\mathbf{x}) = \operatorname{sgn}(f(\mathbf{x}))$. The Bayes decision function can therefore be constructed with several discriminant functions:

$$f_{\text{Bayes0}}(\mathbf{x}) = p(\mathbf{x}, y = 1) - p(\mathbf{x}, y = -1) \tag{9.10}$$

$$f_{\text{Bayes1}}(\mathbf{x}) = P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x}) \tag{9.11}$$

$$f_{\text{Bayes2}}(\mathbf{x}) = \log\left(\frac{p(\mathbf{x}|y = 1)}{p(\mathbf{x}|y = -1)}\right). \tag{9.12}$$

9.3.2.1 MSE Objective Function

A first method consists in seeking the linear discriminant $f(\mathbf{x})$ that best approximates $f_{\text{Bayes1}}(\mathbf{x}) = P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$ in the least squares sense. This function corresponds to the minimum of the objective function:

$$\int_{\mathbb{R}^N \times \{-1,1\}} (f_{\text{Bayes1}}(\mathbf{x}) - f(\mathbf{x}))^2 p(\mathbf{x}, y) d\mathbf{x} dy. \tag{9.13}$$

This problem seems to require the knowledge of $f_{\text{Bayes1}}(\mathbf{x})$. However, it can be shown (see, e.g., [Duda and Hart, 1973]) that the linear discriminant that minimizes this objective function also minimizes:

$$\int_{\mathbb{R}^N \times \{-1,1\}} (y - f(\mathbf{x}))^2 p(\mathbf{x}, y) d\mathbf{x} dy. \tag{9.14}$$

Because the probability densities are unknown, the expected value of the objective function is replaced by its empirical estimate on the training set. The problem is brought back to that of minimizing the mean squared error objective function:

mean square error

$$J_{\mathrm{MSE}}(f) = \sum_{i=1}^{m}(y_i - f(\mathbf{x}_i))^2 = \sum_{i=1}^{m}(1-\rho_f(\mathbf{x}_i, y_i))^2, \tag{9.15}$$

where $\rho_f(\mathbf{x}_i, y_i) = y_i f(\mathbf{x}_i)$ are the margin values.

9.3.2.2 Logistic Regression

logistic regression

The logistic regression method, which uses the *maximum likelihood* framework, optimizes a different objective function. It is assumed that $f_{\mathrm{Bayes2}}(\mathbf{x}) = \log[p(\mathbf{x}|y = 1)/p(\mathbf{x}|y = -1)]$ follows a model linear in its parameters: a linear discriminant $f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x})$. One seeks the parameters of $f(\mathbf{x})$ that maximize the likelihood of the training data, assuming that the model is correct. We derive below an objective function for logistic regression in our notations.

Maximizing the likelihood function:

$$\mathcal{L}(\mathbf{w}) = \prod_{i=1}^{m} p(\mathbf{x}_i, y_i|\mathbf{w}) \tag{9.16}$$

is equivalent to minimizing the cross-entropy objective function:[2]

$$J_{\mathrm{Cross\_entropy}}(f) = -\sum_{i=1}^{m} z_i \log(p_i) + (1-z_i)\log(1-p_i), \tag{9.17}$$

where $z_i = \frac{y_i+1}{2}$, $p_i = P(y_i = 1|\mathbf{x}_i)$. Using Bayes inversion, we can express p_i as:

$$p_i = \frac{1}{1+e^{-f_{\mathrm{Bayes2}}(\mathbf{x})}} = \mathrm{logistic}(f_{\mathrm{Bayes2}}(\mathbf{x})) \tag{9.18}$$

where $\mathrm{logistic}(z) = 1/(1+e^{-z}) = (1+\tanh(z/2))/2$ is the logistic function. By replacing p_i and z_i by their value in Equation 9.17, the problem can be brought back to that of minimizing:

$$2(J_{\mathrm{Cross\_entropy}}(f) + log(2)) = -\sum_{i=1}^{m}(1+y_i)\log\left(1+\tanh\left(\frac{f_{\mathrm{Bayes2}}(\mathbf{x})}{2}\right)\right) \tag{9.19}$$

$$+ (1-y_i)\log\left(1-\tanh\left(\frac{f_{\mathrm{Bayes2}}(\mathbf{x})}{2}\right)\right), \tag{9.20}$$

Under the hypothesis that $f_{\mathrm{Bayes2}}(\mathbf{x})$ follows a model linear in its parameters, the objective function becomes:

2. The solution to an optimization problem is not modified by transforming the objective function with a smooth monotically increasing function such as the log and/or removing or adding positive additive or multiplicative constants.

$$J_{\text{Logistic}}(f) = -\sum_{i=1}^{m} \log(1 + \tanh(\rho_f(\mathbf{x}_i, y_i)/2)), \tag{9.21}$$

where $\rho_f(\mathbf{x}_i, y_i) = y_i f(\mathbf{x}_i)$ are the margin values and $f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x})$.
From the axioms of probability, $P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$ always lies between -1 and $+1$. In the MSE approach, the estimate of $P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$ is the linear discriminant itself $f(\mathbf{x}_i)$, which is not limited to taking values between -1 and $+1$.[3] The logistic function approach is consistent with the axioms of probabilities in that its estimate of $P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$, which is $\tanh(f(\mathbf{x})/2)$, always lies between -1 and $+1$.

neural networks

The similarity between the logistic regression approach and the training of a one layer neural network can be noticed. The tanh function is nothing but a sigmoid. Commonly, neural networks are trained with "back-propagation," a gradient descent method using an MSE objective function [Rumelhart et al., 1986b]. The cross-entropy objective function is also sometimes used.

9.3.2.3 Weight Decay

Learning theoretic predictions of classifier performance indicate that if two classifiers have the same training error, the classifier with smallest capacity is more likely to perform better. This is particularly important when problems are very underdetermined, that is when the number of training examples is small compared to the number of parameters to be estimated. Such is the case when a large number of basis functions $\Phi_k(\mathbf{x})$ is used, e.g., in the case of a polynomial classifier.

weight decay

In an effort to choose one solution among many that minimize $J_{\text{MSE}}(f)$ or $J_{\text{Logistic}}(f)$, additional constraints can be added. One popular constraint is to pick the solution that has minimum weight vector norm ($\min \|\mathbf{w}\|$). This is achieved by adding a penalty term to the objective functions, known as "weight decay" term, e.g.,

regularizer

$$J_{\text{WD MSE}}(f) = \sum_{i=1}^{m} (1 - \rho_f(\mathbf{x}_i, y_i))^2 + \lambda \|\mathbf{w}\|^2, \tag{9.22}$$

where λ is a positive constant. The weight decay can be considered as a regularizer.

9.3.3 Maximizing the Minimum Margin

support vector classifier

As with nearest-neighbor classifiers and linear classifiers such as the Perceptron, the support vector classifier approach does not attempt to estimate probabilities.

3. It may seem contradictory that the MSE solution is an approximation to $P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$ and yet is not limited to taking values between -1 and $+1$. However, this approach is valid in the sense that asymptotic convergence is guaranteed.

Instead it provides directly a solution to the classification problem, as do empirical risk minimization methods.

The empirical risk minimization problem, as addressed for instance with the Perceptron method, also suffers from underdetermination. For a large number of basis functions, the training set is likely to be separable. If so, there are usually many linear discriminant functions that perform the separation without error.

In an effort to choose one solution among many, the support vector method advocates to choose the solution that leaves the largest possible margin on both sides of the decision boundary in Φ-space.

optimum margin classifier

There are many equivalent formulations of the optimum margin problem (see the introduction chapter of this book, Chapter 1). One of them resembles the Perceptron formulation (Equation 9.9). In our previous notations one must maximize the smallest possible "margin value" $\rho_f(\mathbf{x}_i, y_i)$, that is minimize the objective function:

$$J_{SVC1}(f) = -\min_{i, 1 \le i \le m} \rho_f(\mathbf{x}_i, y_i), \tag{9.23}$$

under the constraints that all examples are well classified and that the weight vector is normalized.

Another version resembles the MSE with weight decay formulation (Equation 9.22). One must minimize the norm of the weight vector under the constraints that the margin values are larger or equal to one. Using the method of Lagrange multipliers, the problem is brought back to that of optimizing the objective function:[4]

$$J_{SVC2}(f) = \|\mathbf{w}\|^2 + \sum_{i=1}^{m} \alpha_i (1 - \rho_f(\mathbf{x}_i, y_i)), \tag{9.24}$$

regularizer

under the constraints that $\alpha_i \ge 0, \forall i, 1 \le i \le m$.

In the introduction chapter of this book, Chapter 1, it is explained what type of regularizer is associated with the support vector classifier solution.

9.4 Training Algorithms

There exist a variety of optimization techniques that search for the parameters that minimize the objective functions defined in the previous section. In this section, we focus only on simple gradient descent algorithms. We have several motivations for doing so. We want to provide algorithms that are easy to understand intuitively and easy to implement with a few lines of code. We want to emphasizes connections between the various training algorithms. We refer the advanced user interested in large scale real world problems to algorithms that are more efficient computationally, e.g., use mathematical programming, described elsewhere [Luenberger, 1973].

4. Optimizing means in this case finding the saddle point of $J_{SVC2}(f)$, which is a minimum with respect to $\mathbf{w}$ and a maximimum with respect to α_i.

9.4.1 Gradient Descent

gradient descent

The technique underlying all the algorithms presented in this section is called "gradient descent." Consider an objective function $J(\mathbf{w})$ of some parameters $\mathbf{w}$. Let $\nabla_{\mathbf{w}}$ be the gradient operator:

$$\nabla_{\mathbf{w}} = \left[\frac{\partial}{\partial w_1}, \dots, \frac{\partial}{\partial w_N}\right]^t. \tag{9.25}$$

At the solution of the minimization problem, we have:

$$\nabla_{\mathbf{w}} J(\mathbf{w}) = \mathbf{0}. \tag{9.26}$$

The gradient descent technique consists in iteratively approaching the solution by making small steps in steepest direction on the slope of the objective function, as given by the negative gradient:

$$\mathbf{w} \leftarrow \mathbf{w} - \eta \nabla_{\mathbf{w}} J(\mathbf{w}), \tag{9.27}$$

stochastic gradient

where η is a positive value, called the "learning rate."

We distinguish two variants of gradient descent procedures: true gradient and single sample gradient. The true gradient (or batch gradient) method is the method described above. The single sample gradient method (also sometimes called "on-line" or "stochastic" gradient) uses the gradient computed on the cost incurred by a single sample; it updates one sample at a time with this "local" gradient value. We will describe true gradient methods, from which single sample gradient methods can trivially be inferred. Single sample methods are often preferred in practice because of simplicity and speed of convergence.

9.4.2 Algorithms for Linearly Separable Training Sets

Perceptron algorithm

If the training set is linearly separable in Φ-space, it can be shown that minimizing the Perceptron objective function with gradient descent yields a linear discriminant function with zero training error in a finite number fixed size steps (see the introduction chapter of this book, Chapter 1).

We first use the primal expression of the linear discriminant: $f(\mathbf{x}) = \mathbf{w} \cdot \Phi(\mathbf{x})$. The gradient of the objective function $J_{\text{Perceptron}}(f)$ (Equation 9.9) is given by:

$$\nabla_{\mathbf{w}} J_{\text{Perceptron}}(f) = -\sum_{i \in \mathcal{M}} y_i \Phi(\mathbf{x}_i), \tag{9.28}$$

where $\mathcal{M} = \{i, i = 1, \dots, m : g(\mathbf{x}_i) = \operatorname{sgn}(f(\mathbf{x}_i)) \neq y_i\}$ is the set of misclassified examples. From this expression, we can derive algorithm 9.1. By substituting the dual expression of the linear discriminant $f(\mathbf{x}) = \sum_{j=1}^m y_j \alpha_j k(\mathbf{x}, \mathbf{x}_j)$, one obtains an alternative dual version. Note that the increments on the αs are of fixed size. This algorithm differs from performing gradient descent in $\boldsymbol{\alpha}$-space directly.

The dual Perceptron algorithm has been known for a long time [Aizerman et al., 1964]. It is a simple way of training kernel classifiers, which is proven to converge in a finite number of steps, as per the Perceptron convergence theorem.

Algorithm 9.1 : Primal and Dual Perceptron

Arguments: Training sample, $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_m\} \subset \mathcal{X}$, $Y = \{y_1, \ldots, y_m\} \subset \{\pm 1\}$.
Returns: Decision function g, parametrized with
primal parameters w (dim. N Φ-space coordinate weight vector)
or dual parameters $\boldsymbol{\alpha}$ (dim. m sample weight vector).

Function Perceptron(X, Y)
  Initialize arbitrarily, e.g., $\mathbf{w} = \sum_i y_i \Phi(\mathbf{x}_i)$ or $\boldsymbol{\alpha} = 1$.
  repeat
    for all i from $i = 1, \ldots, m$
      Compute the discriminant function values:
      $f(\mathbf{x}_i) = \mathbf{w} \cdot \Phi(\mathbf{x}_i) = \sum_j y_j \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$.
      Compute the margin values: $\rho_f(\mathbf{x}_i, y_i) = y_i f(\mathbf{x}_i)$.
    endfor
    Update on the set of misclassified training examples,
    $\mathcal{M} = \{i, i = 1, \ldots, m : \rho_f(\mathbf{x}_i, y_i) < 0\}$:
    for all $i \in \mathcal{M}$
      primal parameters $\mathbf{w} \leftarrow \mathbf{w} + y_i \Phi(\mathbf{x}_i)$, or
      dual parameter $\alpha_i \leftarrow \alpha_i + 1$.
    endfor
  until all examples are well classified: $\rho_f(\mathbf{x}_i, y_i) > 0$.
  return $g : \mathbf{x} \mapsto \operatorname{sgn}(\mathbf{w} \cdot \Phi(\mathbf{x})) = \operatorname{sgn}(\sum_j y_j \alpha_j k(\mathbf{x}, \mathbf{x}_j))$.
end

Optimum margin Perceptron algorithm

Since the 1992 paper of Boser et al. [Boser et al., 1992], support vector classifiers are often associated with quadratic programming (QP); because of the perceived complexity of QP, this method has had a relatively small impact among pattern recognition practitioners. While we still recommend using QP for large scale real world problems, we present here an alternate algorithm (Table 9.2), which is very simple, and converges to the same solution asymptotically, for $\theta \to \infty$ [Krauth and Mézard, 1987].

Minover

This algorithm closely resembles the Perceptron algorithm. At each iteration, instead of updating the weight vector with the sum of the misclassified examples, one updates with a single example: the example with smallest "margin value." The name of the algorithm is "Minover," for minimum overlap. The concept originated in the physics community where overlap is a synonym for margin value.

Algorithm 9.2 : Primal and Dual Optimum Margin Perceptron (Minover)

```
Arguments:  Training sample, X = {x_1, ..., x_m} ⊂ 𝒳, Y = {y_1, ..., y_m} ⊂ {±1};
            stopping criterion, θ ∈ ℝ⁺.
Returns:    Decision function g, parametrized with
            primal parameters w (dim. N Φ-space coordinate weight vector)
            or dual parameters α (dim. m sample weight vector).
```

Function `Minover`(X, Y, θ)

Initialize arbitrarily, e.g., $\mathbf{w} = \sum_i y_i \Phi(\mathbf{x}_i)$ or $\boldsymbol{\alpha} = 1$.

repeat

  for all i from $i = 1, \ldots, m$

    Compute the discriminant function values:
    $f(\mathbf{x}_i) = \mathbf{w} \cdot \Phi(\mathbf{x}_i) = \sum_j y_j \alpha_j k(\mathbf{x}_i, \mathbf{x}_j)$.

    Compute the margin values: $\rho_f(\mathbf{x}_i, y_i) = y_i f(\mathbf{x}_i)$.

  endfor

  Find the training example $(\mathbf{x}^c, y^c)$, which is ''worst classified,''
  i.e., has smallest margin value: $(\mathbf{x}^c, y^c) = \operatorname{argmin}_i \rho_f(\mathbf{x}_i, y_i)$.
  Update primal parameters $\mathbf{w} \leftarrow \mathbf{w} + y^c \Phi(\mathbf{x}^c)$, or
  increment dual parameter $\alpha^c \leftarrow \alpha^c + 1$.

until $\rho_f(\mathbf{x}^c, y^c) > \theta$.

return $g : \mathbf{x} \mapsto \operatorname{sgn}(\mathbf{w} \cdot \Phi(\mathbf{x})) = \operatorname{sgn}(\sum_j y_j \alpha_j k(\mathbf{x}, \mathbf{x}_j))$.

end

Both the Perceptron and the Minover algorithms are independent on the learning rate. The norm of the weight vector increases during learning. One can optionally normalize the weight vector at the end of the training procedure.

Support vector classifiers draw their name from the fact that their discriminant function is a function only of a small number of training examples, called support vectors. Those are the examples with smallest margin value, which are closest to the decision boundary in Φ-space.

informative patterns

The learning mechanism of the Minover algorithm reveals an important aspect of support vectors. Support vectors used to be called "informative patterns." Indeed, in the process of learning, the weight vector is updated only with those patterns that are hardest to predict (have smallest margin value). In the information theoretic sense, the examples that are least predictable are the most informative.

9.4.3 Algorithms for Non Linearly Separable Training Sets

As mentioned in Section 9.2.9, a non linearly separable training set may indicate that the selection of basis functions is not adequate or that the capacity is insufficient. This can be remedied by changing the basis functions (or the kernel) and/or increasing the number of basis functions (or the kernel parameter) to increase the classifier capacity. But it may also indicate overlapping classes or "noisy" data (meaningless or mislabeled examples), in which case it may be detrimental to increase the capacity. Also, we may want to reduce the capacity at the expense of introducing some training error in the hope of obtaining a better prediction error. In these last cases, we need learning algorithms that can converge to a solution with non zero training error.

Although the Perceptron algorithm does not converge in the non-separable case, there exist algorithms (e.g., linear programming) that minimize the Perceptron objective function even in the non-separable case.

Pseudo-inverse and least mean square

Training a linear discriminant function can be thought of as solving a system of linear inequalities:

$$y_i(\mathbf{w} \cdot \Phi(x_i)) > 0, \;\; i = 1, \ldots, m$$

Traditionally, people have often replaced this problem by that of solving a system of equations:

$$y_i(\mathbf{w} \cdot \Phi(x_i)) = 1, \;\; i = 1, \ldots, m$$

that provides acceptable solutions in both the separable and non separable case, albeit at the expense of making eventually errors on the training examples in the separable case.

The treatment of the problem is simplified by introducing matrix notations:

$$\mathbf{w}^t \mathbf{\Phi} = \mathbf{y}^t$$

where $\mathbf{w}$ is a $(N, 1)$ matrix, $\mathbf{\Phi} = [\Phi(x_1), \ldots, \Phi(x_m)]$ is a (N, m) matrix, and $\mathbf{y}$ is a $(m, 1)$ matrix. If $\mathbf{\Phi}$ is invertible, the solution can be computed as: $\mathbf{w}^t = \mathbf{y}^t \mathbf{\Phi}^{-1}$. When $\mathbf{\Phi}$ is rectangular, the system is either under or over determined. But one can always seek the best solution in the least square sense, which is given by:

$$\mathbf{w}^t = \mathbf{y}^t \mathbf{\Phi}^+$$

pseudo-inverse

where $\mathbf{\Phi}^+$ is the Moore-Penrose pseudo-inverse, for which many computational algorithms exist [Albert, 1972].

It can be shown that the pseudo-inverse solution minimizes $J_{\mathrm{MSE}}(f)$ (Equation 9.15). Moreover, in the case where the system is underdetermined ($N = dim(\mathbf{w}) > m$) the pseudo-inverse solution is the solution of minimum norm. It is the minimum of $J_{\mathrm{WD\ MSE}}(f)$ (Equation 9.22), for $\lambda \to 0$.

It is also possible to minimize the mean squared error $J_{\mathrm{MSE}}(f)$ with a gradient descent method. The gradient of $J_{\mathrm{MSE}}(f)$ is:

$$\nabla_{\mathbf{w}} J_{\mathrm{MSE}}(f) = -2 \sum_{i=1}^{m} y_i (1 - \rho_f(\mathbf{x}_i, y_i)) \Phi(\mathbf{x}_i).$$

The weights must be initialized to zero to converge to the solution with minimum norm, the learning rate must decrease with the number of iterations, e.g., $\eta(t) = 1/t$.

Algorithm 9.3 : Primal and Dual Gradient Descent

```
Arguments:  Training sample, X = {x_1, ..., x_m} ⊂ 𝒳, Y = {y_1, ..., y_m} ⊂ {±1}.
            learning rate η(t) ∈ ℝ^+,
            stopping criterion, θ ∈ ℝ^+ and τ ∈ ℕ.
Returns:    Decision function g, parametrized with
            primal parameters w (dim. N Φ-space coordinate weight vector)
            or dual parameters α (dim. m sample weight vector).

Function Gradient(X, Y, η(t), θ)
   Initialize w = 0 or α = 0 and t = 0.
   repeat
       for all i from i = 1, ..., m
            Compute the discriminant function values:
            f(x_i) = w · Φ(x_i) = Σ_j y_j α_j k(x_i, x_j).
            Compute the squashed margin values: σ(ρ_f(x_i, y_i)) = σ(y_i f(x_i)).
       endfor
       Update on all training examples,
       for all i from i = 1, ..., m
            primal parameters w ← w + η(t) y_i [1 − σ(ρ_f(x_i, y_i))] Φ(x_i), or
            dual parameter α_i ← α_i + η(t)[1 − σ(ρ_f(x_i, y_i))].
       endfor
       Increment t: t ← t + 1.
   until empirical objective function is small enough J(f) < θ or
       maximum number of iterations is exceeded t > τ.
   return g : x ↦ sgn(w · Φ(x)) = sgn(Σ_j y_j α_j k(x, x_j)).
end
```

Depending on the choice of the squashing function, one obtains various algorithms. (1) Perceptron: $\sigma(z) = \mathrm{sgn}\,(z)$. (2) MSE: $\sigma(z) = z$. (3) Logistic regression: $\sigma(z) = \tanh(z/2)$. (4) Neural soft margin: $\sigma(z) = 1$, if $z < -1$; $\sigma(z) = z$, if $-1 \leq z \leq 1$; $\sigma(z) = 1$, if $z > 1$.

This algorithm highlights the fact, which is not readily apparent in the pseudo-inverse solution, that the MSE solution is a weighted combination of the training patterns. As such, it admits a dual version. Indeed, the Moore-Penrose pseudo-inverse has the following property[5] [Albert, 1972]:

$$\mathbf{\Phi}^+ = (\mathbf{\Phi}^t\mathbf{\Phi})^+\mathbf{\Phi}^t = \mathbf{\Phi}^t(\mathbf{\Phi}\mathbf{\Phi}^t)^+$$

Hence the MSE linear discriminant function has dual forms:

$$f(\mathbf{x}) = \mathbf{w}^t\Phi(\mathbf{x}) = \mathbf{y}^t\mathbf{\Phi}^+\Phi(\mathbf{x}) = \mathbf{y}^t(\mathbf{\Phi}^t\mathbf{\Phi})^+\mathbf{\Phi}^t\Phi(\mathbf{x}) \tag{9.29}$$

$$= \mathbf{y}^t K^+\boldsymbol{\kappa}(\mathbf{x}) = \beta^t\boldsymbol{\kappa}(\mathbf{x}) = \sum_{i=1}^{n} y_i\alpha_i k(\mathbf{x}, \mathbf{x}_i) \tag{9.30}$$

5. $(\mathbf{\Phi}^t\mathbf{\Phi})^+ = (\mathbf{\Phi}^t\mathbf{\Phi})^{-1}$ if the columns of $\mathbf{\Phi}$ are linearly independent and $(\mathbf{\Phi}\mathbf{\Phi}^t)^+ = (\mathbf{\Phi}\mathbf{\Phi}^t)^{-1}$ if the rows of $\mathbf{\Phi}$ are linearly independent. This provides one way of computing the pseudo-inverse.

where $K = \mathbf{\Phi}^t\mathbf{\Phi} = [k(\mathbf{x}_i, \mathbf{x}_j)]_{1\leq i\leq m, 1\leq j\leq m}$ is a (m, m) matrix to be pseudo-inverted, $\boldsymbol{\kappa}(\mathbf{x}) = \mathbf{\Phi}^t\Phi(\mathbf{x}) = [k(\mathbf{x}, \mathbf{x}_i)]_{1\leq i\leq m}$ is a $(m, 1)$ matrix, and $\boldsymbol{\beta}^t = \mathbf{y}^t K^+$ is a $(1, m)$ matrix of elements $y_i\alpha_i$.

Interestingly, the pseudo-inverse/MSE solution coincides with the optimum margin solution when all the training examples are support vectors.

Logistic regression

Another very similar algorithm is obtained by minimizing $J_{\text{logistic}}(f)$. The gradient of $J_{\text{logistic}}(f)$ with respect to $\mathbf{w}$ is:

$$\nabla_{\mathbf{w}} J_{\text{logistic}}(f) = -\frac{1}{2}\sum_{i=1}^{m} y_i(1 - \tanh(\rho_f(\mathbf{x}_i, y_i)/2))\Phi(\mathbf{x}_i).$$

As shown in Algorithm 9.3 the only change in the algorithm is to pass the margin values through a squashing function.

back-propagation

Note that, unlike the back-propagation algorithm that trains one layer neural networks with gradient descent using the MSE objective function, the weight update is not multiplied by the derivative of the sigmoid (tanh) function.

A Minover algorithm for soft margin classifier

One may wonder whether it is possible to obtain a support vector classifier for non separable training data, by extending the idea of an optimum margin.

negative margin

The first idea that comes to mind is to minimize the negative margin, instead of maximizing the positive margin. There are a number of reasons why this is not a good idea. First the solution may not be unique. It is easy to construct examples for which there exist several equivalent negative margin solutions (see Figure 9.4.3) [Lambert, 1969]. Second, the negative margin is solely defined by misclassified examples, which may be "bad" examples to rely on.

Another possibility is to keep maximizing the positive margin, but allow that a number of training examples be misclassified, with a certain penalty for each misclassified example. This is the idea behind the soft margin algorithm [Cortes and Vapnik, 1995] and ν-SVC [Schölkopf et al., 1998c] .

soft margin

The quadratic programming formulation of the soft margin algorithm and ν-SVC are very similar to that of the regular maximum margin algorithm. For the soft margin algorithm, there is only a set of additional constraints on the Lagrange multipliers: $\alpha_i \leq C$, where C is a positive constant. One can trivially extend the Minover algorithm to compute the soft margin solution (Algorithm 9.4).[6]

While the original Minover algorithm insists on trying to learn examples that have negative margin values and cannot be well classified, the soft Minover algorithm gives them up after a while and continues with the remaining examples.

6. ν-SVC provides a more explicit control over the number of non-marginal support vectors, but it does not lend itself to a simple extention of the Minover algorithm.

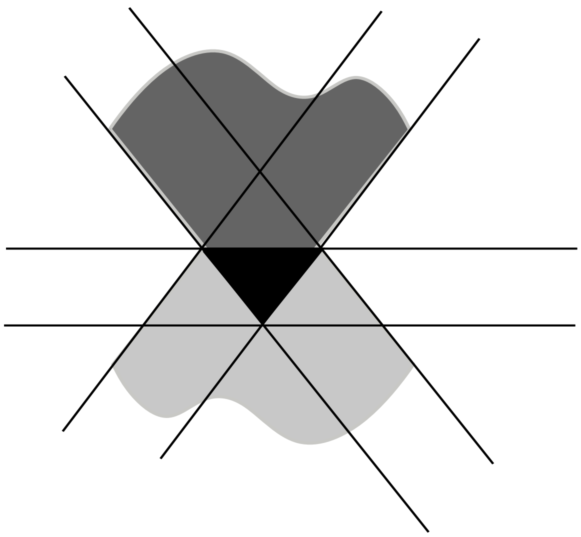

Figure 9.2 Negative margin. Examples can be constructed that have several equivalent negative margin solutions.

Neural networks and large margin classifiers

In the separable training set case, if all the training examples are support vectors, the pseudo-inverse/MSE solution is also the maximum margin solution. This is not the case in general.

The Kernel-Adatron method [Frieß et al., 1998] is a simple modification of the MSE update (Algorithm 9.3 (2)), for which α_i is replaced by zero if it becomes negative. It can be shown to converge to the maximum margin solution. The soft margin constraints $\alpha_i \leq C$ can similarly be enforced by replacing α_i by C if it goes overbound.

One of the effects of the squashing function in the logistic regression is to limit the influence of very well classified examples (with large positive margins). Another effect is to limit the influence of misclassified examples (with large negative margins). This is even emphasized in the back-propagation algorithm, because the weight update is multiplied by the derivative of the sigmoid (tanh) function.

These two effects are similar in spirit to the constraints imposed on α_i by the soft margin algorithm: $0 \leq \alpha_i \leq C$.

If the tanh function is replaced by a piecewise linear squashing function ($\sigma(z) = -1$, if $z < -1$; $\sigma(z) = z$, if $-1 \leq z \leq 1$; $\sigma(z) = 1$, if $z > 1$), another algorithm that computes a large margin solution is obtained. If an example is very well classified ($\rho_f(\mathbf{x}_i, y_i) \geq 1$), it does not contribute to the weight update. If an example is very misclassified ($\rho_f(\mathbf{x}_i, y_i) \leq -1$), it contributes a fixed maximum increment.

Algorithm 9.4 : Primal and Dual Soft Margin Perceptron (SoftMinover)

```
Arguments:  Training sample, X = {x_1,...,x_m} ⊂ 𝒳, Y = {y_1,...,y_m} ⊂ {±1};
            stopping criterion, θ ∈ ℝ⁺;
            soft margin regularization constant, C ∈ ℝ⁺.
Returns:    Decision function g, parametrized with
            primal parameters w (dim. N Φ-space coordinate weight vector)
            or dual parameters α (dim. m sample weight vector).

Function SoftMinover(X, Y, θ, C)
  Initialize: w = Σ_i y_i Φ(x_i) and α = 1.
  repeat
     for all i from i = 1,...,m
          Compute the discriminant function values:
          f(x_i) = w · Φ(x_i) = Σ_j y_j α_j k(x_i, x_j).
          Compute the margin values: ρ_f(x_i, y_i) = y_i f(x_i).
     endfor
     Find the training example (x^c, y^c), which is ''worst classified,''
     i.e., has smallest margin value: (x^c, y^c) = argmin_i ρ_f(x_i, y_i).
     if the corresponding α^c goes over bound: α^c > C
          Take the example (x^c, y^c) out of the training set.
     else
          Update primal parameters w ← w + y^c Φ(x^c), and
          increment dual parameter α^c ← α^c + 1.
     endif
  until ρ_f(x^c, y^c) > θ or the training set is empty.
  return g : x ↦ sgn(w · Φ(x)) = sgn(Σ_j y_j α_j k(x, x_j)).
end
```

In the process of learning, the weight vector increases and more and more examples having margin values greater than one become inactive. The algorithm progressively focuses on the marginal examples. If the algorithm is initialized with zero weights, in the separable training set case, it converges to the MSE solution of minimum norm computed on the marginal examples only. This is precisely the maximum margin solution.

neural soft margin

In the non-separable case, the algorithm ends up focusing on the misclassified examples. The backpropagation variant that multiplies the update by the derivative of the squashing function may be preferable. If an example is very misclassified ($\rho_f(\mathbf{x}_i, y_i) \leq -1$), it does not contribute to the weight update. This implements a kind of soft margin algorithm (9.3) (4) that we refer to as "neural soft margin."

9.5 Which Linear Discriminant?

While there is academic value in comparing methods, in practice it is often difficult to know which linear discriminant method is best suited to the classification problem at hand. As we have already pointed out, each approach has strengths and weaknesses, and may be optimum on some criterion. Although recent results — both theoretical and experimental — argue in favor of SVCs from the point of view of generalization (see the introduction chapter of this book, Chapter 1), occasionally other practical considerations may be important, as we now consider.

9.5.1 Feature Selection and Computational Burden

A fact that is not readily apparent from our presentation is that most linear discriminant methods suffer from intrinsic computational limitations. As discussed in Section 9.2, for a good choice of the basis functions, linear discriminants yield classification functions that can approximate the Bayes optimum decision function arbitrarily well. One approach may be to select a polynomial classifier of high order, that is, to use all the monomials up to a certain order as basis functions. Training such a classifier in the primal representation using traditional methods is impractical since the number of parameters to estimate is too large. In the dual representation, training is possible only if the number of training examples is not too large. Traditionally, this problem is addressed with various auxiliary techniques of feature selection (in primal space) or example selection (in dual space).

The advantage of SVCs is that they avoid this difficult feature selection step. Computationally, one can capitalize on the fact that the solution is a function only of the SVs, a small subset of the training patterns, where learning relies on quadratic programming in the dual space [Boser et al., 1992]. In particular, SVCs perform an automatic feature selection and example selection via the selection of SVs.

9.5.2 Probabilistic Interpretation of the Scores

So far, we have only considered discriminant functions as a means to a final classification decision; only the sign of $f(\mathbf{x})$ matters for this purpose. However, when a classifier is integrated as part of a larger system, the analog value of $f(\mathbf{x})$ itself often provides valuable information. For example, consider the design of a recognition system for zipcodes, unsegmented strings of digits. A single digit classifier may be integrated into such a zipcode recognition system in the following way. The overall system tries several heuristic segmentations of the input string; for each such "tentative" segmentation, the various segments are submitted to the digit classifier. Here, it is important that the digit classifier return a score or a confidence value rather than a simple digit classification so that an overall score for the whole string for various tentative segmentations can be computed. The highest such score is selected, yielding the chosen zipcode classification.

In other problems, it is convenient to use the value of $|f(\mathbf{x})|$ as a confidence value on which a threshold can be set in order to reject test examples that are ambiguous or meaningless, rather than making a classification error. The question is therefore how well $f(\mathbf{x})$ serves as a score for various linear discriminants. While it is not possible to answer that question for every situation, it is worth mentioning that both the MSE and the logistic regression methods approximate the optimum Bayes discriminant; their scores can therefore readily be interpreted in terms of probability estimates. We generally favor the logistic regression estimate because $f(\mathbf{x})$ provides an estimate of $f_{\text{Bayes1}} = P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$, which belongs to the $[-1, +1]$ interval while the MSE estimate does not.

The score provided by SVCs has no direct probabilistic interpretation. Nevertheless, there is an analogy between the tanh squashing function of logistic regression and the piecewise linear squashing function of the "neural soft margin." This suggests that squashing the score of the SVC classifier with a tanh may be a good idea, if a heuristic probabilistic interpretation needs to be made. If additional data is available, it is also possible to add a postprocessor that remaps the scores to probabilities [Schürmann, 1996], e.g., by fitting the parameters of a sigmoid $\tanh(\mathbf{a}\cdot\mathbf{x}+b)$ (see Chapter 5).

9.5.3 Robustness, Regularization, Good and Bad Outliers

Another important practical question is the treatment of outliers. In the context of SVCs, outliers may be defined as a training examples with a small margin value. On one hand, SVCs tend to emphasize outliers, which are often found among the support vectors. On the other hand, robust statistics methods derived from MSE training and logistic regression go in the opposite direction and attempt to de-emphasize the importance of outliers [Hampel et al., 1986]. Increasing a regularization penalty is an effective method of reducing the importance of outliers. The soft margin method [Cortes and Vapnik, 1995] and ν-SVC [Schölkopf et al., 1998c] reconcile SVCs with robust statistics by limiting the influence of the worst outliers. For more precise robust statistics claims, see [Schölkopf et al., 1998c].

There is no single good nor single wrong method; nevertheless there are good and bad outliers. Good outliers are very informative: *ambiguous patterns* that help defining crisply the decision boundary or *rare patterns* that help defining the decision boundary in regions that are not densely populated. Bad outliers may be "informative" in the information theoretic sense — i.e., hard to predict — but very "non-informative" in practice, i.e., not useful. Such outliers include *mislabeled* and *meaningless* patterns. These bad outliers correspond to errors introduced in the data and nearly always reduce the accuracy of the final classifier.

The problem of robust methods is that they de-emphasize outliers regardless of whether they are informative of not. On the contrary, SVCs emphasize them equally blindly. The solution is called "data cleaning." Since cleaning by verifying all data entries is tedious, SVCs can be put to work in a bootstrap method. A first classifier is trained with unclean data. Then, its support vectors (ranked in order of decreasing αs) are examined and the "bad outliers" eliminated. The overall process is iterated until no such "bad" outliers remain [Guyon et al., 1996].

9.6 Conclusion

This chapter has explored the commonalties and relationships between linear discriminant functions and support vector classifiers. Naturally, we have not exhausted this subject, but we have described the following connections between "classical" linear discriminant and SVC: Similarities in the objective functions, which typically exhibit a tradeoff between minimizing the number of training errors and minimizing classifier complexity. Similarities in gradient descent algorithms, whose parameter update is proportional to $[1 - \sigma(\rho_f(\mathbf{x}_i, y_i))]$, where the margin value $\rho_f(\mathbf{x}_i, y_i))$ is proportional to the distance of the training example to the decision boundary, and $\sigma(\cdot)$ is a squashing function such as a sigmoid. Similarities in the way duality can be exploited during training to simplify computations. Similarities in the probabilistic interpretation of the scores, in particular how the function $\sigma(f(\mathbf{x}_i))$ can be thought of as an approximation to the Bayes optimum discriminant function $f_{\text{Bayes1}} = P(y = 1|\mathbf{x}) - P(y = -1|\mathbf{x})$. Of course, by emphasizing similarities we do not mean to minimize the differences and benefits of SVCs over other linear discriminant techniques, as described by many others (see the introduction chapter of this book, Chapter 1). For instance SVCs draw their unique properties from the existence of support vectors and one can capitalize on the fact that the solution is a function only of the support vectors, a small subset of the training patterns. Quadratic programming can be used to find the optimum margin solution in dual space [Boser et al., 1992]. Theoretical results and experimental evidence show that SVCs draw advantages from their unique use of support vectors. Other subjects — robustness, regularization good and bad outliers — deserve more attention but go beyond the scope of this chapter.

Acknowledgments

Isabelle Guyon would like to thank Léon Personnaz for his initial impulse to the ideas of using duality in pseudo-inverse and other linear discriminant algorithms, and Léon Bottou for pointing out the "neural soft margin" algorithm.