CSE 475: Statistical Methods in AI **Monsoon 2019**

SMAI-M-2019 15: Nonlinear SVM and Kernels

Lecturer: C. V. Jawahar *Date: Sep 30, 2019*

15.93 Kernels and Feature Maps

We had seen the idea of dimensionality reduction in the past. We had seen PCA. We can also have many other dimensionality transformation techniques. The idea is that with an appropriate feature transformation the problem becomes "simpler" and the classifier/algorithm can do superior. Why don't we increase the dimension if that makes the problem simpler?

Consider a feature transformation (or feature map) as:

$$\mathbf{p} \rightarrow \phi(\mathbf{p}).$$

Let us start with a specific example: Let $\mathbf{p} = [p_1, p_2]^T$, and $\phi(\mathbf{p})$ be $[p_1^2, p_2^2, \sqrt{2}p_1p_2]$. You may wonder how this helps us or why we do this. Wait. We will see the utility as we move forward. Note the notations. Here the sample $\mathbf{p}$ has two features/dimensions p_1 and p_2.

A number of algorithms in machine learning use dot product as the basic operation. You already had seen $\mathbf{w}^T\mathbf{x}$. The beauty of dot product is that the result is scalar independent of the dimensionality of the vector. i.e., the dot product of two vectors in 2D and 3D will be still a scalar, independent of the dimension. Let us now compute the dot product of $\phi(\mathbf{p})$ and $\phi(\mathbf{q})$. Here $\mathbf{q}$ is also a 2D vector similar to $\mathbf{p}$. i.e., $\mathbf{q} = [q_1, q_2]^T$. We know that $\mathbf{p}^T\mathbf{q} = p_1q_1 + p_2q_2$. Let us compute the dot/scalar/inner product in the new feature space.

$$\begin{aligned}\phi(\mathbf{p})^T\phi(\mathbf{q}) &= [p_1^2, p_2^2, \sqrt{2}p_1p_2]^T[q_1^2, q_2^2, \sqrt{2}q_1q_2] \\ &= p_1^2q_1^2 + p_2^2q_2^2 + 2p_1p_2q_1q_2 \\ &= (p_1q_1 + p_2q_2)^2 \\ &= (\mathbf{p}^T\mathbf{q})^2 = \kappa(\mathbf{p}, \mathbf{q})\end{aligned}$$

What it says is something simple. If we want to compute the dot products of $\phi(\mathbf{p})$ and $\phi(\mathbf{q})$, what we need to do is only computing dot product of $\mathbf{p}$ and $\mathbf{q}$ and then square it. (indeed, that is true only for this specific $\phi()$) Note that if we compute $\phi(\mathbf{p})^T\phi(\mathbf{q})$ like this, we do not have to really compute $\phi()$ explicitly. That could be a huge advantage in many situations. Later today, we also would like to map $\mathbf{p}$ into infinite dimension with a $\phi()$. The advantage of not requiring to compute $\phi()$ will be a big advantage then.

Feature Map: Here, $\phi()$ is popularly called as a feature map.

Kernels: The function $\kappa()$ is a kernel function.

We saw the kernel function of $(\mathbf{p}^T\mathbf{q})^2$. This need not be square. It could be $(\mathbf{p}^T\mathbf{q})^d$. This is a polynomial kernel. With some effort we can write the $\phi()$ corresponding to this kernel. There are many other potential kernels. A kernel functions allows us to compute an inner/dot product in a new feature space.

Let us consider another $\phi()$ for the same $\mathbf{p} = [p_1, p_2]^T$. Let $\phi(\mathbf{p})$ as $[p_1^2, p_2^2, p_1p_2, p_2p_1]$. Here $\phi()$ maps from 2D to 4D. (instead of 2 to 3 as in the previous example).

$$\begin{aligned}\phi(\mathbf{p})^T\phi(\mathbf{q}) &= [p_1^2, p_2^2, p_1p_2, p_1p_2]^T[q_1^2, q_2^2, q_1q_2, q_2q_1] \\ &= p_1^2q_1^2 + p_2^2q_2^2 + p_1p_2q_1q_2 + p_1p_2q_1q_2 \\ &= (p_1q_1 + p_2q_2)^2 \\ &= (\mathbf{p}^T\mathbf{q})^2 = \kappa(\mathbf{p}, \mathbf{q})\end{aligned}$$

There can be many such kernels functions and feature maps. The above ones were only some examples. Does it mean that for any $\kappa(,)$ we have a corresponding $\phi()$? Can any function be a kernel function? There are many such curious questions for investigating later.

Q: What about a kernel like $(\mathbf{p}^T\mathbf{q})^2 + (\mathbf{p}^T\mathbf{q})^3$? What will be the corresponding feature map?

15.93.1 Motivation from Separability

Consider a 2D pattern of two concentric circles (figure missing). Inner circle is class 1 and outer circle from class 2. Consider a feature map of the form

$$\phi(\mathbf{p}) = \phi([p_1, p_2]^T) = [r, \theta]^T$$

where (r, θ) is the polar coordinates. It is simple to see that the concentric circles will become separable in the polar coordinates. (indeed only r could have been enough.)

We can also consider

$$\phi(\mathbf{p}) = \phi([p_1, p_2]^T) = [p_1^2, p_2^2]^T$$

What happens to the concentric circles?

The point to note is that a 'good' $\phi()$ is going to make our (classification) problem simpler (say linearly separable). There are more unanswered questions now. How do we find the useful $\phi()$ or $\kappa()$ for a new problem.

15.94 Kernel Matrix

For many problems we have N sample vectors and we need to compute all kernel values for all pairs. Popularly, a kernel matrix $\mathbf{K}$ with (i, j) th element as $\kappa(\mathbf{x}_i, \mathbf{x}_j)$.

Q: What are the properties of the kernel matrix? square? symmetric? what more?

15.95 Nonlinear/Kernel SVMs

The notion of Kernels is very nicely related to SVMs. One may wonder whether SVMs are designed for Kernels or Kernels are designed for SVMs!!.

Let us now come back to our SVM problem (dual).

$$\max \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j \tag{15.30}$$

$$\sum_{i=1}^{N} \alpha_i y_i = 0$$

with $\alpha_i \geq 0$. Solving this gives us a linear SVM.

Assume the input data $\{(\mathbf{x}_i, y_i)\}$ was mapped by $\phi()$, leading to $\{(\phi(\mathbf{x}_i), y_i)\}$. We can find a linear boundary in the new feature space. And the corresponding decision boundary in the original space is going to be a nonlinear one.

This makes the SVM problem (with appropriate constraints) as

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

or

$$\max_{\alpha} \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{j=1}^{N} \alpha_i \alpha_j y_i y_j \kappa(\mathbf{x}_i, \mathbf{x}_j) \tag{15.31}$$

Many practical solvers use the precomputed $N \times N$ kernel matrix. This avoids repeated computation of kernel functions. But this necessitates the need to store and manipulate a $N \times N$ matrix while solving. Not very nice for big data sets.

15.95.1 Training and Testing

When you solve this nonlinear SVM problem, we get α_i corresponding to the new nonlinear SVM. Or what we get is the nonlinear SVM in the original feature space. Typically the output of the training is a set of α_i (that are non zero).

At the test time, we only need to evaluate the sign of $\mathbf{w}^T \phi(\mathbf{x}) + b$. i.e.,

$$sign(\sum_{i=1}^{N} \alpha_i y_i \phi(\mathbf{x}_i)^T \phi(\mathbf{x}) + b) \tag{15.32}$$

In a kernel setting this simply becomes:

$$sign(\sum_{i=1}^{N} \alpha_i y_i \kappa(\mathbf{x}_i, \mathbf{x}) + b) \tag{15.33}$$

Please note that α_i is sparse and we need to sum this only over the support vectors. However, at the test time, we need to have all the support vectors with us and the number of kernel evaluations is related to the number of support vectors. This makes the nonlinear SVMs slower at run time.

In the case of linear SVM, it is possible to compute $\mathbf{w}$ upfront and make each of the testing simple in computation. (Indeed while training, we still need to solve for α_i.).

Will the support vectors (and the magnitude of α_i) change with the choice of kernels? Yes. If you change the kernel, you need to solve the problem again and obtain the new α_i.

15.95.2 Example

Consider an example of XOR problem with $(-1, -1), -1; (-1, +1), +1; (+1, -1), +1; (+1, +1), -1$
Clearly the given four samples are not linearly separable. Assume we use a kernel $\phi(\mathbf{p}) = p_1 p_2$, the data becomes immediately separable. $+1, -1; -1, +1; -1, +1; +1 - 1$.

A quadratic kernel like $\kappa(\mathbf{p}, \mathbf{q}) = (\mathbf{p}^T \mathbf{q}))^2$ or $(\mathbf{p}^T \mathbf{q}+1)^2$ will have a product term and this can make the XOR problem separable. (remember the example $\phi()$ we had at the beginning.)

15.96 Popular Kernels

We appreciate

- the utility of feature maps and kernels in"simplifying" the problem (eg. making the problem linearly separable).

- the fact that we do not have to evaluate the feature map $\phi()$ in practice. A small kernel computation in the original space is equivalent to the inner product in the feature space. An important point to note is that when we use kernels, we do not evaluate $\phi()$ or even do not have to know the explicit form of $\phi()$ itself. (Knowing $\phi()$ is still required for the exams!!)

The feature maps and the kernels that we saw is a toy kernel in 2D. We saw the $\phi()$ mapping frm 2D to 3D or 4D. Why not $\phi()$ mapping to an infinite dimensional space? Many popular kernels do that.!!

Many of the popular kernel functions

Linear Kernel	: $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})$
Polynomial Kernel	: $K(\mathbf{x}, \mathbf{y}) = (\mathbf{x} \cdot \mathbf{y})^d$
Radial Basis Kernel (RBF)	: $K(\mathbf{x}, \mathbf{y}) = e^{-\gamma \|\mathbf{x}-\mathbf{y}\|^d}$
Sigmoid Kernel	: $K(\mathbf{x}, \mathbf{y}) = tanh(\gamma(\mathbf{x} \cdot \mathbf{y}))$

We may not know the "right" kernel for a problem in our hand always. For example a simple product term worked for XOR. But a polynomial kernel, which has such a term is what we may use in practice. In practice we try multiple popular kernels (i.e., the ones that are in your library!!) and pick the one that gives us best results. RBF kernel is often preferred for many problems.

15.96.1 Kernels for more structured data

We can think of kernels as "similarity functions" that can be plugged into the machine learning algorithm. We had seen kernels for real vectors. In many problems the inputs could be objects like (i) strings, (ii) trees or (iii) graphs. Can kernels be used in such situations?

One can define kernel over strings, trees, graphs etc. There are many such kernels available in the literature. This makes it possible to treat objects like graphs the same way we do the vectors in vector spaces.

Now we can directly work on strings in SVMs. An SVM can train and test Strings (not just vectors) given that the appropriate kernels are used. This also makes the SVM more general and useful.

15.97 Discussions

15.97.1 What is a valid kernel?

We know a number of functions like $(\mathbf{p}^T\mathbf{q})^d$ and $(\mathbf{p}^T\mathbf{q} + 1)^d$ as polynomial kernels.

We also saw a number of exponential kernels. Many of them also correspond to infinite dimensional feature maps. Remember we do not have to find the feature maps always.

Will every kernels function have a corresponding feature map? or what is a valid kernel?

When Kernel matrix K is PSD, the corresponding kernel is valid. This comes from the Mercer's theorem.

15.98 Beyond this course

15.98.1 What more? (*)

Kernels is an exciting topic. There has been extensive use of the "kernel trick" in converting a wide range of linear algorithms into nonlinear form. This takes advantage of the numerical stability of linear algorithms and expressive power of nonlinearity. This resulted in algorithms like Kernel Perceptron, Kernel PCA, Kernel LDA etc. Here is an additional reading. (not an original exposition; not very recent)

- https://www.dropbox.com/s/qryziuo3u143q5e/KERNEL-REVIEW.pdf?dl=0

Urge you to do a quick reading/glance to get better idea of this space.

15.98.2 Kernels for Structured Data (*)

Kernels can be defined for many structured data like strings, trees and graphs. This area has many interesting ideas. Here is a quick read:

`https://people.eecs.berkeley.edu/~jordan/kernels/0521813972c11_p344-396.pdf`

15.98.3 Can Kernels be learned? (*)

A natural question with which we end this lecture is "how do I find the right kernel for my problem?". In practice, most people do not worry about this problem and use a "powerful" kernel like RBF kernel and move on.

But there are many attempts to find the right kernel for a problem. Assume we define the optimal kernel as a linear combination of a number of base kernels, we can define this problem as that of learning the coefficients (say β_i) of this linear combination.

$$\kappa(\cdot, \cdot) = \sum_{i=1}^{P} \beta_i \kappa_i(\cdot, \cdot)$$

There have been some nice algorithms and attempts in such learning, popularly known as Multiple Kernel Learning.