EECS 553: Machine Learning (ECE) **University of Michigan**

Kernel Methods

Winter 2023 **Clayton Scott**

These notes introduce nonlinear feature maps, a powerful technique for converting a linear algorithm to a nonlinear one, and kernels, a structure that makes this technique implementable in high dimensional settings.

1 Nonlinear Feature Maps

One way to create a nonlinear method for regression or classification is to first transform the feature vector via a nonlinear map

$$\Phi : \mathbb{R}^d \to \mathbb{R}^m$$

and then apply a linear method to the transformed data $\Phi(\boldsymbol{x}_1), ..., \Phi(\boldsymbol{x}_n)$. In regression, the nonlinear function estimate is:

$$f(x) = \boldsymbol{w}^T \Phi(\boldsymbol{x}) + b$$

where $\boldsymbol{w} \in \mathbb{R}^m, b \in \mathbb{R}$.

Example 1. The goal of this example is to determine the least squares cubic polynomial fit to training data $(x_1, y_1), ..., (x_n, y_n)$ where $x_i \in \mathbb{R}, y_i \in \mathbb{R}$.

Let's write $f(x) = a + bx + cx^2 + dx^3 = \boldsymbol{w}^T \Phi(x) + a$ where

$$w = \begin{bmatrix} b \\ c \\ d \end{bmatrix} \qquad \Phi(x) = \begin{bmatrix} x \\ x^2 \\ x^3 \end{bmatrix}.$$

Note that $\Phi(x)$ is nonlinear.

Since

$$\sum_i (y_i - f(x_i))^2 = \sum_i (y_i - w^T \Phi(x_i) - a)^2,$$

The minimizer is

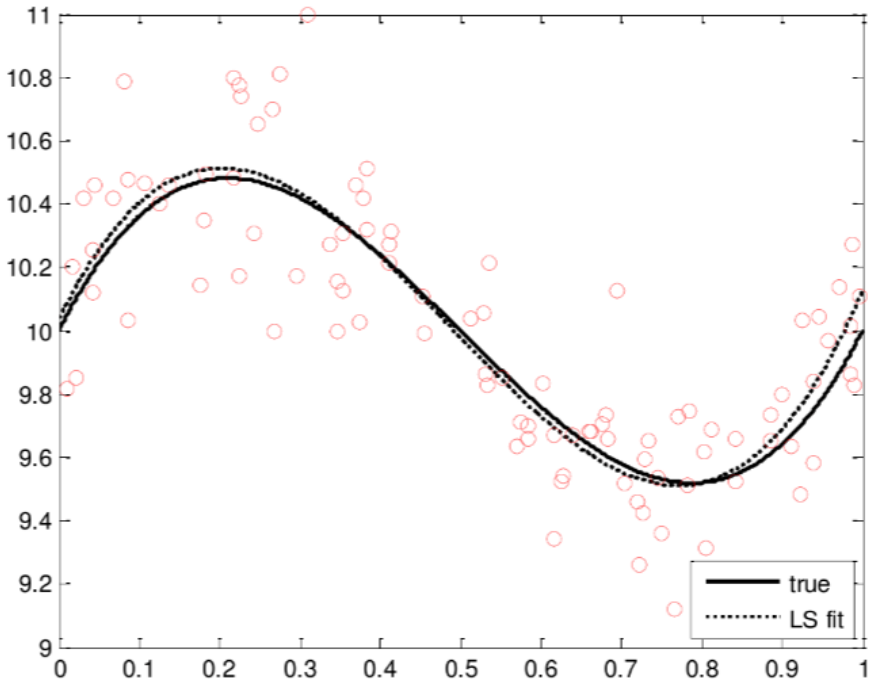
$$\begin{pmatrix} a \\ b \\ c \\ d \end{pmatrix} = (\boldsymbol{X}^T \boldsymbol{X})^{-1} \boldsymbol{X}^T y$$

where

$$y = \begin{bmatrix} y_1 \\ \vdots \\ y_n \end{bmatrix}, \boldsymbol{X} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 \\ 1 & x_2 & x_2^2 & x_2^3 \\ & \vdots & & \\ 1 & x_n & x_n^2 & x_n^3 \end{bmatrix}$$

See Figure 1.

Figure 1: Least squares cubic polynomial fit.



The preceding example is a good motivation for regularization. As the polynomial degree increases, the matrix $\boldsymbol{X}^T\boldsymbol{X}$ becomes ill-conditioned, and regularization is necessary to avoid over-fitting.

In classification, the nonlinear classifier is expressed $f(\boldsymbol{x}) = \text{sign}\{\boldsymbol{w}^T\Phi(\boldsymbol{x}) + b\}$

Example 2. Consider the data shown in Figure 2. This data set can be perfectly classified by a circular classifier, that is, a classifier of the form

$$f(\boldsymbol{x}) = \text{sign}\{(x_1 - c_1)^2 + (x_2 - c_2)^2 - r^2\}$$

for a certain center

$$\boldsymbol{c} = \begin{bmatrix} c_1 \\ c_2 \end{bmatrix}$$

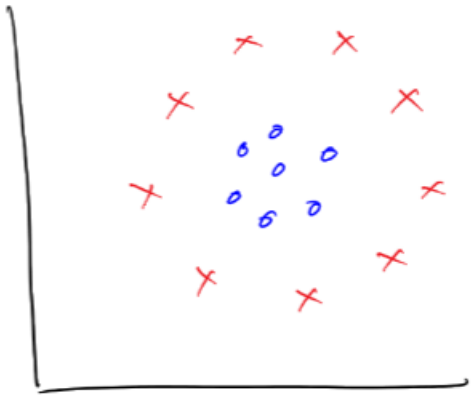
and radius r, and where the feature vector is

$$\boldsymbol{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} \in \mathbb{R}^2.$$

This circular classifier can be viewed as a linear classifier in a transformed feature space. Consider

$$\Phi(\boldsymbol{x}) = \begin{bmatrix} x_1 \\ x_2 \\ x_1x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

Figure 2: Data separated by a circular classifier.



Then the circular classifier can be expressed

$$f(\boldsymbol{x}) = \text{sign}(\boldsymbol{w}^T \Phi(\boldsymbol{x}) + c)$$

where

$$w = \begin{bmatrix} -2c_1 \\ -2c_2 \\ 0 \\ 1 \\ 1 \end{bmatrix}, \qquad b = c_1^2 + c_2^2 - r^2.$$

In both of the preceding examples, it is also possible to eliminate the offset term by incorporating a constant entry of 1 into Φ. The point of these examples is that while linear predictors may not work well in the original feature space, they may be highly effective in a transformed feature space.

Many nonlinear machine learning methods are based on nonlinear feature transformations. Neural networks are a prominent example that we will return to. We now turn our attention to another class of methods known as kernel methods.

2 Kernels

One issue with nonlinear feature transformations is the way m explodes as d increases. $\Phi(\boldsymbol{x})$ cannot be computed explicitly, or stored, because it is too large. For example, if we extend the cubic polynomial to arbitrary degree p and dimension d, the total number of terms in the feature expansion is $\binom{p+d}{d}$. If the dimension is 100 and the degree is 10, this number exceeds 46 trillion!

Fortunately, the following two facts save us: (1) Many machine learning algorithms depend on $\Phi(\boldsymbol{x})$ only via *inner products* $\langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{x}') \rangle$, and (2) For certain Φ, the function

$$k(\boldsymbol{x}, \boldsymbol{x}') = \langle \Phi(\boldsymbol{x}), \Phi(\boldsymbol{x}') \rangle$$

can be computed efficiently even if m is huge (and possibly infinite)!

If a machine learning algorithm satisfies the first property, and Φ satisfies the second property, then we can use the function k to incorporate the nonlinear feature map Φ into the algorithm, without ever having

to explicitly compute $\Phi(\boldsymbol{x})$. Such functions k are called *kernels*,[1] and the algorithms obtained by using them are known as *kernel methods*. We will return to this idea below, and see several examples of kernel methods in subsequent lectures. First, we offer two equivalent formal definitions of kernels and offer some examples.

2.1 Inner Product Kernels

We begin with a simple example to illustrate the second property needed for kernel methods.

Example 3. When $d = 2$, the function $k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^T\boldsymbol{v})^2$ is associated to an $m = 3$ dimensional feature map as follows:

$$\begin{aligned} k(\boldsymbol{u}, \boldsymbol{v}) &= (\boldsymbol{u}^T\boldsymbol{v})^2 \\ &= \left(\begin{bmatrix} u_1 & u_2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \right)^2 \\ &= (u_1v_1 + u_2v_2)^2 \\ &= u_1^2v_1^2 + 2u_1u_2v_1v_2 + u_2^2v_2^2 \\ &= \langle \Phi(\boldsymbol{u}), \Phi(\boldsymbol{v}) \rangle \end{aligned}$$

where

$$\Phi(\boldsymbol{u}) = \begin{bmatrix} u_1^2 \\ \sqrt{2}u_1u_2 \\ u_2^2 \end{bmatrix}.$$

Extending to arbitrary d we have

$$\begin{aligned} k(\boldsymbol{u}, \boldsymbol{v}) &= (\boldsymbol{u}^T\boldsymbol{v})^2 \\ &= (\sum_{i=1}^{d} u_iv_i)^2 \\ &= (\sum_{i} u_iv_i)(\sum_{j} u_jv_j) \\ &= \sum_{i=1}^{d}\sum_{j=1}^{d} u_iu_jv_iv_j \\ &= \langle \Phi(\boldsymbol{u}), \Phi(\boldsymbol{v}) \rangle \end{aligned}$$

where $\Phi(\boldsymbol{u}) = [u_1^2 \ \cdots \ u_d^2 \ \sqrt{2}u_1u_2 \ \cdots \ \sqrt{2}u_{d-1}u_d]^T$, and so $m = d + \frac{d(d-1)}{2}$.

Generalizing the above to arbitrary polynomial degree,

$$\begin{aligned} k(\boldsymbol{u}, \boldsymbol{v}) &= (\boldsymbol{u}^T\boldsymbol{v})^p \\ &= \sum_{(j_1 \cdots j_d)} \binom{p}{j_1 \cdots j_d} u_1^{j_1} \cdots u_d^{j_d} v_1^{j_1} \cdots v_d^{j_d} \\ &= \langle \Phi(\boldsymbol{u}), \Phi(\boldsymbol{v}) \rangle \end{aligned}$$

where $\Phi(\boldsymbol{u}) = [\cdots, \sqrt{\binom{p}{j_1 \cdots j_d}} u_1^{j_1} \cdots u_d^{j_d}, \cdots]^T$. In words, the features associated to k are all monomials of degree p.

[1] The term *kernel* is overloaded in machine learning, and can have a different meaning for different algorithms such as locally linear regression and kernel density estimation.

Another example comes from the function $k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^T\boldsymbol{v} + 1)^p$, whose feature map consists of all monomials (with appropriate scaling) of degree *at most p*. This is left is an exercise.

In these examples, the inner product is just the standard dot product on $\mathbb{R}^d$. More generally, we can allow the range of Φ to be any *(real) inner product space*, which is a vector space V on which we can define a real-valued function $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$ (called an *inner product*), which must satisfy the following axioms:

1. $\langle \alpha_1 \boldsymbol{u}_1 + \alpha_2 \boldsymbol{u}_2, \boldsymbol{v} \rangle = \alpha_1 \langle \boldsymbol{u}_1, \boldsymbol{v} \rangle + \alpha_2 \langle \boldsymbol{u}_2, \boldsymbol{v} \rangle \quad \forall \alpha_1, \alpha_2 \in \mathbb{R}, \boldsymbol{u}_1, \boldsymbol{u}_2, \boldsymbol{v} \in V$
2. $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \langle \boldsymbol{v}, \boldsymbol{u} \rangle \quad \forall\ \boldsymbol{u}, \boldsymbol{v} \in V$
3. $\langle \boldsymbol{u}, \boldsymbol{u} \rangle \geq 0 \quad \forall \boldsymbol{u} \in V$, with equality iff $\boldsymbol{u} = 0$.

Example 4. If $\boldsymbol{A}$ is a symmetric, positive definite matrix, then $\langle \boldsymbol{u}, \boldsymbol{v} \rangle = \boldsymbol{u}^T \boldsymbol{A} \boldsymbol{v}$ is a valid inner product on $\mathbb{R}^d$.

However, we need not be restricted to Euclidean inner product spaces.

We would like to know when it is possible to write a function k as

$$k(\boldsymbol{u}, \boldsymbol{v}) = \langle \Phi(\boldsymbol{u}), \Phi(\boldsymbol{v}) \rangle$$

for some inner product space V and $\Phi : \mathbb{R}^d \to V$. This leads to the following definition.

Definition 1. *We say $k : \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$ is an* inner product kernel *if $\exists$ an inner product space V and a feature map $\Phi : \mathbb{R}^d \to V$ such that*

$$k(\boldsymbol{u}, \boldsymbol{v}) = \langle \Phi(\boldsymbol{u}), \Phi(\boldsymbol{v}) \rangle \quad \forall \boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^d$$

Note that Φ and V are generally not unique.

Example 5. Consider the function

$$k(\boldsymbol{u}, \boldsymbol{v}) = \exp(-\frac{1}{2\sigma^2} \|\boldsymbol{u} - \boldsymbol{v}\|^2),$$

where $\sigma > 0$. Then k is an inner product kernel known as the *Gaussian* kernel. Proof of this fact is left as an exercise. The feature space associated to the Gaussian kernel turns out to be infinite dimensional and is difficult to describe. Nonetheless we can gain some intuition as to what kernel methods that use the Gaussian kernel do, and we will see examples of this later.

2.2 Symmetric, Positive Definite Kernels

One way to determine an inner product kernel is to construct Φ explicitly as we did in the examples above. We can also verify that k is an inner product kernel if it satisfies the following properties.

Definition 2. *Let k: $\mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}$. We say k is* symmetric *if $k(\boldsymbol{u}, \boldsymbol{v}) = k(\boldsymbol{v}, \boldsymbol{u})\ \forall \boldsymbol{u}, \boldsymbol{v}$. We say k is* positive definite *if the $n \times n$ matrix*

$$[k(\boldsymbol{x}_i, \boldsymbol{x}_j)]_{ij} \tag{1}$$

is positive semi-definite[2] for all $n \geq 1$ and all $\boldsymbol{x}_1, ..., \boldsymbol{x}_n \ \in \mathbb{R}^d$.

If k is both symmetric and positive definite, it is referred to as a symmetric, positive definite (SPD) kernel. The matrix in (1) is known as the *kernel matrix*.

Theorem 1. *k is an SPD kernel $\iff$ k is an inner product kernel.*

You are asked to prove one direction of the above equivalence as an exercise. The other direction is more advanced and will not be covered in this course.

[2]This is not a typo. The kernel is said to be positive definite when the associated kernel matrix is always positive semi-definite.

2.3 Examples of kernels

A summary of the above kernels is given below:

1. Homogeneous polynomial kernel
$$k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^T \boldsymbol{v})^p,$$
2. Inhomogeneous polynomial
$$k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^T \boldsymbol{v} + c)^p, \ c > 0$$
3. Gaussian kernel
$$k(\boldsymbol{u}, \boldsymbol{v}) = \exp(-\frac{1}{2\sigma^2}\|\boldsymbol{u} - \boldsymbol{v}\|^2), \ \sigma > 0$$

Another example is the *neural tangent kernel*, which has strong connections to neural networks.

Kernels have also been defined on non-Euclidean feature spaces. For example, kernels have been constructed for strings, trees, graphs, images, functions, and probability distributions. This makes it possible to apply any kernelizable algorithm to data of all of these types. This is because in kernel methods, we only need to be able to compute $k(\boldsymbol{x}, \boldsymbol{x}')$, and never have to work directly with the raw features.

3 Kernel Methods

A machine learning algorithm is said to be *kernelizable* if it is possible to formulate the algorithm such that all training examples $\boldsymbol{x}_i$ and all test instances $\boldsymbol{x}$ occur in inner products of the form $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$ or $\langle \boldsymbol{x}_i, \boldsymbol{x} \rangle$. Many algorithms are kernelizable, but in most cases they must be rewritten in an equivalent form to make this obvious.

To *kernelize* a kernelizable algorithm is to select a kernel k, and replace every occurrence of $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$, where $\boldsymbol{u}, \boldsymbol{v} \in \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{x}\}$ and $\boldsymbol{x}$ denotes a test point at which a learned model is evaluated, by $k(\boldsymbol{u}, \boldsymbol{v})$. By the inner product characterization of kernels, this is equivalent to replacing every occurrence of $\langle \boldsymbol{u}, \boldsymbol{v} \rangle$, where $\boldsymbol{u}, \boldsymbol{v} \in \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n, \boldsymbol{x}\}$, by $\langle \Phi(\boldsymbol{u}), \Phi(\boldsymbol{v}) \rangle$. Therefore, kernelizing an algorithm is equivalent to first applying the nonlinear feature map Φ to elements of the feature space, and then applying the original algorithm to the transformed features. As we saw above, this is an effective way of converting linear algorithms to nonlinear ones. Furthermore, since the algorithm is expressed entirely in terms of the kernel, the feature map does not need to be computed.

Example 6. Consider the nearest centroid classifier. Given training data $(\boldsymbol{x}_i, y_i), i = 1, \ldots, n, \ y_i \in \{-1, +1\}$, define the centroids

$$\boldsymbol{m}_- = \frac{1}{n_-} \sum_{i:y_i=-1} \boldsymbol{x}_i \qquad \boldsymbol{m}_+ = \frac{1}{n_+} \sum_{i:y_i=+1} \boldsymbol{x}_i$$

where $n_- = |\{i : y_i = -1\}|$ and $n_+ = |\{i : y_i = +1\}|$. The nearest centroid classifier is

$$f(\boldsymbol{x}) = \text{sign}\{\|\boldsymbol{x} - \boldsymbol{m}_-\|^2 - \|\boldsymbol{x} - \boldsymbol{m}_+\|^2\}.$$

In words, we assign to a test point $\boldsymbol{x}$ the label associated with the nearest of the two centroids. With a little algebra it can be shown that

$$f(\boldsymbol{x}) = \text{sign}\left\{\boldsymbol{w}^T \boldsymbol{x} + b\right\}$$

where $\boldsymbol{w} = \boldsymbol{m}_+ - \boldsymbol{m}_-$ and $b = \frac{\|\boldsymbol{m}_-\|^2 - \|\boldsymbol{m}_+\|^2}{2}$. To show that this classifier is kernelizable, we need to express $f(\boldsymbol{x})$ such that the training feature vectors $\boldsymbol{x}_i$ and the test vector $\boldsymbol{x}$ only appear in terms of the form $\langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle$, $\langle \boldsymbol{x}_i, \boldsymbol{x} \rangle$, or $\langle \boldsymbol{x}, \boldsymbol{x} \rangle$. To see this simply substitute in the definitions of the centroids which yields

$$f(\boldsymbol{x}) = \text{sign}\left\{ \frac{1}{n_+} \sum_{i:y_i=1} \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle - \frac{1}{n_-} \sum_{i:y_i=-1} \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle + \frac{1}{2} \left(\frac{1}{n_-^2} \sum_{\substack{i:y_i=-1 \\ j:y_j=-1}} \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle - \frac{1}{n_+^2} \sum_{\substack{i:y_i=1 \\ j:y_j=1}} \langle \boldsymbol{x}_i, \boldsymbol{x}_j \rangle \right) \right\}.$$

To kernelize the nearest centroid classifier, we need to choose a kernel k, such as the Gaussian kernel. Then the new classifier, which we might call the *kernel nearest centroid classifier*, is

$$f(\boldsymbol{x}) = \operatorname{sign}\left\{\frac{1}{n_+}\sum_{i:y_i=1} k(\boldsymbol{x}_i, \boldsymbol{x}) - \frac{1}{n_-}\sum_{i:y_i=-1} k(\boldsymbol{x}_i, \boldsymbol{x}) + \frac{1}{2}\left(\frac{1}{n_-^2}\sum_{\substack{i:y_i=-1\\ j:y_j=-1}} k(\boldsymbol{x}_i, \boldsymbol{x}_j) - \frac{1}{n_+^2}\sum_{\substack{i:y_i=1\\ j:y_j=1}} k(\boldsymbol{x}_i, \boldsymbol{x}_j)\right)\right\}.$$

Many kernels, such as the Gaussian kernel, can be viewed as measures of similarity between data points: The closer $\boldsymbol{x}$ and $\boldsymbol{x}'$, the larger $k(\boldsymbol{x}, \boldsymbol{x}')$. If we think of $k(\boldsymbol{x}, \boldsymbol{x}')$ as a measure of similarity between $\boldsymbol{x}$ and $\boldsymbol{x}'$ then, ignoring the bias term, this classifier assigns the label of the class to which the test instance has higher average similarity. The bias term can be interpreted as favoring whichever class has higher average within-class similarity. In the case of the Gaussian kernel, normalized to integrate to one, we can actually say more. The first two terms in the final classifier are actually estimates of their respective class-conditional densities (kernel density estimates, to be precise). If we ignore the bias, the classifier assigns a test point to the class for which it has higher estimated density. Kernel density estimates will be studied in a later chapter.

Although the bias term looks cumbersome, it does not involve the test instance and can therefore be precomputed in $O(n^2 d)$ operations. Then, for each test instance, the classifier can be evaluated in $O(nd)$ operations. This tends to be true for kernel methods in general: the kernelized version of an algorithm tends to have greater computational complexity than the original by one or two factors of n. This can still be much more efficient than computing $\Phi(\boldsymbol{x})$ explicitly when m is large.

Remark 1. The process of kernelizing a kernelizable algorithm is also known as *the kernel trick*.

Exercises

1. (☆) Use properties of Vandermonde matrices to prove that if the data points are unique, then in polynomial regression with $p < n$, the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is invertible.

2. (★) If the smallest eigenvalue of a matrix is too small (relative to the largest eigenvalue), the matrix is said to be *ill-conditioned*. This means that while it may be invertible, computation of the inverse is prone to large numerical errors. In polynomial regression with p sufficiently large relative to n, the matrix $\boldsymbol{X}^T\boldsymbol{X}$ is ill-conditioned. Show that regularization overcomes this issue. In particular, show that adding a quadratic regularizer $\lambda\|\boldsymbol{\theta}\|^2$ to the least-squares objective makes the smallest eigenvalue (of the matrix that needs to be inverted) at least λ.

3. (★★) Let $k(\boldsymbol{u}, \boldsymbol{v}) = (\boldsymbol{u}^T\boldsymbol{v} + 1)^3$ where $\boldsymbol{u}, \boldsymbol{v} \in \mathbb{R}^2$. Find Φ such that $k(\boldsymbol{u}, \boldsymbol{v}) = \langle \Phi(\boldsymbol{u}), \Phi(\boldsymbol{v}) \rangle$.

4. (☆☆) Extend the previous example to arbitrary d and p.

5. (☆☆) Show that the number of monomials of degree p in d variables is $\binom{d+p-1}{p}$.

6. (☆) Use the previous result to show that the number of monomials of degree at most p in d variables is $\binom{d+p}{p}$.

7. (☆☆) Show that the nearest centroid classifier is equivalent to LDA in the situation where the class priors are equal and the covariance matrix is a multiple of the identity.

8. (★★) Kernelize the k-nearest neighbor classifier. Show that if $k = 1$ and the kernel is the Gaussian kernel, then the original and kernelized classifiers are the same.

9. (★★) Show that the sum of two kernels is a kernel.

10. (★★★) This problem asks you to kernelize logistic regression by kernelizing the Newton-Raphson algorithm for logistic regression. Since kernelizing corresponds to a high dimensional feature space, we will consider the regularized form of logistic regression. To simplify the problem somewhat, let's also assume that there is no offset term, so that the class probability model is

$$\eta(\boldsymbol{x}; \boldsymbol{\theta}) = \frac{1}{1 + \exp(-\boldsymbol{\theta}^T\boldsymbol{x})} = g(\boldsymbol{\theta}^T\boldsymbol{x}),$$

where $g(t) = 1/(1 + e^{-t})$. Let $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$ be training data for binary classification, where $\boldsymbol{x}_i \in \mathbb{R}^d$ and $y_i \in \{0, 1\}$.

From an earlier exercise on logistic regression (and making a slight modification to account for the lack of an offset term), the Newton-Raphson update has the following form:

$$\boldsymbol{\theta}^{new} = \boldsymbol{\theta} + (\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}(\boldsymbol{X}^T\boldsymbol{v} - \lambda\boldsymbol{\theta}),$$

where $\boldsymbol{X}$ is an $n \times d$ matrix whose ith row is $\boldsymbol{x}_i^T$, $\boldsymbol{W}$ is a diagonal matrix with ith diagonal entry equal to $g(\boldsymbol{\theta}^T\boldsymbol{x}_i)(1 - g(\boldsymbol{\theta}^T\boldsymbol{x}_i))$, $\boldsymbol{v}$ is an $n \times 1$ column vector with ith entry given by $y_i - g(\boldsymbol{\theta}^T\boldsymbol{x}_i)$, and $\lambda > 0$ is the regularization parameter.

   (a) The matrix inversion lemma states that

$$(\boldsymbol{P} + \boldsymbol{Q}\boldsymbol{R}\boldsymbol{S})^{-1} = \boldsymbol{P}^{-1} - \boldsymbol{P}^{-1}\boldsymbol{Q}(\boldsymbol{R}^{-1} + \boldsymbol{S}\boldsymbol{P}^{-1}\boldsymbol{Q})^{-1}\boldsymbol{S}\boldsymbol{P}^{-1},$$

   provided all of the matrix operations are well defined. Re-express the logistic regression iterative update step by applying the matrix inversion lemma to $(\boldsymbol{X}^T\boldsymbol{W}\boldsymbol{X} + \lambda\boldsymbol{I})^{-1}$.

We are free to initialize the algorithm in any way we choose. Let us initialize $\boldsymbol{\theta}$ so that $\boldsymbol{\theta} = \boldsymbol{X}^T\boldsymbol{\alpha}$ for some vector $\boldsymbol{\alpha} \in \mathbb{R}^n$. In other words, $\boldsymbol{\theta}$ is a linear combination of data points. For example, we could initialize $\boldsymbol{\theta}$ to be the first data point $\boldsymbol{x}_1$, in which case $\boldsymbol{\alpha} = [1\ 0\ 0\ \cdots\ 0]^T$.

(b) Now consider an arbitrary step in the iterative procedure. Show that if $\boldsymbol{\theta} = \boldsymbol{X}^T\boldsymbol{\alpha}$ for some $\boldsymbol{\alpha}$, then $\boldsymbol{\theta}^{new} = \boldsymbol{X}^T\boldsymbol{\alpha}^{new}$ for some $\boldsymbol{\alpha}^{new}$. Give an update formula for $\boldsymbol{\alpha}^{new}$ in terms of $\boldsymbol{\alpha}$ and the other quantities.

(c) Explain how to implement kernel logistic regression. That is, show how to incorporate inner product kernels into the iterative parameter update, and also how to evaluate the final classifier.

11. (☆☆☆) In this problem you are asked to show that the Gaussian kernel

$$k(\boldsymbol{x}, \boldsymbol{x}') = C \exp\left(-\frac{1}{2\sigma^2}\|\boldsymbol{x} - \boldsymbol{x}'\|^2\right), \qquad C > 0, \sigma > 0,$$

is an inner product kernel (equivalently, a symmetric and positive definite kernel). You will be asked to verify various properties that let you construct kernels from other kernels, and you will see that both characterizations of kernels (IP/SPD) are useful in this regard.

(a) Consider a sequence of symmetric, positive definite (SPD) kernels $\{k_n\}$ that coverges pointwise to a function k, i.e., for all $\boldsymbol{x}, \boldsymbol{x}'$, $\lim_{n\to\infty} k_n(\boldsymbol{x}, \boldsymbol{x}') = k(\boldsymbol{x}, \boldsymbol{x}')$. Show that k is also a SPD kernel.

(b) Prove that if $k_1(\boldsymbol{x}, \boldsymbol{x}')$ and $k_2(\boldsymbol{x}, \boldsymbol{x}')$ are SPD kernels, then so is $k(\boldsymbol{x}, \boldsymbol{x}') = k_1(\boldsymbol{x}, \boldsymbol{x}')k_2(\boldsymbol{x}, \boldsymbol{x}')$.
Hint: Consider zero mean, independent random vectors V_1 and V_2 whose covariances are kernel matrices K_1 and K_2. Determine another random vector whose covariance is the element-wise product of K_1 and K_2.

(c) Let $f(t) = \sum_{n=0}^{\infty} a_n t^n$ be a power series that converges for all $t \in \mathbb{R}$, and for which all $a_n \geq 0$. Argue that $k(\boldsymbol{x}, \boldsymbol{x}') = \sum_{n=0}^{\infty} a_n (k(\boldsymbol{x}, \boldsymbol{x}'))^n$ is a kernel.

(d) Deduce that the exponential kernel $k(\boldsymbol{x}, \boldsymbol{x}') = \exp(\gamma\langle \boldsymbol{x}, \boldsymbol{x}'\rangle)$ is an inner product kernel, where $\gamma > 0$ and $\langle \cdot, \cdot \rangle$ denotes the dot product on $\mathbb{R}^d$.

(e) Show that if k is an inner product kernel, then so is the normalized kernel

$$\tilde{k}(\boldsymbol{x}, \boldsymbol{x}') = \begin{cases} 0, & \text{if } k(\boldsymbol{x}, \boldsymbol{x}) = 0 \text{ or } k(\boldsymbol{x}', \boldsymbol{x}') = 0 \\ \frac{k(\boldsymbol{x}, \boldsymbol{x}')}{k(\boldsymbol{x}, \boldsymbol{x})k(\boldsymbol{x}', \boldsymbol{x}')}, & \text{otherwise.} \end{cases}$$

(f) Deduce that the Gaussian kernel is an inner product kernel.