November 8, 2017 Lecture 17, Nov. 1, 2017

Kernel Methods

Lecturer: Justin Domke *Scribe: Annamalai Natarajan*

1 Summary

In the last class we discussed representer theorem along with proof and interpretation. In this class we wrap up kernel methods with some take aways at the end of this lecture.

2 Representer Theorem Recap

Consider a dataset D of the form $\{(x_1, y_1), (x_2, y_2), ..., (x_n, y_n)\}$ of N examples. In this problem setup $x \in \mathbf{R}^p$ and $y \in \mathbf{R}$, where p is the number of features. If β minimizes the risk function,

$$\min_{\beta} \sum_{i=1}^{N} L(y_i, \beta^T h(x_i)) + \lambda \|\beta\|_2^2 \tag{2.1}$$

then there exists a vector $\alpha \in R^N$ such that

$$\beta = \sum_{j=1}^{N} \alpha_j h(x_j) \tag{2.2}$$

is the same as minimizing α,

$$\min_{\alpha} \sum_{i=1}^{N} L(y_i, \sum_{j=1}^{N} \alpha_j k(x_i, x_j)) + \lambda \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) \tag{2.3}$$

$$\tag{2.4}$$

where $\beta \in \mathbf{R}^M$ and $\alpha \in \mathbf{R}^N$. This can be compactly represented in matrix form as,

$$\min_{\alpha} \mathbf{L}(\mathbf{y}, \mathbf{K}\alpha) + \lambda \alpha^T \mathbf{K} \alpha \tag{2.5}$$

3 String Kernels

String kernel is a special kernel that can be applied to input string sequences to compute similarities. Consider an input string $x =$ 'aaba'. The basis expansion, $h(x)$, we are interested is to count n-grams *e.g., 3-grams*. For $x =$ 'aaba', $h(x)$ would look like,

```
aaa = 0
aab = 1
...
aba = 1
...
zzz = 0
```

Given 26 alphabets and space there are 27^n possible n-grams. Problems with strings are common in computational biology with genes expressed as strings of letters 'A','C','T','G' but extremely long sequences. String kernels are useful in these problems since we don't need to explicitly compute the basis expansions. In general a string kernel is expressed as,

$$K(x, x') = \sum_{u \in \Sigma^n} \#u(x)\#u(x') \tag{3.1}$$

where, Σ^n is a set of all possible strings of length n in the alphabet and $\#u(x)$ is simply the number of occurrences of u in x. For example given two input strings $x_1 =$ 'aab' and $x_2 =$ 'aaa', the string kernel computes the similarity as,

Example	h('aab')	h('aaa')
aa	1	2
ab	1	0
ba	0	0
bb	0	0

taking a dot product gives 2. The observation is that there are many possible substrings but only few have non-zero entries. One approach to compute $\mathbf{K}$ efficiently is to only count the substrings that exist between the two pairs of inputs and look for potential matches. The algorithm proceeds as follows,

1. Record all possible substrings of length n for x
2. Record all possible substrings of length n for x'
3. Match them up and sum

The time complexity of this algorithm to compute $K(x, x')$ is $length(x) + length(x')$ and it can be efficiently implemented using hash table data structure.

4 Mercer's Theorem

Up until now we have always assumed that we are given a kernel $\mathbf{K}$ which we evaluated on pairs of inputs under the assumption that $\exists h(x)$ subject to $K(x, x') = h(x)^T h(x')$. Can any similarity computing function qualify as a kernel? For a kernel to be valid it must be positive semi definite (PSD). A matrix is said to be PSD If for all vectors x, $x^T \mathbf{K} x \geq 0$. If $\mathbf{K}$ is not PSD then the equivalence between,

$$\lambda \|\beta\|_2^2 \Leftrightarrow \lambda \alpha^T \mathbf{K} \alpha \tag{4.1}$$

$$\tag{4.2}$$

does not hold since there exists an α such that $\lambda\alpha^T\mathbf{K}\alpha < 0$ which implies that $\|\beta\|_2^2 < 0$ which is not possible. Thus if $\mathbf{K}$ is not a PSD then $K(x, x')$ is not a valid kernel.

Mercer's theorem states that $k(x, x')$ is valid iff $\mathbf{K}$ is PSD for all training sets $(x_1, x_2, ..., x_n)$.

5 Kernel Methods for Classification

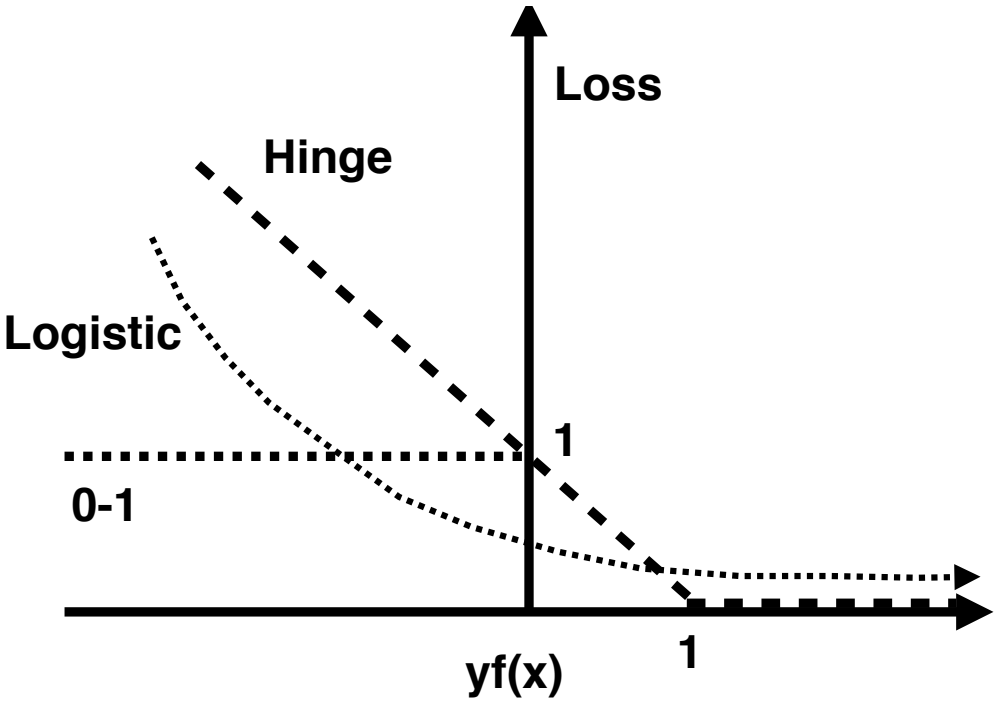

Figure 5.1: Loss functions for classification

Kernel methods can be used for classification as well by simply replacing the loss functions. Recall the loss function from regression is,

$$\min_{\beta} \sum_{i=1}^{N} L(y_i, \beta^T h(x_i)) + \lambda\|\beta\|_2^2 \tag{5.1}$$

Examples of loss function, L, include mean absolute error (MAE), mean squared error (MSE). The representer theorem allowed us to rewrite the above loss function in terms of α as,

$$\min_{\alpha} \sum_{i=1}^{N} L(y_i, \sum_{j=1}^{N} \alpha_j k(x_i, x_j)) + \lambda \sum_{i=1}^{N}\sum_{j=1}^{N} \alpha_i\alpha_j k(x_i, x_j) \tag{5.2}$$

where $\beta^T h(x) = \sum_{j=1}^{N} \alpha_j h(x_j)$. Replacing the above loss function with loss functions from classification results in kernel methods for classification. We are going to look at two loss functions: logistic loss and hinge loss. Both these loss functions are shown in Figure 5.1.

5.1 Kernel Logistic Regression

We first look at kernel methods with logistic loss, Replacing the generic loss function, L, with $L_{logistic}$ we get,

$$\min_{\alpha} \sum_{i=1}^{N} L_{logistic}(y_i, \sum_{j=1}^{N} \alpha_j k(x_i, x_j)) + \lambda \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) \tag{5.3}$$

$$L_{logistic}(y, f) = \log(1 + \exp(-yf)) \tag{5.4}$$

- Recall that the logistic loss is a continuous, smooth function for which we can compute the gradients with respect to α. Typically we find the best α by gradient descent algorithms.
- To make a prediction for a new data sample x as $f(x) = \sum_{j=1}^{N} \alpha_j k(x, x_j)$.

5.2 Support Vector Machines

Replacing the generic loss function, L, with hinge loss, L_{hinge}, we get,

$$\min_{\alpha} \sum_{i=1}^{N} L_{hinge}(y_i, \sum_{j=1}^{N} \alpha_j k(x_i, x_j)) + \lambda \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j k(x_i, x_j) \tag{5.5}$$

$$L_{hinge}(y, f) = \max(0, 1 - yf) \tag{5.6}$$

- Recall that the hinge loss only penalizes examples for which the product of $yf < 1$. It turns out to be the case that many $\alpha_j = 0$ since the associated data examples are further away from the separating hyperplane. Only a few data examples have non-zero α_j.
- To make a prediction for a new data point as,

$$f(x) = \sum_{j=1}^{N} \alpha_j k(x, x_j) \tag{5.7}$$

$$f(x) = \sum_{j \in SV}^{N} \alpha_j k(x, x_j) \tag{5.8}$$

- SV is a set of support vectors. These correspond to data samples that lie on either side of the separating hyperplane
- As the size of the support vector set increases training error rates also increase and vice versa

- For historical reasons α_i is represented as $\hat{\alpha_i} = \alpha_i y_i$. This results in the loss function written as

$$\min_{\alpha} \sum_{i=1}^{N} L_{hinge}(y_i, \sum_{j=1}^{N} y_i \hat{\alpha_j} k(x_i, x_j)) + \lambda \sum_{i=1}^{N} \sum_{j=1}^{N} y_i y_j \hat{\alpha_i} \hat{\alpha_j} k(x_i, x_j) \tag{5.9}$$

6 Kernel - Summary

1. Kernel methods are used to minimize the loss function,

$$\min_{\beta} \sum_{i=1}^{N} L(y_i, \beta^T h(x_i)) + \lambda \|\beta\|_2^2 \tag{6.1}$$

2. This problem setup leads to really long vectors $h(x)$ and β
3. **Representer Theorem:** For the best β,

$$\beta = \sum_{j=1}^{N} \alpha_j h(x_j) \tag{6.2}$$

we search for the best β in low dimensional subspace rather than search in searching in full feature space.

4. Minimize,

$$\min_{\alpha} \sum_{i=1}^{N} L(y_i, \sum_{j=1}^{N} \alpha_j h(x_i)^T h(x_j)) + \lambda \| \sum_{j=1}^{N} \alpha_j h(x_j) \|_2^2 \tag{6.3}$$

where spaces of α's is much less than β's. Therefore by using the representer theorem, we can instead minimize α's.

5. Kernel trick to compute dot product between $h(x_i)^T h(x_j)$ can be used to replace the dot product with the kernel,

$$\min_{\alpha} \sum_{i=1}^{N} L(y_i, \sum_{j=1}^{N} \alpha_j k(x_i, x_j)) + \lambda \sum_{i=1}^{N} \sum_{j=1}^{N} k(x_i, x_j) \alpha_i \alpha_j \tag{6.4}$$

6. Final prediction is computed as,

$$f(x) = \beta^T h(x) \tag{6.5}$$

$$= \sum_{j=1}^{N} \alpha_j h(x)^T h(x_j) \tag{6.6}$$

$$= \sum_{j=1}^{N} \alpha_j k(x, x_j) \tag{6.7}$$

Both equations 6.6 and 6.7 result in the same prediction but are implemented using different algorithms.

7. We have fast and interesting kernels *e.g., polynomial and string kernels*
8. Mercer's theorem states that $K(x, x')$ is valid iff $\mathbf{K}$ is PSD