Kernels

Note: Unless otherwise noted all references including images are from the required textbook, Machine Learning: A Probabilistic Perspective by Kevin P. Murphy.

Kernels

So far, we have been assuming that each object that we wish to classify or process in anyway can be represented as a fixed-size feature vector, typically of the form $\mathbf{x}_i \in \mathbb{R}^D$. However, for certain kinds of objects, it is not clear how to best represent them as fixed-sized feature vectors.

Kernels

For example, how do we represent a text document or protein sequence, which can be of variable length? or a molecular structure, which has complex 3d geometry? or an evolutionary tree, which has variable size and shape?

Kernels

One approach is to assume that we have some way of measuring the similarity between objects, that doesn’t require preprocessing them into feature vector format. For example, when comparing strings, we can compute the edit distance between them. Let $\kappa(\mathbf{x}, \mathbf{x}') \geq 0$ be some measure of similarity between objects $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, where $\mathcal{X}$ is some abstract space; we will call κ a kernel function.

Kernel Functions

We define a kernel function to be a real-valued function of two arguments, $\kappa(\mathbf{x}, \mathbf{x}') \in \mathbb{R}$, for $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$. Typically the function is symmetric (i.e., $\kappa(\mathbf{x}, \mathbf{x}') = \kappa(\mathbf{x}', \mathbf{x})$), and non-negative (i.e., $\kappa(\mathbf{x}, \mathbf{x}') \geq 0$), so it can be interpreted as a measure of similarity, but this is not required.

RBF Kernels

The **radial basis function (RBF) kernel** is defined by

$$\kappa(\mathbf{x}, \mathbf{x}') = \exp\left(-\frac{||\mathbf{x} - \mathbf{x}'||^2}{2\sigma^2}\right)$$

where σ^2 is known as the bandwidth.

Kernels for Comparing Documents

When performing document classification or retrieval, it is useful to have a way of comparing two documents, $\mathbf{x}_i$ and $\mathbf{x}_{i'}$. If we use a bag of words representation, where x_{ij} is the number of times words j occurs in document i, we can use the **cosine** similarity, which is defined by

$$\kappa(\mathbf{x}_i, \mathbf{x}_{i'}) = \frac{\mathbf{x}_i^T \mathbf{x}_{i'}}{||\mathbf{x}_i||_2 ||\mathbf{x}_{i'}||_2}$$

Kernels for Comparing Documents

This quantity measures the cosine of the angle between $\mathbf{x}_i$ and $\mathbf{x}_{i'}$ when interpreted as vectors. Since $\mathbf{x}_i$ is a count vector (and hence non-negative), the cosine similarity is between 0 and 1, where 0 means the vectors are orthogonal and therefore have no words in common.

Kernels for Comparing Documents

Unfortunately, this simple method does not work very well, for two main reasons. First, if $\mathbf{x}_i$ has any word in common with $\mathbf{x}_{i'}$, it is deemed similar, even though some popular words, such as “the” or “and” occur in many documents, and are therefore not discriminative. (These are known as **stop words**.)

Kernels for Comparing Documents

Second, if a discriminative word occurs many times in a document, the similarity is artificially boosted, even though word usage tends to be bursty, meaning that once a word is used in a document it is very likely to be used again.

Kernels for Comparing Documents

We can significantly improve performance using some simple preprocessing. The idea is to replace the word count vector with a new feature vector called the **TF-IDF** representation, which stands for “term frequency inverse document frequency”.

Kernels for Comparing Documents

First, the term frequency is defined as a log-transform of the count:

$$\mathrm{tf}(x_{ij}) \triangleq \log(1 + x_{ij})$$

This reduces the impact of words that occur many times within one document.

Kernels for Comparing Documents

Second, the inverse document frequency is defined as

$$\mathrm{idf}(j) \triangleq \log \frac{N}{1 + \sum_{i=1}^{N} \mathbb{I}(x_{ij} > 0)}$$

where N is the total number of documents, and the denominator counts how many documents contain term j.

Kernels for Comparing Documents

Finally, we define

$$\text{tf-idf}(\mathbf{x}_i) \triangleq [\text{tf}(x_{ij}) \times \text{idf}(j)]_{j=1}^{V}$$

We then use this inside the cosine similarity measure:

$$\kappa(\mathbf{x}_i, \mathbf{x}_{i'}) = \frac{\phi(\mathbf{x}_i)^T \phi(\mathbf{x}_{i'})}{||\phi(\mathbf{x}_i)||_2 ||\phi(\mathbf{x}_{i'})||_2}$$

where $\phi(x) = \text{tf-idf}(x)$.

Mercer Kernels

Some methods require that the kernel function satisfy the requirement that the Gram matrix, defined by

$$\mathbf{K} = \begin{pmatrix} \kappa(\mathbf{x}_1, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_1, \mathbf{x}_N) \\ & \vdots & \\ \kappa(\mathbf{x}_N, \mathbf{x}_1) & \cdots & \kappa(\mathbf{x}_N, \mathbf{x}_N) \end{pmatrix}$$

be positive definite for any set of inputs $\{\mathbf{x}_i\}_{i=1}^{N}$. We call such a kernel a **Mercer kernel**, or **positive definite kernel**.

Mercer Kernels

In general, if the kernel is Mercer, then there exists a function ϕ mapping $\mathbf{x} \in \mathcal{X}$ to $\mathbb{R}^D$ such that

$$\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}')$$

where ϕ depends on the eigen functions of κ

Mercer Kernels

For example, consider the **polynomial kernel** $\kappa(\mathbf{x}, \mathbf{x}') = (\gamma \mathbf{x}^T \mathbf{x}' + r)^M$, where r > 0. The corresponding feature vector $\phi(x)$ will contain all terms up to degree *M*. If $M = 2$, γ=r=1 and $\mathbf{x}, \mathbf{x}' \in \mathbb{R}^2$, we have

$$\begin{aligned}(1 + \mathbf{x}^T\mathbf{x}')^2 &= (1 + x_1x_1' + x_2x_2')^2 \\ &= 1 + 2x_1x_1' + 2x_2x_2' + (x_1x_1')^2 + (x_2x_2')^2 + 2x_1x_1'x_2x_2'\end{aligned}$$

Mercer Kernels

This can be written as , where

$$\phi(\mathbf{x}) = [1, \sqrt{2}x_1, \sqrt{2}x_2, x_1^2, x_2^2, \sqrt{2}x_1x_2]^T$$

So using this kernel is equivalent to working in a 6 dimensional feature space.

Linear Kernels

Deriving the feature vector implied by a kernel is in general quite difficult, and only possible if

the kernel is Mercer. However, deriving a kernel from a feature vector is easy: we just use

$$\kappa(\mathbf{x}, \mathbf{x}') = \phi(\mathbf{x})^T \phi(\mathbf{x}') = \langle \phi(\mathbf{x}), \phi(\mathbf{x}') \rangle$$

if $\phi(\mathbf{x}) = \mathbf{x}$, we get the **linear kernel** defined by

$$\kappa(\mathbf{x}, \mathbf{x}') = \mathbf{x}^T \mathbf{x}'$$

Linear Kernels

This is useful if the original data is already high dimensional, and if the original features are individually informative, e.g., a bag of words representation where the vocabulary size is large. Of course, not all high dimensional problems are linearly separable. For example, images are high dimensional, but individual pixels are not very informative, so image classification typically requires non-linear kernels.

The Kernel Trick

Rather than defining our feature vector in terms of kernels, $\phi(\mathbf{x}) \; = \; [\kappa(\mathbf{x}, \mathbf{x}_{\mathbf{1}}), \ldots, \kappa(\mathbf{x}, \mathbf{x}_{\boldsymbol{N}})]$, we can instead work with the original feature vectors $\mathbf{x}$, but modify the algorithm so that it replaces all inner products of the form $\langle \mathbf{x}, \mathbf{x}' \rangle$ with a call to the kernel function, $\kappa(\mathbf{x}, \mathbf{x}')$ This is called the **kernel trick**.

The kernel has to be a Mercer kernel for this to work.

Kernelized Nearest Neighborhood Classification

Recall that in a 1NN classifier, we just need to compute the Euclidean distance of a test vector to all the training points, find the closest one, and look up its label. This can be kernelized by observing that

$$||\mathbf{x}_i - \mathbf{x}_{i'}||_2^2 = \langle \mathbf{x}_i, \mathbf{x}_i \rangle + \langle \mathbf{x}_{i'}, \mathbf{x}_{i'} \rangle - 2\langle \mathbf{x}_i, \mathbf{x}_{i'} \rangle$$

This allows us to apply the nearest neighbor classifier to structured data objects.

Kernelized Ridge Regression

Let $\mathbf{x} \in \mathbb{R}^D$ be some feature vector, and $\mathbf{X}$ be the corresponding $N \times D$ design matrix. We want to minimize

$$J(\mathbf{w}) = (\mathbf{y} - \mathbf{X}\mathbf{w})^T(\mathbf{y} - \mathbf{X}\mathbf{w}) + \lambda||\mathbf{w}||^2$$

The optimal solution is given by

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X} + \lambda\mathbf{I}_D)^{-1}\mathbf{X}^T\mathbf{y} = (\sum_i \mathbf{x}_i\mathbf{x}_i^T + \lambda\mathbf{I}_D)^{-1}\mathbf{X}^T\mathbf{y}$$

Kernelized Ridge Regression

The optimal solution for **w** is not yet in the form of inner products. Using the matrix inversion lemma, we rewrite the ridge estimate as follows

$$\mathbf{w} = \mathbf{X}^T(\mathbf{X}\mathbf{X}^T + \lambda\mathbf{I}_N)^{-1}\mathbf{y}$$

which takes $O(N^3 + N^2D)$ time to compute. This can be advantageous if D is large.

We can partially kernelize the above, by replacing $\mathbf{XX}^T$ with the Gram matrix $\mathbf{K}$.

Kernelized Ridge Regression

Let us define the following **dual variables**:

$$\boldsymbol{\alpha} \triangleq (\mathbf{K} + \lambda \mathbf{I}_N)^{-1}\mathbf{y}$$

Then we can rewrite the **primal variables** as follows

$$\mathbf{w} = \mathbf{X}^T\boldsymbol{\alpha} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i$$

Kernelized Ridge Regression

This tells us that the solution vector is just a linear sum of the N training vectors. When we plug this in at test time to compute the predictive mean, we get

$$\hat{f}(\mathbf{x}) = \mathbf{w}^T\mathbf{x} = \sum_{i=1}^{N} \alpha_i \mathbf{x}_i^T \mathbf{x} = \sum_{i=1}^{N} \alpha_i \kappa(\mathbf{x}, \mathbf{x}_i)$$