6.867: Lectures 7, 8: SVMs and Kernels

Contents

These lecture notes have not been subject to careful scrutiny usually reserved for research papers.

1 Introduction

In previous lectures we have seen logistic regression, as well as an approximate Bayesian version of it. We've considered both generative and discriminative models. In particular, for classification we saw models where we ultimately ended up learning a hyperplane that approximately separates the date into positive and negative instances. In this lecture we will follow a geometric viewpoint, and beyond the more detailed motivation that we provide for it, one reason to consider such a viewpoint is the opinion that:

If in the end we learn a hyperplane why not try to learn it directly?

Below we study Support Vector Machines (SVMs), very widely used ML models that provide one geometrically motivated solution to learning hyperplanes (in feature space).

2 Learning hyperplanes

To develop intuition, consider the data shown in Figure 1. It shows two views of the same data, and we ask which hyperplane would you choose / prefer, and why? Most people

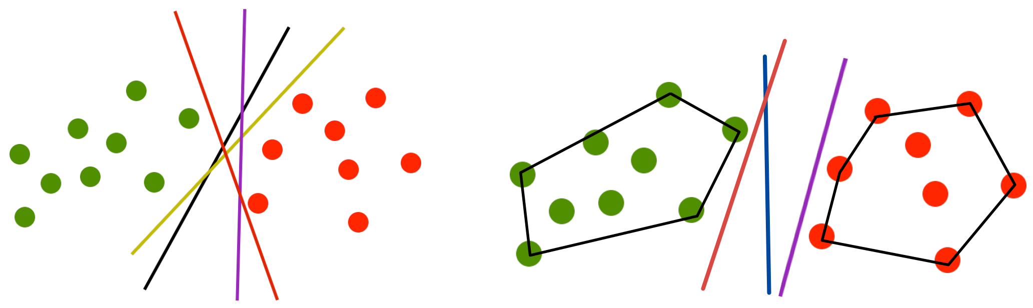
Infinite number of hyperplanes
Which one would you prefer?
The convex hull view
Which one would you prefer?

Figure 1: Hyperplanes

choose to pick hyperplanes that separate the data as "widely" as possible, guided by the intuition that this leads to the sharpest separation and perhaps the least chance of confusing positive points with negative points on new data. In the rest of this section, we make this viewpoint mathematically more precise.

2.1 Separating hyperplanes

Recall that a hyperplane is a set of the form

$$\mathcal{H} := \left\{ x \in \mathbb{R}^D \mid w^T x + b = 0 \right\}, \tag{1}$$

where w denotes a vector normal to the hyperplane and b the offset, which is called *bias*. If we scale w and b by a constant δ then the hyperplane remains unchanged. While trying to fit a hyperplane to training data, we thus make an *arbitrary* scaling choice and consider *canonical hyperplanes*, such that for our training data $\left\{x^{(1)}, \dots, x^{(n)}\right\}$ we have

$$\min_{1 \leqslant i \leqslant n} \quad |w^T x^{(i)} + b| = 1. \tag{2}$$

With this normalization one sees that the point closest to the hyperplane $\mathcal{H}$ is at a distance of $1/\|w\|$. To see how we obtained this value, let us consider the following simple geometric question.
Question: What is the distance of a point x from $\mathcal{H}$?
Recall from Homework0 that if $\bar{x}$ is the projection of a point x onto a hyperplane, then we can decompose x as

$$x = \bar{x} + t\frac{w}{\|w\|}, \tag{3}$$

that is, into a component on the hyperplane and a component normal to the hyperplane. In (3), t denotes the signed distance to the hyperplane, i.e., $|t|$ is the distance of x to the hyperplane. Since $\bar{x} \in \mathcal{H}$, it follows that $w^T\bar{x} + b = 0$. Thus, from (3) we obtain

$$w^Tx + b - t\frac{w^Tw}{\|w\|} = w^T\bar{x} + b = 0,$$

which yields

$$t = \frac{w^Tx + b}{\|w\|},$$

so that the distance of a point x to $\mathcal{H}$ is $|w^Tx + b|/\|w\|$.

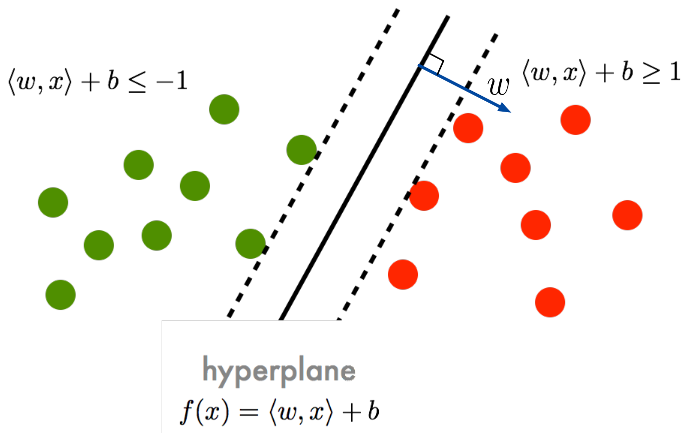
$\langle w, x\rangle + b \leq -1$
w
$\langle w, x\rangle + b \geq 1$
hyperplane
$f(x) = \langle w, x\rangle + b$

Figure 2: Separating hyperplane with large margin. The normal direction is given by $w/\|w\|$ and the red points lie on the positive side of the hyperplane.

Closely related to the signed distance t above is a quantity called the *geometric margin*:

$$yt = y\frac{w^Tx + b}{\|w\|}, \tag{4}$$

where y is the class label ($+1$ or -1) of the point x. As can be seen from Fig. 2, $w^Tx + b > 0$ for the positive points and $w^Tx + b < 0$ for the negative ones. Thus, the geometric margin $yt > 0$ (for points not on $\mathcal{H}$). Since on training data we impose the normalization (2), we see that the smallest geometric margin over training data is $1/\|w\|$. We will mostly omit the word "geometric" and just call it the margin.

These observations can be summarized as follows:

- For a correctly classified point the margin is simply the distance to the hyperplane
- Multiplying by y ensures that the margin is always positive whenever its corresponding point x is correctly classified.

- The canonical condition (2) implies a margin $1/\|w\|$
- We wish to make the margin **as large as possible** (this corresponds to the intuition expressed for selecting hyperplanes in Fig. 1).
- The classifier that we learn is: $h(x) = \mathrm{sgn}(w^T x + b)$.

This idea is central to learning hyperplane based classifiers, namely, the desire to seek a "large margin classifier".

2.2 Further intuition

We provide some additional intuition, and a semiformal justification to the reason why we prefer large margins.

Specifically, assuming that the training and test data points have the same underlying distribution. Then, it is reasonable to assume that most of the test points (except for some outliers) might lie close to at least one of the training points.

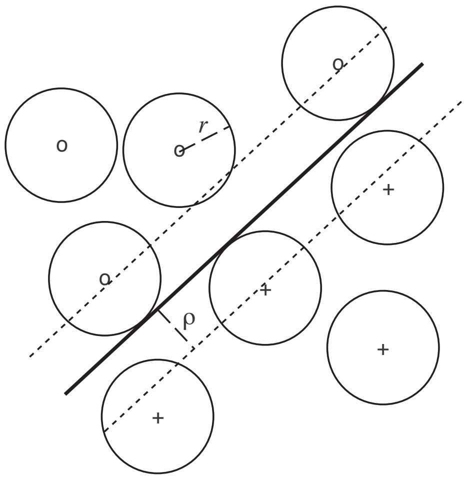

Figure 3: Robustness of margin

Suppose (for the sake of illustration) that the test data points are generated by adding bounded noise to the training data. Thus, $(x, y) \to (x + \delta x, y)$, where $\|\delta x\| \leqslant r$ for some $r > 0$. Fig. 3 illustrates that if we manage to attain a margin of $\rho > r$ on training data, then we will correctly classify **all** the test data points, since the training data are at a distance $\geqslant \rho$ to the separating hyperplane.

This idea is formalized and studied in much greater depth in statistical learning theory, where ultimately a theorem of the following form can be proved: The probability that a test data point is misclassified is bounded from above by

$$\text{margin error} + \mathcal{O}\left(\frac{1}{\text{margin}}\right),$$

where margin error corresponds to the fraction of training examples with margin smaller than $1/\|w\|$. In other words, we have to make a bias-variance tradeoff. We can keep the margin error small by shrinking the margin (overfitting), but that then drives up the $\mathcal{O}(\cdot)$ term; while making the margin large shrinks the second term but increases the chance of margin error (underfitting to training data). Such bias-variance tradeoffs are a recurrent theme in machine learning, and they should be explored when trying to improve or refine an existing model.

Bottomline: Keep margin error (training error) as small as possible while making the margin as large as possible.

The next section casts this desideratum into an optimization problem.

This desideratum arose from statistical learning theory, and is one of the great successes of theory of ML guiding the practice.

2.3 Optimizing the margin

Based on our description above, we now derive an optimization problem. Assume that we have at least one positive and one negatively labeled data point. Our aim is to find a decision function $f_{w,b}(x) = \mathrm{sgn}(w^T x + b)$ such that $f_{w,b}(x^{(i)}) = y^{(i)}$. Assuming that the training data are linearly separable (we will deal with the inseparable case afterwards), our canonical hyperplane requirement (2) implies that

$$y^{(i)}(w^T i^{(+)} b) \geqslant 1, \qquad 1 \leqslant i \leqslant n.$$

Thus, the task of maximizing the margin while separating the data can be written as

$$\max_{w,b} \frac{1}{\|w\|} \qquad \text{s.t.} \quad \min_{1\leqslant i\leqslant n} y^{(i)}(w^T x^{(i)} + b) = 1.$$

Let us rewrite this problem as a convex optimization problem:

$$\begin{aligned} \min_{w,b} \quad & \tfrac{1}{2}\|w\|^2 \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geqslant 1, \quad 1 \leqslant i \leqslant n. \end{aligned} \tag{5}$$

Problem (5) is called the **Hard-SVM** (the hard margin SVM) in its primal formulation. This problem is convex, with a strictly convex cost function and linear constraints. If the constraints are feasible, then the problem has strong duality, whereby the KKT conditions are necessary and sufficient.

The names comes from the "hard" constraint on all the training data.

Note: If you are unfamiliar with KKT conditions, Lagrangians, and duality, we recommend reading Chapter 5 of "Convex Optimization" by Boyd and Vandenberghe. Another accessible source is "Lagrange multipliers without permanent scarring" by Dan Klein.

Let us analyze the optimality conditions of problem (5). Consider the Lagrangian

$$L(w, b, \alpha) := \tfrac{1}{2}\|w\|^2 - \sum_i \alpha_i [y^{(i)}(w^T x^{(i)} + b) - 1]. \tag{6}$$

We have the following KKT conditions for candidate optimal points (w, b, α):

$$\begin{aligned} \frac{\partial L}{\partial w} = 0, \quad \frac{\partial L}{\partial b} &= 0 && \text{(stationarity)} \\ \alpha_i [y^{(i)}(w^T x^{(i)} + b) - 1] &= 0 \quad \forall i && \text{(complementary slackness)} \\ \alpha_i &\geqslant 0 \quad \forall i && \text{(dual feasibility).} \end{aligned}$$

The stationarity condition simplifies to yield

$$w = \sum_i \alpha_i y^{(i)} x^{(i)}, \quad \sum_i \alpha_i y^{(i)} = 0,$$

while from complementary slackness (and dual feasibility) we see that when the Lagrange multipliers $\alpha_i > 0$, the corresponding primal constraint $y^{(i)}(w^T x^{(i)} + b) = 1$ is tight. These training data points lie closest to the separating hyperplane. In fact, if we recall the convex hull view of Fig. 1, it can be shown that these points lie on supporting hyperplanes of the respective convex hulls. It is for this reason that these points are called **support vectors**.

Note: All the remaining training examples are irrelevant, as their constraints are satisfied automatically. Thus, if we were to discard them, the obtained separating hyperplane would

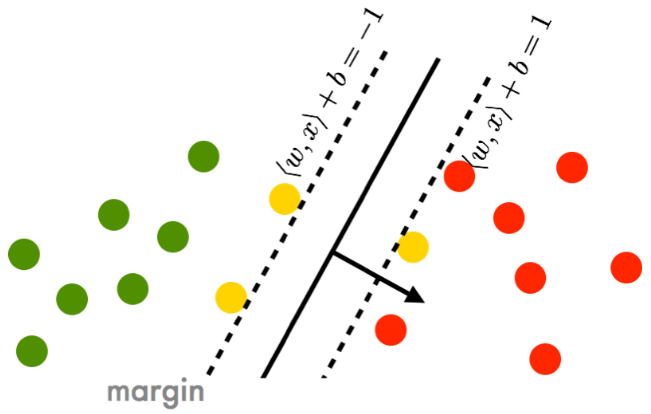

Figure 4: Support vectors, marked in yellow, define the extent of the margin.

be the same. Of course, we do not know *a priori* which points will be support vectors, so we cannot discard non support vectors in advance (although this idea plays a role in the design of certain "active-set" methods for the more general SVM problem that we will soon consider). Moreover, observe that only the support vectors contribute to the classifier that we learn, because we have $w = \sum_i \alpha_i y^{(i)} x^{(i)}$, whereby only the training data points that become support vectors contribute to the classifier (which predicts $\operatorname{sgn}(w^T x + b)$ for a test pattern x). This "sparse" nature of SVMs has also contributed to their popularity (and potential computational speed at test time).

Dual problem. Corresponding to the primal problem (5), upon considering the Lagrangian (6) we arrive at the dual problem

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} & \left[g(\alpha) := \min_{w,b} L(w, b, \alpha) \right] \\ & = -\tfrac{1}{2} \left\| \sum\nolimits_i \alpha_i y^{(i)} x^{(i)} \right\|^2 + \sum\nolimits_i \alpha_i \\ & \text{s.t.} \quad \alpha_i \geqslant 0, \ 1 \leqslant i \leqslant n \\ & \sum\nolimits_i y^{(i)} \alpha_i = 0. \end{aligned} \tag{7}$$

Exercise: Try to discover the relation between $\sum_i \alpha_i$ and the margin $1/\|w\|$.

2.4 Alternative derivation with convex hulls*

We briefly digress to mention an alternative geometric way to arrive at the Hard-SVM. The material of this section is optional and can be skipped at first reading.

We follow the convex-hull view suggested by Fig. 1. It can be seen that the task of maximizing the margin amounts to finding the closest that the hulls of the positive and negative examples come towards each other. We leave it to the reader to verify that the following optimization formulation achieves this aim.

$$\begin{aligned} \min_c & \quad \left\| \sum\nolimits_{y^{(i)}=+1} c_i x^{(i)} - \sum\nolimits_{y^{(i)}=-1} c_i x^{(i)} \right\|^2 \\ \text{s.t.} & \sum_{y^{(i)}=+1} c_i = 1, \quad \sum_{y^{(i)}=+1} c_i = 1, \quad c_i \geqslant 0 \ \forall i. \end{aligned} \tag{8}$$

We need to recover the scale of w from this and to also suitably adjust b. If you are interested in learning more about this view (including its extension to the linearly inseparable case), please refer to the paper "Duality and Geometry in SVM Classifiers" by Bennett and Bredensteiner.

3 C-SVMs

The alert reader may have noticed that for our discussion above we assumed that the primal problem is feasible. However, as is usually the case, there may be no hyperplane that separates the training data points. In this case, problem 5 is infeasible and has no solution. We get around this difficulty by relaxing the margin constraints by introducing slack variables. In particular, we consider the following **Soft-SVM** problem:

$$\begin{aligned} \min_{w,b,\xi} \quad & \tfrac{1}{2}\|w\|^2 + C\sum\nolimits_i \xi_i \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geqslant 1 - \xi_i, \quad 1 \leqslant i \leqslant n \\ & \xi_i \geqslant 0, 1 \leqslant i \leqslant n. \end{aligned} \tag{9}$$

Observe that by making ξ_i large enough the constraints can always be met. Indeed, if we set $\xi_i = 1$ and $w = 0$ and $b = 0$, the constraints are always satisfied and cost is also reduced. To avoid such a trivial solution we penalize the amount of "slack" introduced into the constraints. This penalty is achieved by adding the term $C\sum_i \xi_i$ to the objective function, where $C > 0$ is a hyperparameter.

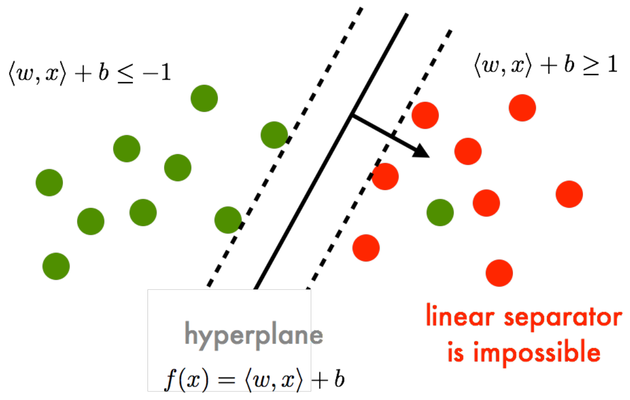
$\langle w, x\rangle + b \leq -1$
$\langle w, x\rangle + b \geq 1$
hyperplane
$f(x) = \langle w, x\rangle + b$
linear separator is impossible

Figure 5: Linearly inseparable data. One of the green points lies on the "wrong" side of the separator, so we have to tolerate training errors while seeking a large margin hyperplane.

As can be seen from (9), whenever $\xi_i = 0$, the margin constraint is met, and the corresponding training data point is *not* a margin error. All the nonzero slacks correspond to margin errors. Thus, formulation (9) makes the tradeoff between the margin width and margin errors apparent, and the amount by which we increase or decrease the importance of training errors is controlled by choosing C. If the slack cost $\sum_i \xi_i$ is significantly larger than the fraction of margin errors, it is likely that the hyperplane obtained by solving (9) does not generalize well.

Dual problem. Forming the Lagrangian of (9) as before, and simplifying we obtain the following *dual formulation* of the Soft-SVM (also called C-SVM):

$$\begin{aligned} \max_{\alpha} \quad & -\frac{1}{2}\left\|\sum\nolimits_i \alpha_i y^{(i)} x^{(i)}\right\|^2 + \sum\nolimits_i \alpha_i \\ \text{s.t.} \quad & \sum\nolimits_i \alpha_i y^{(i)} = 0, \\ & 0 \leqslant \alpha_i \leqslant C,\ 1 \leqslant i \leqslant n. \end{aligned} \tag{10}$$

Comparing (10) with (7) we see the extra upper bound constraint $\alpha_i \leqslant C$.
Remark: The complementary slackness conditions for (9) tell us that

$$\begin{aligned} \alpha_i = 0 &\implies y^{(i)}(w^T x^{(i)} + b) \geqslant 1 && \text{(correctly classified)} \\ \alpha_i = C &\implies y^{(i)}(w^T x^{(i)} + b) \leqslant 1 && \text{(margin error)} \\ 0 < \alpha_i < C &\implies y^{(i)}(w^T x^{(i)} + b) = 1 && \text{(support vector).} \end{aligned}$$

3.1 The ν-SVC

In the C-SVM (either the primal or dual), even though we saw how the parameter C helps us trade off margin maximization with minimization of training error, it is not easy to select this parameter, short of cross-validation.

There is an alternative formulation of the SVM that makes the tradeoff easier to make. This is called the ν-SVC (support vector classifier), and is formulated as follows:

$$\begin{aligned} \min_{w,\xi,b,\rho} \quad & \tfrac{1}{2}\|w\|^2 - \nu\rho + \frac{1}{n}\sum\nolimits_i \xi \\ \text{s.t.} \quad & y^{(i)}(w^T x^{(i)} + b) \geqslant \rho - \xi_i, \;\; 1 \leqslant i \leqslant n, \\ & \xi_i \geqslant 0, \; 1 \leqslant i \leqslant n, \;\; \rho \geqslant 0. \end{aligned} \tag{11}$$

In (11) we have two additional parameters: ν, a hyperparameter and ρ a scaling of the margin that must be optimized. What role does ρ play? Consider a point for which $\xi_i = 0$, then its margin is $2\rho/\|w\|$. The parameter ν upper bounds the margin error (that is, the fraction of points that have $\xi_i > 0$). It can be shown that if $\rho > 0$, then ν is an upper bound on the fraction of margin errors and a lower bound on the fraction of support vectors. So this single parameter allows a direct control on the two competing quantities in an SVM, and given its interpretation, it may be easier to set.

Exercise: Derive the dual of the ν-SVC. Compare with C-SVM.

Note: If ν-SVC solution yields $\rho > 0$, then it can be shown that the usual C-SVM with $C = 1/\rho$ leads to the same decision function (hyperplane classifier).

4 SVM as regularized risk minimization

We penalized the slacks linearly by adding $C\sum_i \xi_i$. One may ask why not something else? Indeed, other choices have also been considered, e.g., $C\sum_i \xi_i^p$ for $p \geqslant 1$. However, consider for the moment what we may actually like to penalize, namely,

$$\|\xi\|_0 = \text{card}(\xi) = \#\text{nonzeros}(\xi),$$

which corresponds to minimizing the number of training errors. Written in this notation, we rewrite the SVM cost function from (7) as

$$\min_{w,b} \quad \tfrac{1}{2}\|w\|^2 + C\sum\nolimits_i [\![y^{(i)} \neq f(x^{(i)})]\!]. \tag{12}$$

The Iverson bracket above can be rewritten as the loss function

$$L(a, g) = L(y^{(i)}, f(x^{(i)})) = L(y^{(i)}, \text{sgn}(w^T x^{(i)} + b)) = [\![y^{(i)}(w^T x^{(i)} + b) \neq 1]\!].$$

With this loss function, the optimization problem (12) is NP-Hard. But by using the soft-margin SVM we make the problem tractable. A brief observation shows that we can rewrite the soft-margin SVM as a regularized loss (risk) minimization problem. Here's how. Consider the constraint

$$y^{(i)}(w^T x^{(i)} + b) \geqslant 1 - \xi_i, \quad \forall i,\ \xi_i \geqslant 0.$$

Since we are trying to make ξ_i small (try to make the argument more rigorous), we see that

$$\xi_i = \max(0, 1 - y^{(i)}(w^T x^{(i)} + b)) = \max\left\{0, 1 - y^{(i)} f(x^{(i)})\right\}.$$

Thus, we can write the C-SVM (7) equivalently in its *hinge-loss* formulation:

$$\min_{w,b} \quad \tfrac{1}{2}\|w\|^2 + C \sum_i \max\left\{0, 1 - y^{(i)} f(x^{(i)})\right\}. \tag{13}$$

The loss in (13) is called the *hinge loss* because of its shape (see Fig. 6), and it provide a convex upper bound to the nonconvex 0/1 loss used in (12).

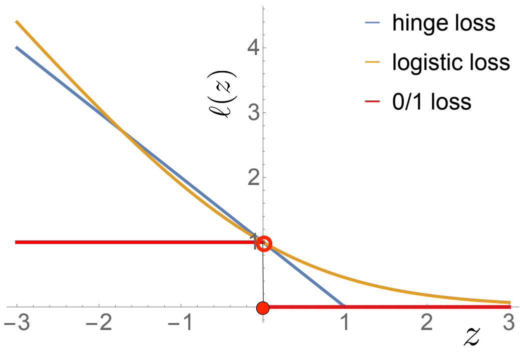

Figure 6: Loss functions, denoted $\ell(z) = \max(0, 1 - z)$ for the hinge loss and $\ell(z) = \log_2(1 + e^{-z})$ for the logistic loss (for SVMs and LR $z = y(w^T x + b)$).

Recall that logistic regression took on a form similar to (13), namely,

$$\min_{w,b} \quad \tfrac{1}{2}\|w\|^2 + \mathrm{NLL}(w, b).$$

In fact, more generally, ℓ_2-norm regularized risk minimization problems in ML are written as (we suppress the bias term for simplicity):

$$\min_{w} \quad \tfrac{1}{2}\|w\|^2 + \frac{1}{n}\sum_i L(y^{(i)}, f(x^{(i)})),$$

where $f(x^{(i)})$ is our "guess" at the true label. The separable (as a sum over training data) structure of this regularized risk minimization problem has made SGD like methods very popular for its solution.

5 Additional remarks

We looked at the basic SVM formulation motivated by the geometric view of seeking a large-margin classifier. Many more variations of the basic theme exist, we have covered only a few key ideas. Some interesting points worth knowing about are the following:

- Novelty detection via 1-class SVMs. This problem is actually an *unsupervised problem*, where we have a dataset drawn from some distribution P, and we wish to estimate a "simple" subset S of the input space such that the probability that a test point from P lies outside the region S equals some a priori specified value in $(0, 1)$. For more details on this topic, the reader is referred to the paper "Support Vector Method for Novelty Detection" by Schölkopf et al. (NIPS 1999; vol 12).

- Running SGD on the hinge-loss formulation is a scalable method for dealing with large datasets. One popular way to solve this formulation is via an SGD variant called Pegasos (covered in Homework 2).
- There exist numerous strategies for obtaining b (there is no unique bias, and various choices can be obtained from the KKT conditions). It is worth noting that in high-dimensional problems, the bias may not be that important.
- An idea key to the early success and adoption of SVMs was how researchers could incorporate prior (domain) knowledge into the formulation, as well as how they could encode invariances. These ideas proved crucial to making SVMs reliably outperform competition and become established as a fundamental classification method.
- We did not cover this point in class, but often one wishes to obtain probabilities from an SVM output. There have been several approaches in the literature for extracting probabilities, i.e., $P(Y|X)$ from the SVM parameters. One of the most popular choices is "Platt scaling" (see Wikipedia for a detailed description). There exist other more recent, theoretically better grounded alternatives, e.g., "Support Vector Machines as Probabilistic Models" by Franc, Zien, Schölkopf (ICML 2011).
- Unsurprisingly, researchers have also considered endowing SVMs with a Bayesian viewpoint. There is reasonably substantive literature available on this topic, with very recent works (dating to 2014).
- SVMs have contributed extensively to the growth of interest in optimization algorithms with machine learning, and they have been solved using numerous techniques by now. Coordinate descent on the dual is a very popular strategy, and is used in the popular libary LIBSVM. For the primal, Pegasos is a popular approach too.
- Multi-class versions of the SVM are also possible, however we omit discussing these.

6 Nonlinear classification, Kernels

We are now ready to discuss nonlinear classification. Indeed, the ability to efficiently perform nonlinear classification was one of the factors that contributed to the SVM's early success. From our presentation above, it may not be apparent (to the uninitiated) how one can perform efficient nonlinear classification using SVMs. This section outlines the principal details of this idea.

6.1 Nonlinear feature maps

An idea that we have also encountered in previous lectures is that of using nonlinear features. The idea here is to map the input data into a (potentially) high dimensional space, and then run the usual linear separators on the new representation. Which map one uses to transform the input data nonlinearly depends on the domain, and is in general a hard question.

Figures 7 and 8 illustrate how patterns that are not linearly separable can become linearly separable after creating nonlinear features. Notice how easy it becomes to separate the XOR data (Fig. 8) by embedding it into $\mathbb{R}^3$.

In general, it is either trial-and-error, or an art on how to create nonlinear maps that embed data into (typically) high dimensional space where they become linearly separable.

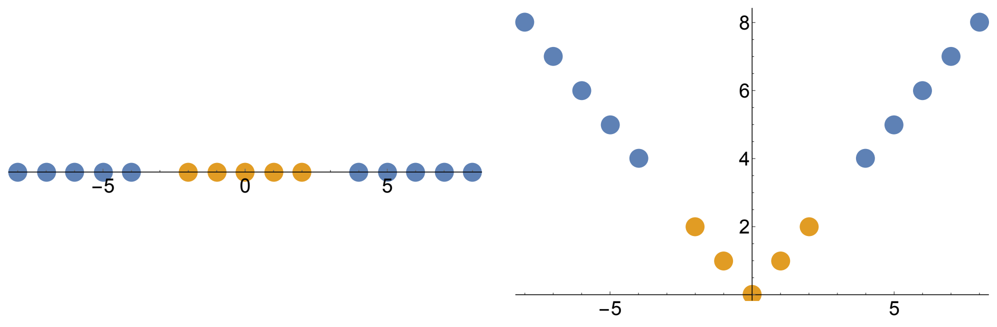

Figure 7: Effect of the nonlinear map $\phi : \mathbb{R} \to \mathbb{R}^2$ given by $x \mapsto (x, |x|)$

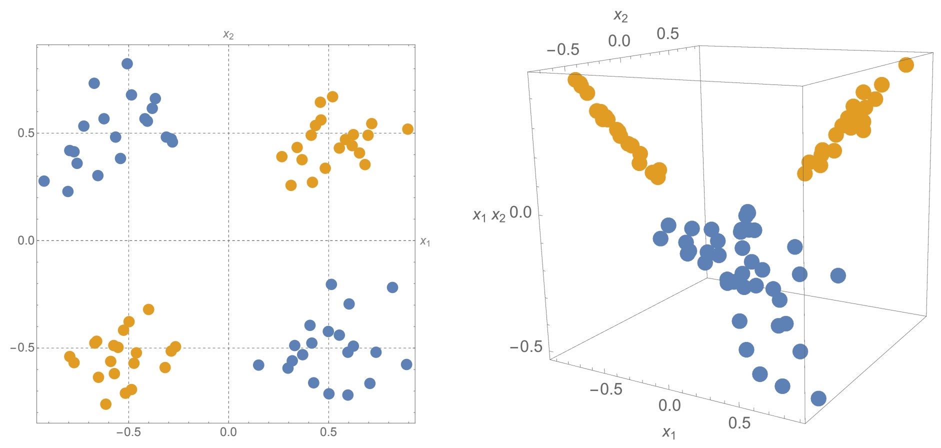

Figure 8: Effect of the nonlinear map $\phi : \mathbb{R}^2 \to \mathbb{R}^3$ given by $x \mapsto (x_1, x_2, x_1x_2)$

You can imagine a variety of such nonlinear maps: the crux being that we still continue to use linear separators, but linear in the nonlinear features $\phi(x)$. In the next lecture, we will see a powerful way to construct nonlinear features using neural networks.
Note: It is important to reflect at this point on the computational and storage ramifications of constructing the nonlinear maps. Typically, we use $\phi : \mathbb{R}^d \to \mathbb{R}^m$, where $m \gg d$ (we can even have $m = \infty$!). In this case, the cost of transforming input data into vectors in $\mathbb{R}^m$ can be very high. Moreover, storing such long vectors is also very expensive, as is the cost of any operation (e.g., $w^T\phi(x)$) that we perform on such data. Remarkably, it turns out that in many cases:

The mapping $\phi(x)$ need not be explicitly performed!

This remarkable idea brings us to the main topic of this section: **kernel functions**.

6.2 Kernel functions

The key assumption here is that we have access to a function that can compute the inner product $\langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$ without having to explicitly construct the "coordinates." While this idea may seem mysterious at first sight, a few examples help illustrate the point.

Example: Consider the feature map $\phi : \mathbb{R}^2 \to \mathbb{R}^3$ given by

$$\phi(x) = (x_1^2, \sqrt{2}x_1x_2, x_2^2).$$

If we compute the inner product between two points $x, z \in \mathbb{R}^2$ by first forming $\phi(x)$ and $\phi(z)$, we obtain two vectors of length 3 each. Thus, computing $\langle \phi(x), \phi(z) \rangle$ will cost us 3 multiplications and 2 additions. However, observe that

$$\begin{aligned} \langle \phi(x), \phi(z) \rangle &= (x_1^2 z_1^2 + 2x_1x_2z_1z_2 + x_2^2 z_2^2) \\ &= (x_1z_1 + x_2z_2)^2 = \langle x, z \rangle^2. \end{aligned}$$

Notice that to compute the inner product $\langle \phi(x), \phi(z) \rangle$, we do *not need* to compute the vectors $\phi(x)$ and $\phi(z)$. We simply compute $\langle x, z \rangle$ (at a cost of 2 multiplications and 1 addition) and then square it. Thus, in this case we have $k(x, z) = \langle x, z \rangle^2$.

Example: Consider the kernel $k(x, z) = \langle x, z \rangle^m$ defined on vectors $x, z \in \mathbb{R}^d$. To compute $k(x, z)$ we require only $d - 1$ multiplications to evaluate $\langle x, z \rangle$ and the cost to compute its m-th power. However, if we were to instead first compute the nonlinear map

$$\phi(x) = (x_{i_1}x_{i_2} \ldots x_{i_m}) \quad 1 \leqslant i_1, i_2, \ldots, i_m \leqslant d,$$

we would require much larger storage and much larger (how much?) amount of computation to perform $\langle \phi(x), \phi(z) \rangle$ explicitly.

Implications for SVM. To solve the dual of the SVM, we assume that we have access to a kernel function k that computes $k(x^{(i)}, x^{(j)}) = \langle \phi(x^{(i)}), \phi(x^{(j)}) \rangle$. Introducing the *kernel matrices* K and $\hat{K}$ with entries given by

$$K_{ij} := k(x^{(i)}, x^{(j)}), \quad 1 \leqslant i, j \leqslant n, \qquad \hat{K}_{ij} = y^{(i)} K_{ij} y^{(j)}, \tag{14}$$

respectively, and the all ones vector $\mathbf{1}_n \in \mathbb{R}^n$, the dual of the SVM becomes

$$\begin{aligned} \max_{\alpha \in \mathbb{R}^n} \quad & -\tfrac{1}{2}\alpha^T \hat{K} \alpha + \alpha^T \mathbf{1}_n \\ & \alpha^T y = 0, \quad \alpha \in [0, C]. \end{aligned} \tag{15}$$

Observe that for problem (15) to be convex, the matrix $\hat{K}$ must be positive (semi)definite. This matrix will be positive (semi)definite if K is positive (semi)definite. And we will soon see that this positive definiteness holds because of how we define a kernel function.

Even though we have "kernelized" the dual, what we are truly interested in is the classifier, namely

$$w = \sum\nolimits_i \alpha_i y^{(i)} \phi(x^{(i)}).$$

Naively, this classifier still requires high storage because it involves long vectors $\phi(x^{(i)})$. However, since we are using a linear classifier in the ϕ space, we can simply compute

$$\begin{aligned} \operatorname{sgn}(w^T \phi(x) + b) &= \operatorname{sgn}\left(\sum\nolimits_i \alpha_i y^{(i)} \langle \phi(x^{(i)}), \phi(x) \rangle + b\right) \\ &= \operatorname{sgn}\big(\sum\nolimits_i \alpha_i y^{(i)} k(x^{(i)}, x) + b\big), \end{aligned}$$

which involves only computation of the kernel function.

6.3 Kernel ridge regression

Consider the penalized empirical risk for ridge regression (RR) using nonlinear features:

$$R_n(w) := \frac{1}{n} \sum_{i=1}^{n} (y^{(i)} - \langle w, \phi(x^{(i)}) \rangle)^2 + \frac{\lambda}{2} \|w\|^2. \tag{16}$$

We assume below that $\lambda > 0$. Our aim is to show how, like the kernelized SVM from the previous section, we can solve (16) without explicitly computing or storing $\phi(\cdot)$. Differentiating $R_n(w)$ we must solve

$$\frac{\partial R_n(w)}{\partial w} = 0, \qquad \Longrightarrow \lambda w - \frac{1}{n}\sum_i \underbrace{(y^{(i)} - \langle w, \phi(x^{(i)})\rangle)}_{=:n\lambda\alpha_i}\phi(x^{(i)}) = 0,$$

which we can rewrite as

$$\lambda w = -\lambda \sum\nolimits_i \alpha_i \phi(x^{(i)}), \qquad \Longrightarrow w = \sum\nolimits_i \alpha_i \phi(x^{(i)}),$$

a form that holds provided for all j we can find α_j such that

$$n\lambda\alpha_j = y^{(j)} - \langle w, \phi(x^{(j)})\rangle \qquad 1 \leqslant j \leqslant n. \tag{17}$$

Note that now our predictions are

$$f(x) = \sum\nolimits_i \alpha_i \langle \phi(x^{(i)}), \phi(x)\rangle = \sum\nolimits_i \alpha_i k(x^{(i)}, x),$$

which requires just calls to the kernel function.

It remains to show that we can find suitable α_i values. Plugging in the formula for w into (17) we obtain

$$n\lambda\alpha_j = y^{(j)} - \sum\nolimits_i \alpha_i \langle \phi(x^{(i)}), \phi(x^{(j)})\rangle = y^{(j)} - \sum\nolimits_i \alpha_i k(x^{(i)}, x^{(j)}).$$

Writing the matrix $K = [k(x^{(i)}, x^{(j)})]_{i,j=1}^n$ we obtain a large $(n \times n)$ linear system

$$n\lambda\alpha = y - K\alpha \qquad \Longrightarrow \qquad \alpha = (K + n\lambda I)^{-1} y.$$

Thus, we have managed to kernelize ridge regression. Importantly, note the similarities between the kernel SVM predictor and the Kernel RR predictor

$$\begin{aligned} \text{SVM}: \quad & w = \sum\nolimits_i \alpha_i y^{(i)} \phi(x^{(i)}), \qquad && \alpha_i \text{ dual variables} \\ \text{KRR}: \quad & w = \sum\nolimits_i \alpha_i \phi(x^{(i)}), && \alpha = (K + n\lambda I)^{-1} y. \end{aligned}$$

The most useful fact about representing w as a linear combination of the $\phi(x^{(i)})$ (for SVM, KRR, or any other model where we have such a representation) is that the prediction also reduces to being able to evaluate the kernel function since $w^T\phi(x) = \sum_i \alpha_i k(x^{(i)}, x)$.

This similarity between the SVM and KRR predictors is not a coincidence, but part of a much more general result about regularized problems involving kernels, which roughly says that even though the feature vectors $\phi(x^{(i)})$ are potentially infinite dimensional (and therefore, as is our predictor w), the intrinsic dimension of our predictor is at most n as we can represent it as a linear combination of the n training data points. This claim is made more precise in the section below.

6.4 The representer theorem

We state here an important theorem that presents a general setting under which we can expect the predictor to be a linear combination of the input training data. Initial aspects of this theorem (for the squared loss) were presented in a seminal work of Wahba et al. in the 1970s; much later, Schölkopf, Herbrich, and Smola presented sufficient conditions in 2001. However, only recently in 2013 was the version noted below (which contains both necessary and sufficient conditions) was discovered by Yu et al (ICML 2013).

This theorem uses some Hilbert space terminology that we roughly define subsequently. It can be omitted on first reading.

Theorem 1 *Let $x^{(1)}, \ldots, x^{(n)}$ be training data drawn from a set $\mathcal{X}$; R_n be some loss function from $\mathbb{R}^n \to \mathbb{R}$, and $\mathcal{H}$ the RKHS induced by the kernel $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, with induced norm $\|f\| := \sqrt{\langle f, f \rangle}$. Let $\Omega : \mathcal{H} \to \mathbb{R} \cup \{\infty\}$. Consider the penalized empirical risk minimization problem*

$$\min_{f \in \mathcal{H}} \quad R_n(f(x^{(1)}), \ldots, f(x^{(n)})) + \lambda \Omega(f). \tag{18}$$

Then, at least one of the minimizers of (18) *has the representation $w = f(\cdot) = \sum_i \alpha_i k(\cdot, x^{(i)})$ for some $\alpha \in \mathbb{R}^n$ if and only if*

$$\forall f, g \in \mathcal{H}, \quad \|f\| > \|g\| \implies \Omega(f) \geqslant \Omega(g).$$

Moreover, if Ω is strictly increasing then all the minimizers have this representation.

In other words, if we wish our predictor to enjoy a linear representation in terms of training data, we are forced to use a regularizer that depends monotonically on the Hilbert space norm of the predictor.

This result immediately implies the representations we saw for Kernel-SVM and Kernel RR; using it we can obtain similar conclusions with all kinds of other loss functions. It just relies on the kernel k generating an RKHS.

7 Basic theory of kernels

The alert reader may have noticed that when writing out the representer theorem, as well as when using kernels we did not require the kernel function to be defined over vectors. We permitted any set $\mathcal{X}$. Indeed, this ability to seamlessly work with non-vector valued data, as long as we have a "similarity function" (that is, a kernel) over pairs available, we can perform the usual classification and regression over such data. This powerful ability of kernel methods was also an early contributor to their widespread adoption, e.g., in bioinformatics where researchers were working with genetic data encoded as strings, or phylogenetic trees; in computational chemistry, where the data were small graphs / networks of molecules, etc.

Let us formalize the notion of a kernel function below. Let $\mathcal{X}$ be any nonempty set, and let $x^{(1)}, \ldots, x^{(n)}$ be any set of n elements drawn from $\mathcal{X}$. A kernel function is a map

$$k : \mathcal{X} \times \mathcal{X} \to \mathbb{R},$$

that satisfies a few important properties noted below. In particular, we will reserve the term "kernel function" for a *positive definite (PD) kernel*, also known as *Mercer kernel*, or henceforth, simply a "kernel."

In the literature, sometimes the word "kernel" is also used for any bivariate map $k(\cdot,\cdot)$, but we reserve the use for PD kernels.

Kernel function: Specifically, a map $k : \mathcal{X} \to \mathcal{X} \to \mathbb{R}$ is called a kernel function if there exists a Hilbert space $\mathcal{H}$ and a feature map $\phi : \mathcal{X} \to \mathcal{H}$ such that

(i) The map k is symmetric, i.e., $k(x, x') = k(x', x)$ for all $x, x' \in \mathcal{X}$

(ii) For all $x, x' \in \mathcal{X}$, we have $k(x, x') = \langle \phi(x), \phi(x') \rangle$,

(iii) For all $n \in \mathbb{N}$, for arbitrary $x^{(1)}, \ldots, x^{(n)} \in \mathcal{X}$, the $n \times n$ matrix

$$K = [k(x^{(i)}, x^{(j)})]_{i,j=1}^{n},$$

is positive semidefinite.

Please note that the map ϕ *need not* be unique—corresponding to a given kernel function, there can be an infinite number of different feature maps ϕ such that $k(x, x') = \langle \phi(x), \phi(x') \rangle$. Given the above definition of kernel functions, by using properties of positive definite matrices, we can conclude some important algebraic properties of kernel functions that allow us to construct new kernel functions by combining old ones:

- ***(CC).*** *If* k_1 *and* k_2 *are kernels, then so is* $ak_1 + bk_2$ *for scalars* $a, b \geqslant 0$
- ***(PP).*** *If* k_1 *and* k_2 *are kernels, then so is* $k_1 k_2$*.*

Using these two properties we can easily obtain new kernels. The following examples are immediate:

(i) $k(x, x') = \langle x, x' \rangle$ (Linear kernel); simply from definition

(ii) $\langle x, x' \rangle^m$ (polynomial kernel); inductively use the product property PP.

(iii) $e^{\langle x, x' \rangle} = 1 + \langle x, x' \rangle + \frac{\langle x, x' \rangle^2}{2!} + \cdots$ (exp kernel); use CC and PP

(iv) $e^{-\gamma \|x - x'\|^2}$ (Gaussian RBF kernel; $\gamma > 0$) use PP and (iii).

(v) Challenge: Is $\frac{1}{1+\|x-x'\|}$ a kernel? What about $e^{-\gamma\|x-x'\|}$?

7.1 Additional examples

We include below some interesting examples of kernels. We do not include proofs of these facts. The interested reader is encouraged to attempt proving them on their own.

- **Probability kernel**: Let (Ω, P) be a probability space with measure P. Then for $A, B \in \Omega$ the function
$$k(A, B) := P(A \cap B) - P(A)P(B),$$
is a kernel.

- **Jaccard kernel**: This kernel is very useful for computing a similarity between arbitrary sets. For arbitrary nonempty sets A, B it is defined by
$$k(A, B) = \frac{|A \cap B|}{|A \cup B|}.$$

- **String kernel**: Let $x \in \mathcal{A}^d$ and $x' \in \mathcal{A}^{d'}$ be two strings on the alphabet $\mathcal{A}$, of length d and d', respectively. Then,
$$k(x, x') := \sum_{S \in \mathcal{A}^*} w_S \phi_S(x) \phi_S(x'),$$
is a kernel, where $\mathcal{A}^*$ is the set of all finite strings over the alphabet, w_S is a nonnegative weight function of the string S, and $\phi_S(x)$ is a feature map whose coordinate corresponding to S is 1 if the substring S occurs in x (this feature map can be viewed as a fancy way of doing 1-hot encoding).

- **Graph kernels:** This is a topic of even research interest. Some kernels in this domain are *graphlet kernels* (obtained by enumerating subgraphs of up to a certain size and using feature maps akin to that in the string kernel), the *Weisfeiler-Lehman* kernel, amongst many others. The ideal graph kernel would help characterize graph isomorphism or subgraph isomorphism, but clearly, that would make the kernel rather impractical (subgraph isomorphism is NP-Hard).

- **Translation invariant kernels:** These are kernel functions of the form $k(x, x') = h(x - x')$ for a suitable function. It is known (Bochner's theorem) that a function $h(t)$ generates a translation invariant kernel if and only if its Fourier transform is nonnegative.

- **Tensor product kernel:** Let k_1 and k_2 be kernels overs $\mathcal{X}_1 \times \mathcal{X}_1$ and $\mathcal{X}_2 \times \mathcal{X}_2$, respectively. Then, their tensor product $k_1 \otimes k_2(x_1, x_2, x_1', x_2') = k_1(x_1, x_1')k_2(x_2, x_2')$ is a kernel on the product space $(\mathcal{X}_1 \times \mathcal{X}_2) \times (\mathcal{X}_1 \times \mathcal{X}_2)$.

- **Kernels over kernel matrices:** These kernels have been used in computer vision and other areas where the input data is not vectorial, but covariance matrices (e.g., graph Laplacians, kernel matrices, etc.). For example, if A, B are two positive definite matrices, then

$$k(A,B) := \frac{1}{\det(A+B)^{\alpha}}, \quad \alpha \in \left\{ \tfrac{j}{2} \cup s \mid 1 \leqslant j \leqslant m-1, s \geqslant \tfrac{m-1}{2} \right\},$$

  is a kernel function (proving this is not easy).

- **Monotonic kernels:** Let $k(x,x') = h(x+y)$, where h is given by the Laplace transform of a nonnegative measure. Then k is a kernel. For example, $k(x,x') = 1/(x+x')^{\alpha}$ is a kernel for positive numbers x, x' and $\alpha \geqslant 0$.

7.2 Concluding remarks

An important question that you may wonder about is: *how should we choose a kernel?* There is no clear answer to this question, as it depends on the application. If we have non vector valued data, then either we could first encode it using vectors (e.g., 1-hot encoding) and then use a kernel over vectors, or we could directly define a kernel over non-vectorial data (easier to interpret and use the results). Even if our data are vectors, out of the vast choice of kernels which ones should we use, and with what parameters?

One option is to use a *universal kernel*, like the Gaussian RBF (see e.g., the paper "Universal kernels" by Michelli, Xu, and Zhang (JMLR 2006)). There are also arguments based on generalization properties of different kernels. However, typically, the choice is application dependent, and the user should tune the choice of the kernel to the application. The choice of kernel reflects our "bias" or *prior* knowledge about the application.

A final remark about kernels that we make in passing is that storing a large $n \times n$ kernel matrix is not feasible for big-data problems. Although we did not cover the topics in the lectures, on how to scale kernel methods to problems where n is large, we mention here the Nyström approximation as a popular way to scale up kernel methods. Many other methods exist that are based on approximating the kernel matrix, or the feature map. The reader is encouraged to consult the wider literature on these topics if they are interested in scaling up kernel methods (which remains a topic of current research interest).

8 Reproducing kernel Hilbert Space*

The material in this section is optional, but we encourage you to read through it. It provides a superficial sketch of how we build the Reproducing Kernel Hilbert Space, better known as RKHS, corresponding to a kernel function.

Think of ϕ as a map that takes x to the space of real valued functions on x. Thus,

$$\phi \equiv x \mapsto \mathbb{R}^{\mathcal{X}} = \{f : \mathcal{X} \to \mathbb{R}\}.$$

Thus, x is mapped to a function parameterized by x, we write this as $x \mapsto k(\cdot, x)$. In other words, $\phi(x)$ denotes the function that assigns the value $k(x', x)$ to a point $x' \in \mathcal{X}$. Thus,

$$[\phi(x)](\cdot) = k(\cdot, x),$$

where the ("long vector") $\phi(x)$ is actually a function. Thus, each pattern in the input data gets mapped into a function, and we may consider that each pattern is now represented by its similarity to *all other points in the domain!* This representation seems quite rich.

Now, consider the set of functions obtained from training data: $\left\{\phi(x^{(1)}), \ldots \phi(x^{(n)})\right\}$ for $n \in \mathbb{N}$. We turn this set into a vector space and endow this space with a dot product. Then, we show that this dot product actually satisfies $k(x, x') = \langle \phi(x), \phi(x') \rangle$.

Once we have performed the above construction, then we see that for all functions ψ in the span of $\left\{\phi(x^{(1)}), \ldots \phi(x^{(n)})\right\}$ (i.e., in the vector space we constructed), we have the *reproducing property*:

$$\langle k(\cdot, x), \psi \rangle = \psi(x).$$

This leads us to formally define an RKHS as follows.

RKHS. Let $\mathcal{H}$ be a Hilbert space of functions $f : \mathcal{X} \to \mathbb{R}$. Then, $\mathcal{H}$ is called an RKHS if there exists a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ with the following properties:

(i) k has the reproducing property: $\langle \psi, k(\cdot, x) \rangle = \psi(x)$ for all $\psi \in \mathcal{H}$

(ii) In particular, $\langle k(\cdot, x), k(\cdot, x') \rangle = k(x, x')$,

(iii) k spans $\mathcal{H}$, i.e., $\mathcal{H} = \overline{\text{span}\{k(\cdot, x) \mid x \in \mathcal{X}\}}$

A key aspect of RKHS's is that uniquely determines the kernel k. That is, for each kernel there is a unique RKHS where the kernel has the reproducing property. Note that this just means that $\mathcal{H}$ is unique; the map ϕ need not be unique.