CS 536 : Non-Linear Support Vector Machines

16:198:536

In the previous set of notes on SVMs, we considered the question of finding not only a linear separator for binary class data, but finding the 'best' separator - the one that maximizes the distance between the two classes, and thus hopefully provides better generality than any other linear separator. Two questions that remained at this point were:

- Can this classifier be found efficiently?
- What if the data is not linearly-separable?

To answer the first, the mechanics of linear and quadratic programming can be drawn on to provide computationally efficient solutions to the problem. To answer the second, the proposed solution was to consider 'soft margin' classifiers, effectively classifiers that allowed misclassified points, but attempted to minimize the overall misclassification on the training set. This is a reasonable approach when the data might be 'mostly' linearly separable, and noise could be pushing some data points across the boundary, etc. But in the case of highly non-linear data, for instance the **XOR** data or data where the positive class lies in a circle and the negative class lies outside a circle - these are situations that can't really be approximated with linear separators even allowing some mistakes. How can we apply the mechanics of support vector machines in these cases?

The first observation is that while linear separators on the data may not exist, non-linear separators will. Consider the **XOR** data:

$$\begin{aligned} A &: (-1,+1),+1 \\ B &: (-1,-1),-1 \\ C &: (+1,-1),+1 \\ D &: (+1,+1),-1. \end{aligned} \tag{1}$$

For each data point (x_1, x_2), consider the function $-x_1 * x_2$. In this case, we get $A \rightarrow -(-1)(+1) = 1, B \rightarrow -(-1)(-1) = -1$, etc, and the data can be correctly classified with $\text{sign}(-x_1 x_2)$.

Similarly, consider data that is distributed such that anything within distance 1 of the origin is of class $+1$, and anything greater than distance 1 of the origin is of class -1. In this case, there is no linear separator, but the data can be classified correctly with $\text{sign}(1 - x_1^2 - x_2^2)$.

In each of these cases, while there is no linear separator on the 'raw' data (x_1, x_2), there is a linear separator on higher dimensional 'features', in particular the quadratic features $x_1^2, x_2^2, x_1 x_2$. If we imagined transforming the data from the original data space to this quadratic feature space,

$$\phi(x_1, x_2) = \left(1, x_1, x_2, x_1 x_2, x_1^2, x_2^2\right), \tag{2}$$

while there is no linear separator on the raw data, there is a linear separator in both of these cases when the data is mapped into the new feature space under ϕ. This is an instance of the following result that, while true, is not always useful.

> Given any data set of binary class data $\{(\underline{x}^i, y^i)\}_{i=1,\ldots m}$, there is a linear separator for this data in **some** higher dimensional non-linear embedding.

Extending the SVM architecture to non-linearly separable data is then, to some extent, as simple as finding a useful higher dimensional embedding, and then running the SVM solver on the $\phi(\underline{x}^i)$ instead of the raw $\underline{x}^i$. But some questions remain at this point:

- Where does this feature space embedding ϕ come from?
- What effect does this have on the complexity of the solution?
- What does the resulting non-linear classifier 'look like' in the original data space? What is the structure of the solution?

To some extent, all three of these questions are addressed with the classical 'kernelization' trick.

1 Kernelization

We would like to find some kind of high dimensional embedding $\phi : \mathbb{R}^k \to \mathbb{R}^N$, where the data set may not have a linear separator in the original k-dimensional space but does have one in the larger N-dimensional space. It is worth starting with the following question though - if we had one, what would we do with it? One problem to note is that if the embedding space is quite high dimensional, this introduces some additional computational complexity into the analysis of the problem.

Note that the Primal SVM problem requires solving for a weight vector with a component corresponding to each of the data dimensions. The 'raw' primal problem is therefore trying to solve for $k+1$ variables, using m-many linear inequality constraints. The raw dual problem, however, is trying to solve for m-variables (one for each data point), using only 1 linear equality constraint (classically, the assumption that all the variables must be non-negative is not a serious one). Notice then that the dual problem is in some ways *independent* of the ambient data dimension.

Why does this matter? If we imagine trying to solve the primal SVM on the embedded ϕ-data, we are now looking for a weight vector in N dimensions - potentially quite large compared to k. But for the dual problem on the ϕ-data, we are still only looking for m-variables, one for each data point. If the embedded dimension N is much larger than the number of data points (as can happen), this suggests that the dual problem doesn't necessarily suffer from any added complexity from operating in this higher dimension. Given ϕ, we therefore want to solve

$$\begin{aligned} &\max_{\underline{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{j=1}^{m} \alpha_i y^i \phi(\underline{x}^i).\phi(\underline{x}^j) y^j \alpha_j, \\ &\text{(s.t.)} \quad \sum_{i=1}^{m} \alpha_i y^i = 0 \\ &\qquad \forall i : \alpha_i \geq 0. \end{aligned} \tag{3}$$

We can recover the weight vector $\underline{w} = \sum_i \alpha_i y^i \phi(\underline{x}^i)$ - and the previous structure results on support vectors apply, the boundary is really only defined by the vectors $\phi(\underline{x}^i)$ which are closest to the separator - recovered in this case when $\alpha_i > 0$. We can then define the classifier in terms of

$$\text{sign}(\underline{w}.\phi(\underline{x}) + b) = \text{sign}\left(\sum_i \alpha_i y^i \phi(\underline{x}^i).\phi(\underline{x}) + b\right). \tag{4}$$

This allows us to control somewhat the dimensionality of the problem and still utilize the established SVM mechanics. But additional computation is required still, in the sense that for each data point $\underline{x}$ we need to compute $\phi(\underline{x})$, and additionally we need to compute all the relevant dot products. If ϕ maps into a very high dimensional space, computing these dot products could potentially be quite expensive.

We are saved somewhat here, however, in that it is often the case that computing these high dimensional dot products may actually be quite easy. Consider, for instance, the embedding above of

$$\phi(x_1, x_2) = (1, x_1, x_2, x_1x_2, x_1^2, x_2^2). \tag{5}$$

In this case, we are going from 2-dimensions to 6-dimensions. If we were trying to map some original k-dimensional data space to a feature space that captured all quadratic and linear terms, there will be 1 constant term, k linear terms, and k^2 quadratic terms, so $N = O(k^2)$, so that computing all the relevant dot products should be something like $O(m^2k^2)$. However, consider the slightly modified embedding:

$$\phi(x_1, x_2) = (1, \sqrt{2}x_1, \sqrt{2}x_2, \sqrt{2}x_1x_2, x_1^2, x_2^2). \tag{6}$$

In this case, the dot product simplifies in the following way:

$$\begin{aligned}
\phi(x, y).\phi(a, b) &= (1, \sqrt{2}x, \sqrt{2}y, \sqrt{2}xy, x^2, y^2).(1, \sqrt{2}a, \sqrt{2}b, \sqrt{2}ab, a^2, b^2) \\
&= 1 + 2xa + 2yb + 2xyab + x^2a^2 + y^2b^2 \\
&= 1 + 2xa + 2yb + (xa + yb)^2 \\
&= (1 + (xa + yb))^2 \\
&= (1 + (x, y).(a, b))^2,
\end{aligned} \tag{7}$$

i.e., $\phi(\underline{x}).\phi(\underline{a}) = (1 + \underline{x}.\underline{a})^2$. In this case, computing the dot product really requires only two more operations than computing the dot product in the original space - rather than a $O(k^2)$ operation (in this quadratic feature space), an $O(k)$ operation over all. We see that embedding in this high dimensional feature space then really adds little to no computational overhead on top of the original SVM framework then (when looking at the dual problem).

But how to find embeddings ϕ with this property - that computing dot products in this space is 'nice'? This is on top of the original problem - how to find ϕ where the resulting feature space is nice and admits a linear separator.

One thing worth noting here is that to some extent, the actual ϕ does not really matter. It matters in terms of identifying what the features are in this new space and what features are relevant to the classification problem - but in terms of computing and executing a solution, *the only place the data or ϕ enters into the picture is through the form of the dot product.* Consider defining a 'kernel function' K by

$$K(\underline{x}, \underline{y}) = \phi(\underline{x}).\phi(\underline{y}). \tag{8}$$

We can restate the entire SVM problem purely in terms of this kernel function:

$$\begin{aligned}
&\max_{\underline{\alpha}} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i y^i K(\underline{x}^i, \underline{x}^j) y^j \alpha_j, \\
&\text{(s.t.)} \quad \sum_{i=1}^{m} \alpha_i y^i = 0 \\
&\qquad \forall i : \alpha_i \geq 0,
\end{aligned} \tag{9}$$

$$\text{classifier}(\underline{x}) = \text{sign}\left(\sum_i \alpha_i y^i K(\underline{x}^i, \underline{x}) + b \right). \tag{10}$$

Note: the soft margin classifiers previously discussed can be kernelized as well!

Taking this definition - we might even have an underlying embedding ϕ that is infinite dimensional! As long as the kernel is easy to compute, the dual is easy to solve, and the classifier easy to implement. The separator or boundary between the two classes is then defined implicitly by the equation

$$\sum_i \alpha_i y^i K(\underline{x}^i, \underline{x}) + b = 0. \tag{11}$$

Some common choices for kernels are the following:

- **Polynomial Kernels:** Given c, and some dimension d, define

$$K(\underline{x}, \underline{y}) = (c + \underline{x}.\underline{y})^d. \tag{12}$$

The above examples look at the case of $d = 2, c = 1$ as corresponding to a quadratic feature space embedding. In general, for a given d, this kernel corresponds to a feature space embedding that looks at all the order-d combinations of the components of the data. This is frequently very useful when dealing with things like boolean or categorical data, where the components of $\underline{x}$ are zero or one depending on whether or not the data point belongs to a given category - in which case order-d products and recombinations begin to represent all the logical combinations of which groups of categories the $\underline{x}$ belongs to.

- **Radial Basis Functions:** Given some parameter $\sigma > 0$, the *Gaussian Kernel* is defined as

$$K(\underline{x}, \underline{y}) = \exp\left(-\frac{1}{\sigma^2}||\underline{x} - \underline{y}||^2\right). \tag{13}$$

Note that this is an example of K as a sort of similarity function - the more similar $\underline{x}$ and $\underline{y}$ are, the larger the value of the kernel. The hyperparameter σ^2 controls the bandwidth of the kernel, or how close two data points have to be in order to count as 'similar'. This kernel is frequently useful and a good starting point for a lot of real valued data analysis; it corresponds to an infinite dimensional feature space which can be hard to visualize.

There are a variety of other choices that can occur for the kernel function - frequently the best bet may be to try to choose one that is particularly relevant to the specific problem you are trying to solve; the above represent generally useful choices that may or may not be relevant. One idea that it useful here is the notion of K as a sort of similarity metric, looking at how similar $\underline{x}$ and $\underline{y}$ are. One non-analytic version of this, $\underline{x}$ and $\underline{y}$ could represent strings, and K could represent the number of common substrings that $\underline{x}$ and $\underline{y}$ share - the more similar they are, the more substrings they share, and the higher value of K.

1.1 What makes a Valid Kernel?

Can we just throw any kernel at the SVM problem, and get a reasonable answer? Not all functions can be valid kernels - that is, not all functions K in this way are going to correspond to some $K(\underline{x}, \underline{y}) = \phi(\underline{x}).\phi(\underline{y})$ for some embedding ϕ in some high dimensional feature space. We can say a couple of things about how valid K should behave though - for instance, if K is valid (i.e., corresponds to a dot product of *some* feature embedding), it should be symmetric, i.e., $K(\underline{x}, \underline{y}) = K(\underline{y}, \underline{x})$. Additionally, it should be positive definite. We would have

$$K(\underline{x}, \underline{x}) = \phi(\underline{x}).\phi(\underline{x}) = ||\phi(\underline{x})||^2, \tag{14}$$

which will always be non-negative, and only zero if $\phi(\underline{x}) = 0$. Additionally, applying the properties of dot products, consider defining

$$\underline{v} = \sum_{i=1}^{n} \beta_i \phi(\underline{x}^i). \tag{15}$$

In this case, $\underline{v}.\underline{v} \geq 0$, and hence,

$$0 \leq \underline{v}.\underline{v} = \left(\sum_{i=1}^{n} \beta_i \phi(\underline{x}^i)\right) . \left(\sum_{i=1}^{n} \beta_i \phi(\underline{x}^i)\right) = \sum_{i=1}^{n} \beta_i \phi(\underline{x}^i) . \left(\sum_{j=1}^{n} \beta_j \phi(\underline{x}^j)\right) = \sum_{i=1}^{n}\sum_{j=1}^{n} \beta_i \beta_j \phi(\underline{x}^i).\phi(\underline{x}^j), \tag{16}$$

or

$$\sum_{i=1}^{n}\sum_{j=1}^{n} \beta_i \beta_j K(\underline{x}^i, \underline{x}^j) \geq 0. \tag{17}$$

And this needs to hold for *all* possible $\{\underline{x}^i\}$ and all possible $\{\beta_i\}$. And in fact, this is sufficient:

> Suppose that $K(\underline{x}, \underline{x}')$ satisfies that for any choice of $\{\underline{x}^1, \ldots, \underline{x}^n\}$ and any $\{\beta_1, \ldots, \beta_n\} \subset \mathbb{R}$, the following holds:
>
> $$\sum_{i=1}^{n}\sum_{j=1}^{n} \beta_i \beta_j K(\underline{x}^i, \underline{x}^j) \geq 0. \tag{18}$$
>
> In this case, the kernel map K corresponds to the inner product of some (non-unique) feature embedding ϕ in some unique feature space. *This is an extension of the Moore-Aronszajn Theorem.*

This result has some deeper analysis behind it (representation of linear functionals on Hilbert spaces), but the significance of it can be seen in part in that if the condition did not hold, it would be possible to construct data sets $\{\underline{x}^i\}$ and corresponding $\{y^i\}$ such that the quadratic part of the objective function in the dual SVM is unbound, and therefore no solution would exist. Valid kernels are ones for which the solution to the dual SVM problem is always bounded.

It is worth noting here again that while a kernel may be valid (correspond to *some* embedding in some feature space), it can be potentially difficult to work out exactly what that feature space is. But knowing the feature space embedding is not actually necessary to solving the SVM.

2 Data Space vs Feature Space

One way to interpret the above mechanics is that we are exchanging the 'raw' data space for a higher level space that expresses more complicated statistics and features of the data. But at least for 'classical' kernel functions above, we are essentially looking at pre-built feature spaces, features that may or may not be relevant to the problem as we are interested in it - feature spaces that are effectively chosen independent of the data itself.

As a preview of what is to come - neural networks, etc - it is worth considering what 'data-defined' features might look like. In two dimensions, a feature map might be expressed as

$$(x_1, x_2) \mapsto (1, g_1(x_1, x_2), g_2(x_1, x_2)), \tag{19}$$

and then the goal would be to learn a separator of the form

$$f(x_1, x_2) = \text{sign}(b + w_1 g_1(x_1, x_2) + w_2 g_2(x_1, x_2)). \tag{20}$$

However, there is nothing stopping the feature functions g_1, g_2 from *also* being functions that could be learned. For instance,

$$g_i(x_1, x_2) = \text{sign}(b_i + w_1^i x_1 + w_2^i x_2), \tag{21}$$

where b_i, w_1^i, w_2^i are also constants that need to be learned. As an example of this, consider again the **XOR** data:

$$\begin{aligned} A &: (-1, +1), +1 \\ B &: (-1, -1), -1 \\ C &: (+1, -1), +1 \\ D &: (+1, +1), -1. \end{aligned} \tag{22}$$

No linear separator exists generally, but consider the two feature maps defined by:

$$\begin{aligned} g_1(x_1, x_2) &= \text{sign}(x_2 - x_1 - 1) \\ g_2(x_1, x_2) &= \text{sign}(x_2 - x_1 + 1) \end{aligned} \tag{23}$$

Applying this here:

$$\begin{aligned}
A' &= (g_1(A), g_2(A)) = \text{sign}((+1) - (-1) - 1), \text{sign}((+1) - (-1) + 1) = \text{sign}(1), \text{sign}(3) = (1, 1) \\
B' &= (g_1(B), g_2(B)) = \text{sign}((-1) - (-1) - 1), \text{sign}((-1) - (-1) + 1) = \text{sign}(-1), \text{sign}(1) = (-1, 1) \\
C' &= (g_1(C), g_2(C)) = \text{sign}((-1) - (+1) - 1), \text{sign}((-1) - (+1) + 1) = \text{sign}(-3), \text{sign}(-1) = (-1, -1) \\
D' &= (g_1(D), g_2(D)) = \text{sign}((+1) - (+1) - 1), \text{sign}((+1) - (+1) + 1) = \text{sign}(-1), \text{sign}(1) = (-1, 1)
\end{aligned} \tag{24}$$

This mapped data, in the (g_1, g_2) feature space, is now linearly separable while it wasn't previously. Note that we have

$$\begin{aligned}
g_1(A) - g_2(A) &= 0 \\
g_1(B) - g_2(B) &= -2 \\
g_1(C) - g_2(C) &= 0 \\
g_1(D) - g_2(D) &= -2,
\end{aligned} \tag{25}$$

and hence we can successfully classify the data using the following separator:

$$f(x_1, x_2) = \text{sign}(g_1(x_1, x_2) - g_2(x_1, x_2) + 1). \tag{26}$$

In the future, we'll construct the feature maps effectively automatically, trained and informed on the data itself.