E0 370 Statistical Learning Theory **Lecture 2 (Aug 11, 2011)**

Support Vector Machines

Lecturer: Shivani Agarwal *Scribe: Shivani Agarwal*

1 Introduction

In the last lecture we introduced the probabilistic/batch learning setting, where a learner receives a finite sample of labeled training examples[1] $S = ((x_1, y_1), \ldots, (x_m, y_m))$, the x_i being instances in some instance space $\mathcal{X}$ and y_i being labels in some label space $\mathcal{Y}$, and the goal is to learn from these examples a function $h : \mathcal{X} \to \mathcal{Y}$ that predicts accurately labels of new instances. We also mentioned briefly the *binary classification* problem in this setting, where there are just two labels denoted $\mathcal{Y} = \{-1, 1\}$, and the loss incurred on predicting a label $\hat{y}$ when the true label is y is simply $\ell_{0\text{-}1}(y, \hat{y}) = \mathbf{1}(\hat{y} \neq y)$.

In this lecture, we will review the support vector machine (SVM) algorithm, one of the most well studied and widely used learning algorithms for binary classification (extensions of SVMs exist for a variety of other learning problems, including regression, multiclass classification, ordinal regression, ranking, structured prediction, and many others).

In its most basic form, the SVM is an algorithm that learns a linear threshold function. Specifically, let $\mathcal{X} \subseteq \mathbb{R}^n$, so that each instance in $\mathcal{X}$ is an n-dimensional real vector (e.g. a vector of gene expression values corresponding to a patient sample in bioinformatics, or a vector of pixel intensities corresponding to an image in computer vision). Then given a sample of labeled examples $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)) \in (\mathcal{X} \times \{-1, 1\})^m$, the basic SVM algorithm learns a classifier of the form[2]

$$h(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \tag{1}$$

for some $\mathbf{w} \in \mathbb{R}^n$ and $b \in \mathbb{R}$; here $\text{sign}(z) = 1$ if $z \geq 0$ and -1 otherwise, and $\mathbf{w} \cdot \mathbf{x}$ denotes the dot product between the vectors $\mathbf{w}$ and $\mathbf{x}$ given by $\mathbf{w} \cdot \mathbf{x} = \sum_{k=1}^{n} w_k x_k$. Thus for $\mathcal{X} \subseteq \mathbb{R}^n$, the basic SVM algorithm selects a classifier from the class of *linear threshold functions* or *linear classifiers* over $\mathcal{X}$:

$$\mathcal{H} = \{h : \mathcal{X} \to \{-1, 1\} \mid h(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) \text{ for some } \mathbf{w} \in \mathbb{R}^n \text{ and } b \in \mathbb{R}\} \ . \tag{2}$$

However the algorithm can also be used to learn nonlinear classifiers, as well as classifiers for instance spaces other than $\mathbb{R}^n$, via the use of 'kernel' functions; we will describe these later in the lecture. We start by considering the simple case of linearly separable data in $\mathbb{R}^n$.[3]

[1] Note that we are concerned primarily with supervised learning; problems of unsupervised learning (where one is given only instances $x_i \in \mathcal{X}$ drawn from some distribution on $\mathcal{X}$ and the goal is to estimate the distribution or obtain a clustering) or of semi-supervised learning (where one is given some labeled examples $(x_i, y_i) \in \mathcal{X} \times \mathcal{Y}$ and some unlabeled examples $x_j \in \mathcal{X}$ and the goal is to learn a predictive function $h : \mathcal{X} \to \mathcal{Y}$) are also of interest, but will not be covered in this course.

[2] Note that for $\mathcal{X} \subseteq \mathbb{R}^n$, we are using boldface notation $\mathbf{x}$ for vector instances in $\mathcal{X}$.

[3] Note that, in describing a learning algorithm, we do not need to be concerned with the probabilistic framework described in the previous lecture; we only need to specify how the algorithm constructs its output, which is a function $h : \mathcal{X} \to \mathcal{Y}$, from a given input, which is a training sample $S \in \cup_{m=1}^{\infty} (\mathcal{X} \times \mathcal{Y})^m$. The probabilistic framework plays a role in the theoretical analysis of the algorithm; specifically, it allows us to ask questions about what properties the output of the learning algorithm (the learned function) satisfies when the input (training sample) is drawn randomly from some distribution. In this sense, one can think of the probabilistic framework described last time as corresponding to a *probabilistic analysis* of a learning algorithm, much as is done in the probabilistic analysis of other algorithms in computer science, except that (a) the distribution considered on the inputs is not a specific fixed distribution, but rather can take the form of independent draws from any distribution D on $\mathcal{X} \times \mathcal{Y}$; (b) the analysis is used in a statistical setting where one is interested in obtaining confidence intervals on the error of the learned function for a given input (data sample); and (c) one is interested in convergence properties as the number of training examples approaches infinity. (Of course, the probabilistic framework and corresponding generalization considerations can also help in motivating the design of a learning algorithm.)

2 Linearly Separable Data – Hard Margin SVM

A training sample $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)) \in (\mathbb{R}^n \times \{-1, 1\})^m$ is said to be *linearly separable* if there exists a linear classifier $h(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$ which classifies all examples in S correctly, i.e. for which $y_i(\mathbf{w} \cdot \mathbf{x}_i + b) > 0 \ \forall i \in \{1, \ldots, m\}$. For example, Figure 1 (left) shows a training sample in $\mathbb{R}^2$ that is linearly separable, together with two possible linear classifiers that separate the data correctly (note that the decision surface of a linear classifier in 2 dimensions is a line, and more generally in $n > 2$ dimensions is a hyperplane). Which of the two classifiers is likely to give better generalization performance?

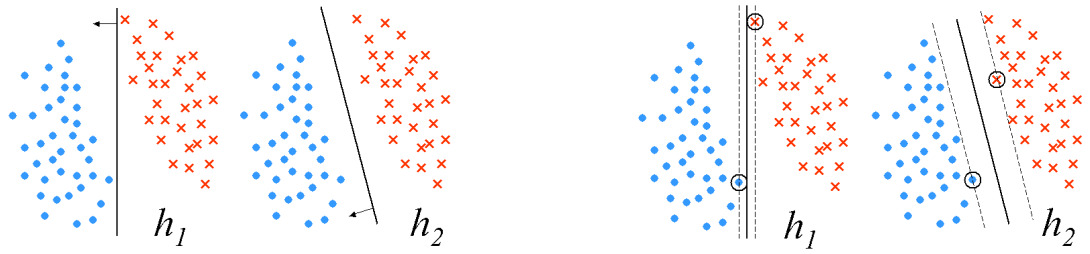

Figure 1: **Left:** A linearly separable data set, with two possible linear classifiers that separate the data. Blue circles represent class label 1 and red crosses -1; the arrow represents the direction of positive classification. **Right:** The same data set and classifiers, with margin of separation shown.

Although both classifiers separate the data, the distance or margin with which the separation is achieved is different; this is shown in Figure 1 (right). The SVM algorithm selects the *maximum margin* classifier, i.e. the linear classifier that separates the training data with the largest margin. More precisely, define the *(geometric) margin* of a linear classifier $h(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b)$ on an example $(\mathbf{x}_i, y_i) \in \mathbb{R}^n \times \{-1, 1\}$ as

$$\gamma_i = \frac{y_i(\mathbf{w} \cdot \mathbf{x}_i + b)}{\|\mathbf{w}\|}, \tag{3}$$

where $\|\mathbf{w}\|$ denotes the Euclidean norm of $\mathbf{w}$. (Note that the distance of $\mathbf{x}_i$ from the hyperplane $\mathbf{w} \cdot \mathbf{x} + b = 0$ is given by $\frac{|\mathbf{w} \cdot \mathbf{x}_i + b|}{\|\mathbf{w}\|}$; therefore the above margin on $(\mathbf{x}_i, y_i)$ is simply a signed version of this distance, with a positive sign if the example is classified correctly and negative otherwise.) The (geometric) margin of the classifier given by $(\mathbf{w}, b)$ on a sample $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$ is then defined as the minimal margin on examples in S:

$$\gamma = \min_{1 \le i \le m} \gamma_i \,. \tag{4}$$

Given a linearly separable training sample $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)) \in (\mathbb{R}^n \times \{-1, 1\})^m$, the hard margin SVM algorithm finds a linear classifier that maximizes the above margin on S. In particular, any linear classifier that separates S correctly will have margin $\gamma > 0$; without loss of generality, we can represent any such classifier by some $(\mathbf{w}, b)$ such that

$$\min_{1 \le i \le m} y_i(\mathbf{w} \cdot \mathbf{x}_i + b) = 1 \,. \tag{5}$$

The margin of such a classifier on S then becomes simply

$$\gamma = \min_{1 \le i \le m} \frac{y_i(\mathbf{w} \cdot \mathbf{x}_i + b)}{\|\mathbf{w}\|} = \frac{1}{\|\mathbf{w}\|} \,. \tag{6}$$

Thus, maximizing the margin becomes equivalent to minimizing the norm $\|\mathbf{w}\|$ subject to the constraints in Eq. (5), which can be written as the following optimization problem:

$$\min_{\mathbf{w}, b} \ \frac{1}{2}\|\mathbf{w}\|^2 \tag{7}$$

subject to

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \ \ge \ 1 \,, \quad i = 1, \ldots, m. \tag{8}$$

This is a convex quadratic program (QP) and can in principle be solved directly. However it is useful to consider the dual of the above problem, which allows one to easily obtain the *support vectors*, namely the points closest to the separating hyperplane, and also facilitates the extension to nonlinear classifiers. Introducing Lagrange multipliers $\alpha_i \geq 0$ $(i = 1, \ldots, m)$ for the inequality constraints above gives the Lagrangian

$$\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha}) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_{i=1}^{m} \alpha_i (y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1)\,. \tag{9}$$

We would like to maximize this w.r.t. $\boldsymbol{\alpha}$ and minimize w.r.t. $(\mathbf{w}, b)$. Setting the derivatives of $\mathcal{L}(\mathbf{w}, b, \boldsymbol{\alpha})$ w.r.t. $\mathbf{w}$ and b to zero gives:

$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x}_i \tag{10}$$

$$\sum_{i=1}^{m} \alpha_i y_i = 0\,. \tag{11}$$

This leads to the following dual problem:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j y_i y_j (\mathbf{x}_i \cdot \mathbf{x}_j) \tag{12}$$

subject to

$$\sum_{i=1}^{m} \alpha_i y_i = 0 \tag{13}$$

$$\alpha_i \geq 0\,, \quad i = 1, \ldots, m. \tag{14}$$

This is again a convex QP (in the m variables α_i) and can be solved efficiently (see for example [1, 2]). The weight vector $\mathbf{w}$ corresponding to the maximal margin classifier can be obtained from the solution $\boldsymbol{\alpha}$ to the dual problem via Eq. (10). To obtain a solution for the threshold b, one can consider the primal constraints in Eq. (8), which leads to

$$b = -\frac{1}{2}\left(\min_{i:y_i=1} \mathbf{w} \cdot \mathbf{x}_i + \max_{i:y_i=-1} \mathbf{w} \cdot \mathbf{x}_i\right)\,. \tag{15}$$

In order to classify a new point $\mathbf{x} \in \mathbb{R}^n$ using the learned classifier, one then computes

$$h(\mathbf{x}) = \text{sign}(\mathbf{w} \cdot \mathbf{x} + b) = \text{sign}\left(\sum_{i=1}^{m} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + b\right)\,. \tag{16}$$

Moreover, from the Karush-Kuhn-Tucker (KKT) conditions, it follows that the solution must satisfy

$$\alpha_i (y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1) = 0\,, \quad i = 1, \ldots, m. \tag{17}$$

This means that $\alpha_i > 0$ for only those training examples $(\mathbf{x}_i, y_i)$ that are closest to the separating hyperplane; the corresponding points $\mathbf{x}_i$ are called the **support vectors**. Thus when computing the prediction on a new point $\mathbf{x}$ using Eq. (16), one only needs to compute the dot products of $\mathbf{x}$ with the support vectors; all other terms in the sum will vanish.

3 Non-Linearly Separable Data – Soft Margin SVM

The above derivation assumed the existence of a linear classifier that can correctly classify all examples in a given training sample $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m))$. But what if the sample is not linearly separable?

In this case, one needs to allow for the possibility of errors in classification. This is usually done by relaxing the constraints in Eq. (8) through the introduction of slack variables $\xi_i \geq 0$ $(i = 1, \ldots, m)$, and requiring only that

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i\,, \quad i = 1, \ldots, m. \tag{18}$$

An extra cost for errors can be assigned as follows:

$$\min_{\mathbf{w},b,\boldsymbol{\xi}} \ \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{m}\xi_i \tag{19}$$

subject to

$$y_i(\mathbf{w}\cdot\mathbf{x}_i + b) \ \geq \ 1-\xi_i\,, \quad i=1,\ldots,m \tag{20}$$

$$\xi_i \ \geq \ 0\,, \quad i=1,\ldots,m. \tag{21}$$

Thus, whenever $y_i(\mathbf{w}\cdot\mathbf{x}_i+b) < 1$, we pay an associated cost of $C\xi_i = C(1-y_i(\mathbf{w}\cdot\mathbf{x}_i+b))$ in the objective function; a classification error occurs when $y_i(\mathbf{w}\cdot\mathbf{x}_i+b)\leq 0$, or equivalently when $\xi_i \geq 1$. The parameter $C>0$ controls the tradeoff between increasing the margin (minimizing $\|\mathbf{w}\|$) and reducing the errors (minimizing ξ_i): a large value of C keeps the errors small at the cost of a reduced margin; a small value of C allows for more errors while increasing the margin on the remaining examples. Forming the dual of the above problem as before leads to the same convex QP as in the linearly separable case, except that the constraints in Eq. (29) are replaced by

$$0 \leq \alpha_i \leq C \quad i=1,\ldots,m. \tag{22}$$

The solution is obtained similarly to the linearly separable case; in this case, there are three types of support vectors with $\alpha_i > 0$ (see Figure 2):

1. Margin support vectors ($\xi_i = 0$; these lie on the margin and are correctly classified)
2. Non-margin support vectors with $0 < \xi_i < 1$ (these are correctly classified, but lie within the margin)
3. Non-margin support vectors with $\xi_i \geq 1$ (these correspond to classification errors)

The above formulation of the SVM algorithm for the general (nonseparable) case is often called the *soft margin* SVM.

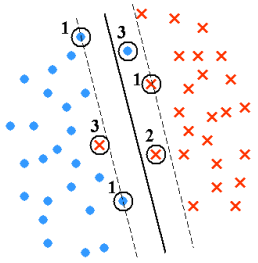

Figure 2: Three types of support vectors in the non-separable case.

4 Hinge Loss Interpretation

An alternative motivation for the (soft margin) SVM algorithm that will be useful later in the course is in terms of minimizing the hinge loss. In particular, recall that in the probabilistic setting, the training examples in $S=((\mathbf{x}_1,y_1),\ldots,(\mathbf{x}_m,y_m))$ are assumed to be drawn randomly and independently according to some fixed but unknown distribution D on $\mathcal{X}\times\{-1,1\}$, and the goal is to a learn a classifier $h:\mathcal{X}\to\{-1,1\}$ that minimizes the generalization error $\mathrm{er}_D^{0\text{-}1}[h]$. In the absence of knowledge about D, one can consider minimizing the *empirical error*

$$\mathrm{er}_S^{0\text{-}1}[h] = \frac{1}{m}\sum_{i=1}^{m}\ell_{0\text{-}1}(y_i,h(\mathbf{x}_i)) = \frac{1}{m}\sum_{i=1}^{m}\mathbf{1}(h(\mathbf{x}_i)\neq y_i)\,. \tag{23}$$

However, for most interesting function classes (including the class of linear threshold functions), minimizing the empirical zero-one error over the function class is an NP-hard problem. Consequently, many learning algorithms for binary classification are designed to minimize instead a convex upper bound on the zero-one error, which makes the problem tractable. In particular, for real-valued predictions $\hat{p}\in\mathbb{R}$ (to be used in classification via $\mathrm{sign}(\hat{p})$), consider the following *hinge loss* $\ell_{\mathrm{hinge}}:\{-1,1\}\times\mathbb{R}\to[0,\infty)$:

$$\ell_{\mathrm{hinge}}(y,\hat{p}) = (1-y\hat{p})_+\,, \tag{24}$$

where $z_+ = \max(0,z)$. This loss is convex in $\hat{p}$; moreover, it is easy to see that

$$(1-y\hat{p})_+ \geq \mathbf{1}(\mathrm{sign}(\hat{p})\neq y)\,. \tag{25}$$

Now consider learning a linear classifier that minimizes the empirical hinge loss, plus a regularization term:

$$\min_{\mathbf{w},b} \frac{1}{m}\sum_{i=1}^{m}(1-y_i(\mathbf{w}\cdot\mathbf{x}_i+b))_+ + \frac{\lambda}{2}\|\mathbf{w}\|^2\,, \tag{26}$$

where $\lambda > 0$ is a regularization parameter and $\frac{1}{2}\|\mathbf{w}\|^2$ acts as a *regularizer* (we will study regularization in more detail later; briefly, the role of a regularizer is to encourage solutions of lower complexity, measured here by $\|\mathbf{w}\|$). The above optimization problem is equivalent to that in Eqs. (19-21) with $C = \frac{1}{\lambda m}$ (convince yourself that this is the case!). Thus, the SVM algorithm can be viewed as minimizing a regularized form of the empirical hinge loss.

Later in the course, we will consider questions related to whether minimizing the hinge or other losses can lead one to learn an optimal (Bayes) classifier in the limit of infinite data.

5 Nonlinear SVMs via Kernel Functions

So far we have described the basic SVM algorithm for learning a linear classifier in $\mathbb{R}^n$. However one of the main reasons for the widespread use of SVMs is their elegant and efficient application to learning nonlinear classifiers – and classifiers over spaces other than $\mathbb{R}^n$ – via the use of kernel functions.

Consider first learning a nonlinear classifier in $\mathbb{R}^n$. The basic idea is to map the given data in $\mathbb{R}^n$ to some (high-dimensional) feature space, say $\mathbb{R}^N$, using some nonlinear mapping $\boldsymbol{\psi} : \mathbb{R}^n \to \mathbb{R}^N$, and then to learn a linear classifier in the new space. Given a training sample $S = ((\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_m, y_m)) \in (\mathbb{R}^n \times \{-1, 1\})^m$, training then involves solving the following (dual) optimization problem:

$$\max_{\boldsymbol{\alpha}} \sum_{i=1}^{m}\alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j(\boldsymbol{\psi}(\mathbf{x}_i)\cdot\boldsymbol{\psi}(\mathbf{x}_j)) \tag{27}$$

subject to

$$\sum_{i=1}^{m}\alpha_i y_i = 0 \tag{28}$$

$$\alpha_i \geq 0\,, \quad i = 1, \ldots, m. \tag{29}$$

The label of a new instance $\mathbf{x} \in \mathbb{R}^n$ is predicted as $h(\mathbf{x}) = \text{sign}(\mathbf{w}\cdot\boldsymbol{\psi}(\mathbf{x}) + b)$, where $\mathbf{w} \in \mathbb{R}^N$ is the learned weight vector in $\mathbb{R}^N$, given by $\mathbf{w} = \sum_{i=1}^{m}\alpha_i y_i \boldsymbol{\psi}(\mathbf{x}_i)$. In particular, the prediction on $\mathbf{x}$ can be written as

$$h(\mathbf{x}) = \text{sign}\left(\sum_{i=1}^{m}\alpha_i y_i(\boldsymbol{\psi}(\mathbf{x}_i)\cdot\boldsymbol{\psi}(\mathbf{x})) + b\right). \tag{30}$$

The important thing to note is that in both training and prediction, the instances $\mathbf{x}_i, \mathbf{x}$ are used only via dot products between their images $\boldsymbol{\psi}(\mathbf{x}_i), \boldsymbol{\psi}(\mathbf{x})$. In general, computing these dot products directly in a high-dimensional feature space may be expensive (or even infeasible). This is one situation where kernels are useful.

A *kernel function* over $\mathcal{X}$ is a symmetric function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$. A positive semi-definite kernel (sometimes called a *Mercer kernel*), in addition, computes a dot product in some (high-dimensional) space:

$$K(\mathbf{x}, \mathbf{x}') = \boldsymbol{\psi}(\mathbf{x})\cdot\boldsymbol{\psi}(\mathbf{x}') \tag{31}$$

for some $\boldsymbol{\psi} : \mathcal{X} \to V$ (where V is some inner product space, such as $\mathbb{R}^N$ for some appropriate N, or even some infinite-dimensional space). Thus if we can find a Mercer kernel $K : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}$ that computes a dot product in the feature space $V = \mathbb{R}^N$ we are interested in, we can use the kernel evaluations $K(\mathbf{x}, \mathbf{x}')$ to replace the dot products $\boldsymbol{\psi}(\mathbf{x})\cdot\boldsymbol{\psi}(\mathbf{x}')$ in the SVM algorithm. The advantage is that in certain cases, one can find a Mercer kernel that allows one to efficiently simulate dot products in the high-dimensional feature space using computations in only the original, lower-dimensional space.

For example, consider learning a quadratic classifier in $\mathbb{R}^2$:

$$h(\mathbf{x}) = \text{sign}(w_1 x_1^2 + w_2 x_2^2 + w_3 x_1 x_2 + w_4 x_1 + w_5 x_2 + b)\,. \tag{32}$$

This is equivalent to learning a linear classifier in the feature space $\boldsymbol{\psi}(\mathbb{R}^2)$, where $\boldsymbol{\psi} : \mathbb{R}^2 \to \mathbb{R}^5$ is given by

$$\boldsymbol{\psi}\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ x_1 x_2 \\ x_1 \\ x_2 \end{bmatrix}. \tag{33}$$

Without the use of a kernel, learning such a classifier using an SVM would require computing dot products in $\mathbb{R}^5$. Now consider the kernel $K : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ given by

$$K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + 1)^2\,. \tag{34}$$

It can be verified that $K(\mathbf{x}, \mathbf{x}') = \boldsymbol{\psi}'(\mathbf{x}) \cdot \boldsymbol{\psi}'(\mathbf{x}')$, where $\boldsymbol{\psi}' : \mathbb{R}^2 \to \mathbb{R}^6$ is given by

$$\boldsymbol{\psi}'\left(\begin{bmatrix} x_1 \\ x_2 \end{bmatrix}\right) = \begin{bmatrix} x_1^2 \\ x_2^2 \\ \sqrt{2} x_1 x_2 \\ \sqrt{2} x_1 \\ \sqrt{2} x_2 \\ 1 \end{bmatrix}. \tag{35}$$

Thus, an SVM with the above kernel can be used to learn a quadratic classifier in $\mathbb{R}^2$ using only dot products in $\mathbb{R}^2$.

Other commonly used kernels in $\mathbb{R}^n$ include polynomial kernels of degree q, given by $K(\mathbf{x}, \mathbf{x}') = (\mathbf{x} \cdot \mathbf{x}' + 1)^q$, and Gaussian kernels, given by $K(\mathbf{x}, \mathbf{x}') = \exp(\frac{-\|\mathbf{x}-\mathbf{x}'\|^2}{2\sigma^2})$ for $\sigma > 0$. The Gaussian kernel corresponds to a dot product in an infinite dimensional space.

Kernels are also useful for learning in instance spaces $\mathcal{X}$ other than $\mathbb{R}^n$. In particular, note that any positive semi-definite kernel function $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ can be used directly with the SVM algorithm (by replacing dot products $(\mathbf{x} \cdot \mathbf{x}')$ in the algorithm with kernel evaluations $K(x, x')$), even if the feature space implemented by the kernel cannot be described explicitly. Since dot products $(\mathbf{x} \cdot \mathbf{x}')$ in $\mathbb{R}^n$ can be viewed as computing a 'similarity measure' between $\mathbf{x}, \mathbf{x}'$, typically, one looks for a kernel $K : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ such that $K(x, x')$ similarly computes some form of similarity between x, x' in $\mathcal{X}$. This approach has been used, for example, for learning classifiers using the SVM algorithm for instances represented as strings, trees, graphs, etc.[4] The learned classifier in this case takes the form $h(x) = \text{sign}(f(x) + b)$, where $f : \mathcal{X} \to \mathbb{R}$ is given by $f(x) = \sum_{i=1}^m \alpha_i y_i K(x_i, x)$ and belongs to the *reproducing kernel Hilbert space* (RKHS) $\mathcal{F}_K$ associated with K (don't worry if you haven't seen RKHSs before; we'll discuss their properties later as needed).

6 Next Lecture

Say we are given training examples $(x_1, y_1), \ldots, (x_m, y_m) \in (\mathcal{X} \times \{-1, 1\})$ drawn independently from some fixed but unknown distribution D, and use the SVM algorithm to learn from these examples a classifier $h(x) = \text{sign}(f(x) + b)$. What can we say about the generalization error of the learned classifier, i.e. the expected performance of the classifier on future examples drawn from D? In the next lecture, we will start to address this question using the classical technique of uniform convergence. Later, we will see other techniques for studying the generalization error. The SVM algorithm will be used as a case study throughout to compare the generalization error bounds that are obtained using different techniques.

[4] Note that the same kernel trick can be used with any algorithm that uses only dot products between instances in its formulation.

References

[1] Christopher J.C. Burges. A tutorial on support vector machines for pattern recognition. *Data Mining and Knowledge Discovery*, 2:121–167, 1998.

[2] John C. Platt. Fast training of support vector machines using sequential minimal optimization. In *Advances in Kernel Methods – Support Vector Learning*. MIT Press, 1998.