CIS5200: Machine Learning Spring 2023

Lecture 9: Kernels

Date: February 9, 2023 *Author: Surbhi Goel*

Acknowledgements. These notes are heavily inspired by material from CIS5200 Spring 2022 and Cornell University's CS 4/5780 — Spring 2022.

Disclaimer. These notes have not been subjected to the usual scrutiny reserved for formal publications. If you notice any typos or errors, please reach out to the author.

1 Kernels

A common problem we keep stumbling upon in our lectures is: *there may not be a linear classifier or regressor that explains our model?* In this lecture, we will address this question (to some degree) by looking at more expressive functions.

1.1 Feature Maps

One way to make linear functions non-linear is by applying a non-linear feature map on the input features x. More formally, you can take your input $x \in \mathbb{R}^d$ and map to $\phi(x) \in \mathbb{R}^D$ where $\phi : \mathbb{R}^d \to \mathbb{R}^D$ is a feature map. Usually $D >> d$, sometimes even infinite dimensional.

For example, consider the example we looked at in Lecture 2 (see Figure 1). This is clearly not linearly separable. But consider the following feature map $\phi(x) = [x_1, x_2, x_1^2, x_2^2]^\top$. Now observe that under this feature map, we can linearly separate the data using $w = [0, 0, 1, 1]^\top$ and $b = -r^2$.

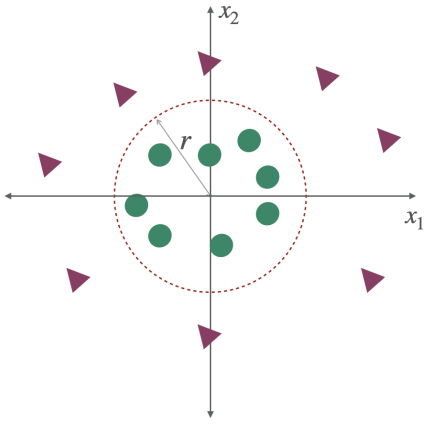

Figure 1: Example of non-linearly separable data in 2 dimensions.

Let us consider another common example, polynomial feature map of degree 2:

$$\phi(x) = \begin{bmatrix} 1 \\ x_1 \\ \vdots \\ x_d \\ x_1^2 \\ x_1 x_2 \\ \vdots \\ x_d^2 \end{bmatrix}.$$

This contains all monomials of degree at most 2. Imagine a similar map of degree r. The number of features grows as $D \approx d^r$. On one hand, it increases the ability for the model to express more complex functions, but on the other hand, it makes the algorithms slower since D can be prohibitively large.

In this lecture, we will look at the kernel trick that allows us to use large feature maps without paying heavy computational costs.

1.2 Recap: SVM Dual

Recall the dual of the soft-SVM problem from last lecture:

$$\begin{aligned} \text{maximize over } \alpha \qquad & -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j (x_i^\top x_j) + \sum_{i=1}^{m}\alpha_i \\ \text{such that} \qquad & \sum_{i=1}^{m}\alpha_i y_i = 0 \\ & \forall i \in [m], 0 \le \alpha_i \le C. \end{aligned}$$

If we wish to run this on the feature map ϕ, we can easily modify this to be:

$$\begin{aligned} \text{maximize over } \alpha \qquad & -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j (\phi(x_i)^\top \phi(x_j)) + \sum_{i=1}^{m}\alpha_i \\ \text{such that} \qquad & \sum_{i=1}^{m}\alpha_i y_i = 0 \\ & \forall i \in [m], 0 \le \alpha_i \le C. \end{aligned}$$

Observe that ϕ appears only as inner (dot) products: $\phi(x_i)^\top \phi(x_j)$ for $i, j \in [m]$. In fact, even to make a prediction, we need to compute

$$w^\top \phi(x) + b = \sum_{i=1}^{m}\alpha_i y_i \phi(x_i)^\top \phi(x) + b$$

which also depends on only inner products between feature vectors. Naively computing this inner product would require $O(D)$ time, but in some cases we can actually do this much faster.

Let us recall the polynomial feature map ϕ of degree 2. We have

$$\phi(x)^\top \phi(x') = 1 + \sum_{i \in [d]} x_i x'_i + \sum_{i,j \in [d]} x_i x_j x'_i x'_j = 1 + x^\top x' + (x^\top x')^2.$$

Observe that instead of computing the inner product between two vectors of dimension $D = (d+1)^2$, we only need inner products between two d-dimensional vectors. In fact, this extends to any degree r. This is known as the kernel trick! Here computing inner products can be done exponentially faster than actually computing them!

1.3 Kernel Function

A kernel is a function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ that satisfies:

$$k(x, x') = \langle \phi(x), \phi(x') \rangle$$

for some feature map ϕ that maps $\mathcal{X}$ to some inner product space $\mathcal{V}$. For given data $x_1, \ldots, x_m$, we can define the corresponding kernel matrix K such that $K_{ij} = k(x_i, x_j)$. For a kernel to be valid, K must be positive semi-definite (all eigenvalues are non-negative) and symmetric. We can directly think about kernels now and not worry about the feature map.

Let us walk through some examples of kernel functions:

- **Linear**: $k(x, x') = x^\top x'$. This is the same as using a linear classifier.
- **Polynomial**: $k(x, x') = (1 + x^\top x')^r$ for degree r. Note that this is slightly different from the kernel from our example and leads to a slightly different feature map, but is the more commonly used one.
- **Gaussian or radial basis function (RBF)**: s for parameter $\sigma > 0$. This kernel corresponds to a feature map that is infinite dimensional. This is the most popular kernel function, and a uniform approximator, that is, it can approximate any function.
- **Other**: There are kernels over strings, sets, trees, graphs, etc.

One way to think of a kernel is that it is a similarity map. Higher the value of the kernel function on two points, the more similar they are. *Check to see if this intuition holds for the above kernels.*

1.4 Kernel Machines

Now that we are equipped with the kernel trick, we can use the idea to "kernalize" algorithms. The key properties are that the algorithm accesses training points using inner products and the solution is a in the span of the training points. Let us look at some examples:

Kernel SVM. We can rewrite the optimization from Section 2 using the kernel k corresponding to the feature map ϕ,

$$\begin{aligned} \text{maximize over } \alpha \qquad & -\frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m}\alpha_i\alpha_j y_i y_j k(x_i,x_j) + \sum_{i=1}^{m}\alpha_i \\ \text{such that} \qquad & \sum_{i=1}^{m}\alpha_i y_i = 0 \\ & \forall i \in [m], 0 \le \alpha_i \le C. \end{aligned}$$

We can then make the prediction as

$$\text{sign}\left(w^\top \phi(x) + b\right) = \text{sign}\left(\sum_{i=1}^{m}\alpha_i y_i k(x_i,x) + b\right).$$

Note that b can also be computed using only inner products since for any support vector $i \in SV$,

$$b = y_i - w^\top \phi(x_i) = y_i - \sum_{j=1}^{m}\alpha_i y_i k(x_j,x_i).$$

Kernel Perceptron. Recall the Perceptron algorithm:

Algorithm 1: Perceptron

Initialize $w_1 = 0 \in \mathbb{R}^d$
for $t = 1, 2, \ldots$ **do**
  if $\exists i \in \{1,\ldots,m\}$ *s.t.* $y_i \neq \text{sign}\left(w_t^\top x_i\right)$ **then** update $w_{t+1} = w_t + y_i x_i$
  else output w_t
end

Note that the updates are always in the space spanned by $\{x_1,\ldots,x_m\}$ therefore we have that the optimal $w_* = \sum_{i=1}^{m}\alpha_i x_i$ for some $\alpha \in \mathbb{R}^m$. Using this, we can rewrite the algorithm in terms of α. *Work this out for yourself once more.*

Algorithm 2: Perceptron - Dual

Initialize $\alpha_1 = 0 \in \mathbb{R}^d$
for $t = 1, 2, \ldots$ **do**
  if $\exists i \in \{1,\ldots,m\}$ *s.t.* $y_i \neq \text{sign}\left(\sum_{j=1}^{m}\alpha_{tj}x_j^\top x_i\right)$ **then** update $\alpha_{(t+1)i} = \alpha_{ti} + y_i$
  else output α_t
end

Since this only depends on inner products, we can kernalize it straightforwardly.

Kernel Ridge Regression. Let us look at ℓ_2 regularized linear regression

$$\text{minimize over } w \qquad \frac{1}{m}\sum_{i=1}^{m}\left(y_i - w^\top x_i\right)^2 + \lambda\|w\|_2^2 = \frac{1}{m}\|Y - Xw\|_2^2 + \lambda\|w\|_2^2.$$

We want to show that an optimal w_* can be expressed as a linear combination of the input data points. Suppose this is not true, that is, there is an optimal $w_* = w + u$ where w belongs to the span of $x_1, \ldots, x_m$ and u lies in the orthogonal subspace. Then we have,

$$w_*^\top x_i = w^\top x_i + u^\top x_i = w^\top x_i.$$

The prediction remains unchanged, so $\|Y - Xw_*\|_2^2 = \|Y - Xw\|_2^2$. However, $\|w^*\|^2 = \|w\|^2 + \|u\|^2$ implying $\|w\|^2 < \|w_*\|^2$. This contradicts the fact that w_* is optimal. Therefore, we can assume $u = 0$. This gives us that

$$w_* = \sum_{i=1}^{m} \alpha_i x_i = X^\top \alpha.$$

Using this, we can modify our problem as:

$$\text{minimize over } \alpha \qquad \frac{1}{m}\|Y - XX^\top \alpha\|_2^2 + \lambda \alpha^\top XX^\top \alpha.$$

Observe that to kernalize, we can replace $XX^\top$ by K to get

$$\text{minimize over } \alpha \qquad \frac{1}{m}\|Y - K\alpha\|_2^2 + \lambda \alpha^\top K\alpha.$$

The optimal $\alpha = (K + \lambda mI)^{-1}Y$. To predict on input x, we can construct a vector $k_x = [k(x, x_1) \ldots k(x, x_m)]^\top$, and output $k_x^\top \alpha$. *Try to check why this is the correct predictor.*

2 Demo

Here's a wonderful demo made by folks at Cornell to try out various classification algorithms we have looked at (and some more) and test your understanding for each of them!