4

Support Vector Machines and Kernel Functions

In this lecture we begin the exploration of the 2-class hyperplane separation problem. We are given a training set of instances $\mathbf{x}_i \in R^n$, $i = 1, ..., m$, and class labels $y_i = \pm 1$ (i.e., the training set is made up of "positive" and "negative" examples). We wish to find a hyperplane direction $\mathbf{w} \in R^n$ and an offset scalar b such that $\mathbf{w} \cdot \mathbf{x}_i - b > 0$ for positive examples and $\mathbf{w} \cdot \mathbf{x}_i - b < 0$ for negative examples — which together means that the margins $y_i(\mathbf{w} \cdot \mathbf{x}_i - b) > 0$ are positive.

Assuming that such a hyperplane exists, clearly it is not unique. We therefore need to introduce another constraint so that we could find the most "sensible" solution among all (infinitley many) possible hyperplanes which separate the training data. Another issue is that the framework is very limited in the sense that for most real-world classification problems it is somewhat unlikely that there would exist a linear separating function to begin with. We therefore need to find a way to extend the framework to include non-linear decision boundaries at a reasonable cost. These two issues will be the focus of this lecture.

Regarding the first issue, since there is more than one separating hyperplane (assuming the training data is linearly separable) then the question we need to ask ourselves is among all those solutions which of them has the best "generalization" properties? In other words, our goal in constructing a learning machine is not necessarily to do very well (or perfect) on the training data, because the training data is merely a sample of the instance space, and not necessarily a "representative" sample — it is simply a sample. Therefore, doing well on the sample (the training data) does not necessarily guarantee (or even imply) that we will do well on the entire instance space. The goal of constructing a learning machine is to maximize the performance on the test data (the instances we haven't seen), which in turn means that

we wish to generalize "good" classification performance on the training set onto the entire instance space.

A related issue to generalization is that the distribution used to generate the training data is unknown. Unlike the statistical inference material we had so far, this time we will not attempt to estimate the distribution. The reason one can derive optimal learning algorithms yet bypass the need for estimating distributions would be explained later in the course when PAC-learning will be introduced. For now we will focus only on the algorithmic aspect of the learning problem.

The idea is to consider a subset C_γ of all hyperplanes which have a fixed margin γ where the margin is defined as the distance of the closest training point to the hyperplane:

$$\gamma = \min_i \left\{ \frac{y_i(\mathbf{w}^\top \mathbf{x}_i - b)}{\|\mathbf{w}\|} \right\}.$$

The Support Vector Machine (SVM), first introduced by Vapnik and his colleagues in 1992, seeks a separating hyperplane which simultaneously minimizes the empirical error *and* maximizes the margin. The idea of maximizing the margin is intuitively appealing because a decision boundary which lies close to some of the training instances is less likely to generalize well because the learning machine will be susceptible to small perturbations of those instance vectors. A formal motivation for this approach is deferred to the PAC-learning material we will introduce later in the course.

4.1 Large Margin Classifier as a Quadratic Linear Programming

We would first like to set up the linear separating hyperplane as an optimization problem which is both consistent with the training data and maximizes the margin induce by the separating hyperplane over all possible consistent hyperplanes.

Formally speaking, the distance between a point $\mathbf{x}$ and the hyperplane is defined by

$$\frac{\mid \mathbf{w} \cdot \mathbf{x} - b \mid}{\sqrt{\mathbf{w} \cdot \mathbf{w}}}.$$

Since we are allowed to scale the parameters $\mathbf{w}, b$ at will (note that if $\mathbf{w} \cdot \mathbf{x} - b > 0$ so is $(\lambda\mathbf{w}) \cdot \mathbf{x} - (\lambda b) > 0$ for all $\lambda > 0$) we can set the distance between the boundary points to the hyperplane to be $1/\sqrt{\mathbf{w} \cdot \mathbf{w}}$ by scaling $\mathbf{w}, b$ such the point(s) with smallest margin (closest to the hyperplane) will be normalized: $\mid \mathbf{w} \cdot \mathbf{x} - b \mid = 1$, therefore the margin is simply $2/\sqrt{\mathbf{w} \cdot \mathbf{w}}$ (see Fig. 5.1). Note that $\operatorname{argmax}_{\mathbf{w}} 2/\sqrt{\mathbf{w} \cdot \mathbf{w}}$ is equivalent to $\operatorname{argmax}_{\mathbf{w}} 2/(\mathbf{w} \cdot \mathbf{w})$

which in turn is equivalent to $\text{argmin}_{\mathbf{w}} \frac{1}{2}\mathbf{w} \cdot \mathbf{w}$. Since all positive points and negative points should be farther away from the boundary points we also have the separability constraints $\mathbf{w} \cdot \mathbf{x} - b \geq 1$ when $\mathbf{x}$ is a positive instance and $\mathbf{w} \cdot \mathbf{x} - b \leq -1$ when $\mathbf{x}$ is a negative instance. Both separability constraints can be combined: $y(\mathbf{w} \cdot \mathbf{x} - b) \geq 1$. Taken together, we have defined the following optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\mathbf{w} \cdot \mathbf{w} && (4.1) \\ & \textit{subject to} \\ & y_i(\mathbf{w} \cdot \mathbf{x}_i - b) - 1 \geq 0 \qquad i = 1, ..., m && (4.2) \end{aligned}$$

This type of optimization problem has a quadratic criteria function and linear inequalities and is known in the literature as a *Quadratic Linear Programming* (QP) type of problem.

This particular QP, however, requires that the training data are linearly separable — a condition which may be unrealistic. We can relax this condition by introducing the concept of a "soft margin" in which the separability holds approximately with some error:

$$\begin{aligned} \min_{\mathbf{w},b,\epsilon_i} \quad & \frac{1}{2}\mathbf{w} \cdot \mathbf{w} + \nu \sum_{i=1}^{l} \epsilon_i && (4.3) \\ & \textit{subject to} \\ & y_i(\mathbf{w} \cdot \mathbf{x}_i - b) \geq 1 - \epsilon_i \qquad i = 1, ..., m \\ & \epsilon_i \geq 0 \end{aligned}$$

Where ν is some pre-defined weighting factor. The (non-negative) variables ϵ_i allow data points to be *miss-classified* thereby creating an approximate separation. Specifically, if $\mathbf{x}_i$ is a positive instance ($y_i = 1$) then the "soft" constraint becomes:

$$\mathbf{w} \cdot \mathbf{x}_i - b \geq 1 - \epsilon_i,$$

where if $\epsilon_i = 0$ we are back to the original constraint where $\mathbf{x}_i$ is either a boundary point or laying further away in the half space assigned to positive instances. When $\epsilon_i > 0$ the point $\mathbf{x}_i$ can reside inside the margin or even in the half space assigned to negative instances. Likewise, if $\mathbf{x}_i$ is a negative instance ($y_i = -1$) then the soft constraint becomes:

$$\mathbf{w} \cdot \mathbf{x}_i - b \leq -1 + \epsilon_i.$$

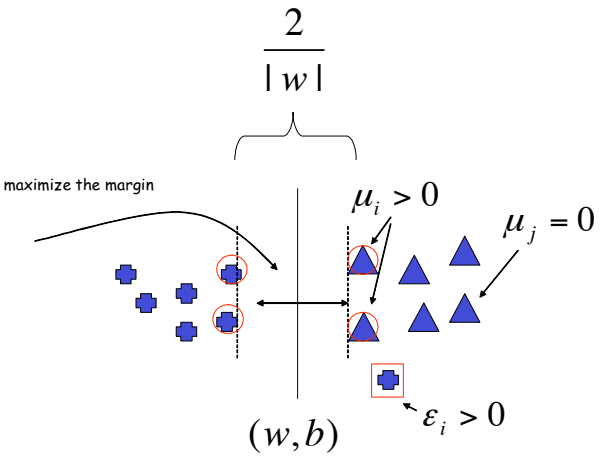

Fig. 4.1. Separating hyperplane $\mathbf{w}, b$ with maximal margin. The boundary points are associated with non-vanishing Lagrange multipliers $\mu_i > 0$ and margin errors are associated with $\epsilon_i > 0$ where the criteria function encourages a small number of margin errors.

The criterion function penalizes (the L_1-norm) for non-vanishing ϵ_i thus the overall system will seek a solution with few as possible "margin errors" (see Fig. 5.1). Typically, when possible, an L_1 norm is preferable as the L_2 norm overly weighs high magnitude outliers which in some cases can dominate the energy function. Another note to make here is that strictly speaking the "right thing" to do is to penalize the margin errors based on the L_0 norm $\|\epsilon\|_0^0 = |\{i : \epsilon_i > 0\}|$, i.e., the number of non-zero entries, and drop the balancing parameter ν. This is because it does not matter how far away a point is from the hyperplane — all what matters is whether a point is classified correctly or not (see the definition of empirical error in Lecture 4). The problem with that is that the optimization problem would no longer be *convex* and non-convex problems are notoriously difficult to solve. Moreover, the class of convex optimization problems (as the one described in Eqn. 4.3) can be solved in polynomial time complexity.

So far we have described the problem formulation which *when solved* would provide a solution with "sensible" generalization properties. Although we can proceed using an off-the-shelf QLP solver, we will first pursue the "dual" problem. The dual form will highlight some key properties of the approach and will enable us to extend the framework to handle non-linear

decision surfaces at a very little cost. In the appendix we take a brief tour on the basic principles associated with constrained optimization, the Karush-Kuhn-Tucker (KKT) theorem and the dual form. Those are recommended to read before moving to the next section.

4.2 The Support Vector Machine

We return now to the primal problem (eqn. 6.3) representing the maximal margin separating hyperplane with margin errors:

$$\begin{aligned} \min_{\mathbf{w},b,\epsilon_i} \quad & \frac{1}{2}\mathbf{w}\cdot\mathbf{w} + \nu\sum_{i=1}^{l}\epsilon_i \\ & \textit{subject to} \\ & y_i(\mathbf{w}\cdot\mathbf{x}_i - b) \geq 1-\epsilon_i \qquad i = 1,...,m \\ & \epsilon_i \geq 0 \end{aligned}$$

We will now derive the Lagrangian Dual of this problem. By doing so a new key property will emerge facilitated by the fact that the criteria function $\theta(\mu)$ (note there are no equality constraints thus there is no need for λ) involves only inner-products of the training instance vectors $\mathbf{x}_i$. This property will form the key of mapping the original input space of dimension n to a higher dimensional space thereby allowing for non-linear decision surfaces for separating the training data.

Note that with this particular problem the strong duality conditions are satisfied because the criteria function and the inequality constraints form a convex set. The Lagrangian takes the following form:

$$L(\mathbf{w},b,\epsilon_i,\mu) = \frac{1}{2}\mathbf{w}\cdot\mathbf{w} + \nu\sum_{i=1}^{m}\epsilon_i - \sum_{i=1}^{m}\mu_i\left[y_i(\mathbf{w}\cdot\mathbf{x}_i - b) - 1 + \epsilon_i\right] - \sum_{i=1}^{m}\delta_i\epsilon_i$$

Recall that

$$\theta(\mu) = \min_{\mathbf{W},b,\boldsymbol{\epsilon}} L(\mathbf{w},b,\boldsymbol{\epsilon},\boldsymbol{\mu},\boldsymbol{\delta}).$$

Since the minimum is obtained at the vanishing partial derivatives of the Lagrangian with respect to $\mathbf{w}, b$, the next step would be to evaluate those

constraints and substitute them back into $L()$ to obtain $\theta(\mu)$:

$$\frac{\partial L}{\partial \mathbf{w}} = \mathbf{w} - \sum_i \mu_i y_i \mathbf{x}_i = 0 \tag{4.4}$$

$$\frac{\partial L}{\partial b} = \sum_i \mu_i y_i = 0 \tag{4.5}$$

$$\frac{\partial L}{\partial \epsilon_i} = \nu - \mu_i - \delta_i = 0 \tag{4.6}$$

From the first constraint (4.4) we obtain $\mathbf{w} = \sum_i \mu_i y_i \mathbf{x}_i$, that is, $\mathbf{w}$ is described by a linear combination of a *subset* of the training instances. The reason that not all instances participate in the linear superposition is due to the KKT conditions: $\mu_i = 0$ when $y_i(\mathbf{w} \cdot \mathbf{x}_i - b) > 1$, i.e., the instance $\mathbf{x}_i$ is classified correctly and is not a boundary point, and conversely, $\mu_i > 0$ when $y_i(\mathbf{w} \cdot \mathbf{x}_i - b) = 1 - \epsilon_i$, i.e., when $\mathbf{x}_i$ is a boundary point or when $\mathbf{x}_i$ is a margin error ($\epsilon_i > 0$) — note that for a margin error instance the value of ϵ_i would be the smallest possible required to reach an equality in the constraint because the criteria function penalizes large values of ϵ_i. The boundary points (and the margin errors) are called *support vectors* thus $\mathbf{w}$ is defined by the support vectors *only*. The third constraint (4.6) is equivalent to the constraint:

$$0 \leq \mu_i \leq \nu \qquad i = 1, ..., l,$$

since $\delta_i \geq 0$. Also note that if $\epsilon_i > 0$, i.e., point $\mathbf{x}_i$ is a margin-error point, then by KKT conditions we must have $\delta_i = 0$. As a result $\mu_i = \nu$. Therefore based on the values of μ_i alone we can make the following classifications:

- $0 < \mu_i < \nu$: point $\mathbf{x}_i$ is on the margin and is not a margin-error.
- $\mu_i = \nu$: points $\mathbf{x}_i$ is a margin-error point.
- $\mu_i = 0$: point $\mathbf{x}_i$ is not on the margin.

Substituting these results/constraints back into the Lagrangian $L()$ we obtain the *dual problem*:

$$\begin{aligned} \max_{\mu_1,...,\mu_m} \quad & \theta(\boldsymbol{\mu}) = \sum_{i=1}^{m} \mu_i - \frac{1}{2} \sum_{i,j} \mu_i \mu_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j \\ & \textit{subject to} \\ & 0 \leq \mu_i \leq \nu \qquad i = 1, ..., m \\ & \sum_{i=1}^{m} y_i \mu_i = 0 \end{aligned} \tag{4.7}$$

The criterion function $\theta(\boldsymbol{\mu})$ can be written in a more compact manner as

follows: Let M be a $l \times l$ matrix whose entries are $M_{ij} = y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j$ then $\theta(\boldsymbol{\mu}) = \boldsymbol{\mu}^\top \mathbf{1} - \frac{1}{2}\boldsymbol{\mu}^\top M \boldsymbol{\mu}$ where $\mathbf{1}$ is the vector of $(1, ..., 1)$ and $\boldsymbol{\mu}$ is the vector $(\mu_1, ..., \mu_m)$ and $\mu^\top$ is the transpose (row vector). Note that M is *positive definite*, i.e., $\mathbf{x}^\top M \mathbf{x} > 0$ for all vectors $\mathbf{x} \neq 0$ — a property which will be important later.

The key feature of the dual problem is not so much that it is simpler than the primal (in fact it isn't since the primal has no equality constraints) or that it has a more "elegant" feel, the key feature is that the problem is completely described by the inner products of the training instances $\mathbf{x}_i$, $i = 1, ..., m$. This fact will be shown to be a crucial ingredient in the so called "kernel trick" for the computation of inner-products in high dimensional spaces using simple functions defined on pairs of training instances.

4.3 The Kernel Trick

We ended with the dual formulation of the SVM problem and noticed that the input data vectors $\mathbf{x}_i$ are represented by the Gram matrix M. In other words, only inner-products of the input vectors play a role in the dual formulation — there is no explicit use of $\mathbf{x}_i$ or any other function of $\mathbf{x}_i$ besides inner-products. This observation suggests the use of what is known as the "kernel trick" to replace the inner-products by non-linear functions.

The common principle of kernel methods is to construct nonlinear variants of linear algorithms by substituting inner-products by nonlinear kernel functions. Under certain conditions this process can be interpreted as mapping of the original measurement vectors (so called "input space") onto some higher dimensional space (possibly infinitely high) commonly referred to as the "feature space". Mathematically, the kernel approach is defined as follows: let $\mathbf{x}_1, ..., \mathbf{x}_l$ be vectors in the input space, say R^n, and consider a mapping $\phi(\mathbf{x}) : R^n \to \mathcal{F}$ where $\mathcal{F}$ is an inner-product space. The kernel-trick is to calculate the inner-product in $\mathcal{F}$ using a kernel function $k : R^n \times R^n \to R$, $k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j)$, while avoiding explicit mappings (evaluation of) $\phi()$.

Common choices of kernel selection include the d'th order polynomial kernels $k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + \theta)^d$ and the Gaussian RBF kernels $k(\mathbf{x}_i, \mathbf{x}_j) = \exp(-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2)$. If an algorithm can be restated such that the input vectors appear in terms of inner-products only, one can substitute the inner-products by such a kernel function. The resulting kernel algorithm can be interpreted as running the original algorithm on the space $\mathcal{F}$ of mapped objects $\phi(\mathbf{x})$.

We know that M of the dual form is positive semi-definite because M

can be written is $M = Q^\top Q$ where $Q = [y_1\mathbf{x}_1, ..., y_l\mathbf{x}_l]$. Therefore $\mathbf{x}^\top M\mathbf{x} = \|Q\mathbf{x}\|^2 \geq 0$ for all choices of $\mathbf{x}$ (which means that the eigenvalues of M are non-negative). If the entries of M are to be replaced with $y_i y_j k(\mathbf{x}_i, \mathbf{x}_j)$ then the condition we must enforce on the function $k()$ is that it is a *positive definite kernel* function. A positive definite function is defined such that for any set of vectors $\mathbf{x}_1, ..., \mathbf{x}_q$ and for any values of q the matrix K whose entries are $K_{ij} = k(\mathbf{x}_i, \mathbf{x}_j)$ is positive semi-definite. Formally, the conditions for admissible kernels $k()$ are known as Mercer's conditions summarized below:

Theorem 4 (Mercer's Conditions) *Let $k(x, y)$ be symmetric and continuous. The following conditions are equivalent:*

(i) *$k(x, y) = \sum_{i=1}^{\infty} \alpha_i \phi_i(x)\phi_i(y) = \phi(x)^\top \phi(y)$ for any uniformly converging series $\alpha_i > 0$.*

(ii) *for all $\psi()$ satisfying $\int_x \psi^2(x)dx < \infty$, then*

$$\int_x \int_y k(x, y)\psi(x)\psi(y)dxdy \geq 0$$

(iii) *for all $\{\mathbf{x}_i\}_{i=1}^q$ and for all q, the matrix $K_{ij} = k(x_i, x_j)$ is positive semi-definite.*

Perhaps the non-obvious condition is No. 1 which allows for the feature map $\phi()$ to have infinitely many coordinates (a vector in Hilbert space). For example, as we shall see below, the kernel $\exp(-\frac{1}{2\sigma^2}\|\mathbf{x}_i - \mathbf{x}_j\|^2)$ is an inner-product of two vectors with infinitely many coordinates. We will consider next a number of popular kernels.

4.3.1 The Homogeneous Polynomial Kernel

Let $\mathbf{x}, \mathbf{y} \in R^k$ and define $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top \mathbf{y})^d$ where $d > 0$ is a natural number. Then, the corresponding feature map $\phi(\mathbf{x})$ has $\binom{k+d-1}{d} = O(k^d)$ coordinates which take the value:

$$\phi(\mathbf{x}) = \left(\sqrt{\binom{d}{n_1, ..., n_k}} x_1^{n_1} \cdots x_k^{n_k} \right)_{n_i \geq 0, \sum_i n_i = d}$$

where $\binom{d}{n_1,...,n_k} = d!/(n_1! \cdots n_k!)$ is the multinomial coefficient (number of ways to distribute d balls into k bins where the j'th bin hold exactly $n_j \geq 0$ balls):

$$(x_1 + ... + x_k)^d = \sum_{n_i \geq 0, \sum_i n_i = d} \binom{d}{n_1, ..., n_k} x_1^{n_1} \cdots x_k^{n_k}.$$

The dimension of the vector space $\phi(\mathbf{x})$ where $\mathbf{x} \in R^k$ can be measured using the following combinatorial problem: how many arrangements of $k-1$ partitions to be placed among d items? the answer is $\binom{k+d-1}{k-1} = \binom{k+d-1}{d} = O(k^d)$. For example, $k = d = 2$ gives us :

$$(\mathbf{x}^\top \mathbf{y})^2 = x_1^2 y_1^2 + 2x_1 x_2 y_1 y_2 + x_2^2 y_2^2 = \phi(\mathbf{x})^\top \phi(\mathbf{y}),$$

where $\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1 x_2)$.

4.3.2 The non-homogeneous Polynomial Kernel

The feature map $\phi(\mathbf{x})$ contains all monomials whose power is lesser or equal to d, i.e., $\sum_i n_i \leq d$. This can be acheived by increasing the dimension to $k+1$ where n_{k+1} is used to fill the gap between $\sum_{i=1}^{k} n_i < d$ and d. Therefore the dimension of $\phi(\mathbf{x})$ where $\mathbf{x} \in R^k$ would be $\binom{k+d}{d}$. We have:

$$\begin{aligned}(\mathbf{x}^\top \mathbf{y} + \theta)^d &= (x_1 y_1 + ... + x_k y_k + \sqrt{\theta}\sqrt{\theta})^d \\ &= \sum_{n_i \geq 0, \sum_{i=1}^{k+1} n_i = d} \binom{d}{n_1, ..., n_{k+1}} x_1^{n_1} y_1^{n_1} \cdots x_1^{n_k} y_1^{n_k} \cdot \theta^{n_{k+1}/2} \theta^{n_{k+1}/2}\end{aligned}$$

Therefore, the entries of the vector $\phi(\mathbf{x})$ take the values:

$$\phi(\mathbf{x}) = \left(\sqrt{\binom{d}{n_1, ..., n_{k+1}}} x_1^{n_1} \cdots x_k^{n_k} \cdot \theta^{n_{k+1}/2} \right)_{n_i \geq 0, \sum_{i=1}^{k+1} n_i = d}$$

For example, $k = d = 2$ gives us :

$$(\mathbf{x}^\top \mathbf{y} + \theta)^2 = x_1^2 y_1^2 + 2x_1 x_2 y_1 y_2 + x_2^2 y_2^2 + 2\theta x_1 y_1 + 2\theta x_2 y_2 + \theta = \phi(\mathbf{x})^\top \phi(\mathbf{y}),$$

where $\phi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1 x_2, \sqrt{2\theta}x_1, \sqrt{2\theta}x_2, \sqrt{\theta})$. In this example, $\phi()$ is a mapping from R^2 to R^6 and hyperplanes $\phi(\mathbf{w})^\top \phi(\mathbf{x}) - b = 0$ in R^6 correspond to *conics* in R^2:

$$(w_1^2)x_1^2 + (w_2^2)x_2 + (2w_1 w_2)x_1 x_2 + (2\theta w_1)x_1 + (2\theta w_2)x_2 + (\theta - b) = 0$$

Assume we would like to find a separating *conic* (Parabola, Hyperbola, Ellipse) function rather than a line in R^2. The discussion so far suggests we construct the Gram matrix M in the dual form with the $d = 2$ polynomial kernel $k(\mathbf{x}, \mathbf{y}) = (\mathbf{x}^\top \mathbf{y} + \theta)^2$ for some parameter θ of our choosing. The extra effort we will need to invest is negligible — simply replace every occurrence $\mathbf{x}_i^\top \mathbf{x}_j$ with $(\mathbf{x}_i^\top \mathbf{x}_j + \theta)^2$.

4.3.3 The RBF Kernel

The function $k(\mathbf{x}, \mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^2/2\sigma^2}$ known as a *Radial Basis Function* (RBF) is a kernel function but with an infinite expansion. Without loss of generality let $\sigma = 1$, then we have:

$$\begin{aligned}
e^{-\|\mathbf{x}-\mathbf{y}\|^2/2} &= e^{-\|\mathbf{x}\|^2/2} e^{-\|\mathbf{y}\|^2/2} e^{\mathbf{x}^\top \mathbf{y}} \\
&= \sum_{j=0}^{\infty} \frac{(\mathbf{x}^\top \mathbf{y})^j}{j!} e^{-\|\mathbf{x}\|^2/2} e^{-\|\mathbf{y}\|^2/2} \\
&= \sum_{j=0}^{\infty} \left(\frac{e^{-\frac{\|\mathbf{x}\|^2}{2j}}}{\sqrt{j!}^{1/j}} \frac{e^{-\frac{\|\mathbf{y}\|^2}{2j}}}{\sqrt{j!}^{1/j}} \mathbf{x}^\top \mathbf{y} \right)^j \\
&= \sum_{j=0}^{\infty} \sum_{\sum_i n_i = j} \frac{e^{-\frac{\|\mathbf{x}\|^2}{2j}}}{\sqrt{j!}^{1/j}} \binom{j}{n_1, ..., n_k}^{1/2} x_1^{n_1} \cdots x_k^{n_k} \frac{e^{-\frac{\|\mathbf{y}\|^2}{2j}}}{\sqrt{j!}^{1/j}} \binom{j}{n_1, ..., n_k}^{1/2} y_1^{n_1} \cdots y_k^{n_k}
\end{aligned}$$

From which we can see that the entries of the feature map $\phi(\mathbf{x})$ are:

$$\phi(\mathbf{x}) = \left(\frac{e^{-\frac{\|\mathbf{x}\|^2}{2j}}}{\sqrt{j!}^{1/j}} \binom{j}{n_1, ..., n_k}^{1/2} x_1^{n_1} \cdots x_k^{n_k} \right)_{j=0,..,\infty,\sum_{i=1}^k n_i = j}$$

4.3.4 Classifying New Instances

By adopting some kernel $k()$ we are in fact mapping $\mathbf{x} \rightarrow \phi(\mathbf{x})$, thus we then proceed to solve for $\phi(\mathbf{w})$ and b using some QLP solver. The QLP solution of the dual form will yield the solution for the Lagrange multipliers $\mu_1, ..., \mu_m$. We saw from eqn. (4.4) that we can express $\phi(\mathbf{w})$ as a function of the (mapped) examples:

$$\phi(\mathbf{w}) = \sum_i \mu_i y_i \phi(\mathbf{x}_i).$$

Rather than explicitly representing $\phi(\mathbf{w})$ — a task which may be prohibitly expensive since in general the dimension of the feature space of a polynomial mapping is $\binom{k+d}{d}$ — we store all the support vectors (those input vectors with corresponding $\mu_i > 0$) and use them for the evaluation of test examples:

$$\begin{aligned}
f(\mathbf{x}) &= sign(\phi(\mathbf{w})^\top \phi(\mathbf{x}) - b) = sign(\sum_i \mu_i y_i \phi(\mathbf{x}_i)^\top \phi(\mathbf{x}) - b) \\
&= sign(\sum_i \mu_i y_i k(\mathbf{x}_i, \mathbf{x}) - b).
\end{aligned}$$

We see that the kernel trick enabled us to look for a non-linear separating surface by making an implicit mapping of the input space onto a higher dimensional feature space using the same dual form of the SVM formulation — the only change required was in the way the Gram matrix was constructed. The price paid for this convenience is to carry *all* the support vectors at the time of classification $f(\mathbf{x})$.

A couple of notes may be worthwhile at this point. The constant b can be recovered from any of the support vectors. Say, $\mathbf{x}^+$ is a positive support vector (but not a margin error, i.e., $\mu_i < \nu$). Then $\phi(\mathbf{w})^\top \phi(\mathbf{x}^+) - b = 1$ from which b can be recovered. The second note is that the number of support vectors is typically around 10% of the number of training examples (empirically). Thus the computational load during evaluation of $f(\mathbf{x})$ may be relatively high. Approximations have been proposed in the literature by looking for a reduced number of support vectors (not necessarily aligned with the training set) — but this is beyond the scope of this course.

The kernel trick gained its popularity with the introduction of the SVM but since then has taken a life of its own and has been applied to principal component analysis (PCA), ridge regression, canonical correlation analysis (CCA), QR factorization and the list goes on. We will meet again with the kernel trick later on.