Kernel Methods

Andrés Ponce

May 24, 2020

Kernel methods allow us to find patterns in data, useful for classification and regression. These methods allow us to quickly and efficiently work with high dimensional data. We break down a function that might not be linear in nature to linear functions of its components. $k(x, y) = (f(x), g(y))$, where f and g are the dimensional inputs.

Dual representations

Parametric vs. non-parametric models

Parametric methods refer to when we train a model that is a function of $y(x, w)$, where we map x to y. In this scenario, the training data are thrown away after training.

On the other hand, **non-parametric methods** use a nearest-neighbor classifier strategy to perform classification. However, in this scheme we keep the training data.

Kernel methods evaluate a **kernel function** based on the training data. The dual part of this representation comes from the kernel function itself. We define a basic kernel function as: [1]

$$k(x, x') = \phi(x)^T \phi(x')$$

[1] Notice this function is **symmetric**, since $k(x, x') = k(x', x)$

Based on this formulation, we can then define vector a, which we use instead of the weight vector w. What a allows us to rephrase the optimal solution as a linear combination of the entire training data set. We define a: [2]

$$a_n = -\frac{1}{\lambda}\{w^T \phi(x_n) - t_n\}$$

[2] a is the result of setting the derivative of $J(w)$ to zero, where $J(w)$ is a sum of squares formultion

Then, we next define the **Kernel Matrix** $K = \Phi\Phi^T$, which makes K a symmetric matrix with each cell in the matrix:

$$K_{nm} = \phi(x_n)^T \phi(x_m) = k(x_n, x_m)$$

Finally, the solution again is of the form $y(x) = w^T \phi(x)$, however with the new concepts we've introduced we can write the final solution as

$$k(x)^T (K + \lambda I_N)^{-1} t$$

Final notes on Dual representations

If we have M dimensions in our data, with N data points, we will usually have more data points than dimensions in our data. This

might make dual representations less efficient and might lead to more sparse matrices in the end. However, since we don't take into account the feature set of our data $\phi(x)$, we can handle data with potentially infinite features, making it more flexible.

Constructing Kernels

How do we build a kernel to lead to more accurate classification? Maybe we can first find a feature space for its inputs and then check if there is a space where the output is equal to the inner product of the data points (i.e. $y(x) = k(f(x), g(x'))$. So when we multiply the inner products in some lower dimensional space, the result exists in a higher space where we are wanting to find the kernel anyways.[3]

Once we find a candidate function, how canwe show it's a proper kernel function? We can know its a valid kernel function by calculating its eigenvalues. If all its eigenvalues are positive, then the matrix K is **positive** and **semi-definite**.

Some types of kernel functions:

- **Polynomial kernel**: $k(x, x') = (x^T x' + c)^M$

  Similar to our basic kernel function,this one is symmetric.

- **Gaussian kernel**: $k(x, x') = \exp(-\|x - x'\|^2)$
- **Sigmoidal kernel**: $k(x, x') = \tanh(\alpha x^T x' + b)$ where a and b are constants. The Gram matrix of this function is usually not positive or semi-definite, however it can be useful in practice.

[3] Maybe this is why it's used in facial recognition? Becuase each pixel in an image can correpsond to a dimension in the data, we want to break it down to a combination of lower dimensions.

Support Vector Machines for classification

Maximum Margin classifier

Suppose we have N training data points and their corresponding labels t. SVMs use a normal-enough looking formula like

$$y(x) = w^T \phi(x) + b$$

where $\phi(x)$ denotes the fixed feature-space transformation.

If we have linearly separable classes, we can have multiple decision boundaries that give adequate solutions. Is there an optimal one, though? If so, how do we calculate that?

We want to maximize the **margin**, since a greater distance would mean that the data is split more cleanly along the decision boundary. Then the perpendicular distance between the decision boundary and the data point is given by

$$\frac{t_n y(x_n)}{\|w\|} = \frac{t_n (w^T \phi(x_n) + b)}{\|w\|}$$

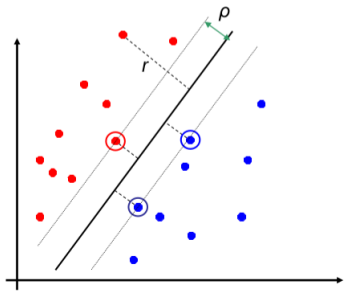

Figure 1: The margin is the smallest distance between the decision boundary and any of the training points.

which is the equation we want to maximize.

Here we can also use Lagrangian multipliers to find an optimization. The optimization can be done in $O(n^3)$.

According to the **KKT** conditions, we can have three conditions regarding SVMs:

1. $a_n \geq 0$
2. $t_n y(x_n) - 1 \geq 0$
3. $a_n\{t_n y(x_n) - 1\} = 0$

Due to the third contstraint above, either $a_n = 0$ or $t_n y(x_n) - 1 = 0$