QUICK SUMMARY OF KERNEL METHODS

BEN RUBINSTEIN - 2014-09-19

- Consider n training points each with d features
- The SVM is fundamentally a linear classifier. It selects the max-margin classifier by solving a "primal program" that picks an optimal $w^\star$. The complexity of the primal solution is roughly $O(d^3)$
- To learn a non-linear classifier, we run the SVM in a high-dimensional feature space. We can map each training point x_i to feature space $\phi(x_i)$ which might have $f \gg d$ features. For example, we could take $\phi(x) = (x_1, \ldots, x_d, x_1^2, x_1x_2, \ldots, x_{d-1}x_d)$ with $O(d^2)$ features. To compute the primal solution to the SVM we now must transform the $n \times d$ training matrix to a $n \times f$ matrix which requires $O(nf)$ work (at least). For the previous example this would be $O(nd^2)$ work. Next we actually train which is $O(f^3)$ work, or for the example a whopping $O(d^6)$ work!
- We have an alternative way to solve the SVM, called the dual program that gives the same solution as the primal. There are two useful facts about the dual approach: the first is the complexity is $O(n^3)$ which is neither better nor worse than the primal, but depends on whether $n > f$ or not.
- The second fact is the dual approach doesn't need the features of the data directly, the $\phi(x_i)$, but only needs the dot product $\phi(x_i) \cdot \phi(x_j)$ for each i, j. We call this dot product of two feature vectors the kernel function and define $k(u, v) = \phi(u) \cdot \phi(v)$ and the $n \times n$ matrix made up of $k(x_i, x_j)$ the kernel matrix. The dual needs this matrix. To compute this matrix we need to compute $k(x_i, x_j)$ a total of $O(n^2)$ times so roughly $O(fn^2)$ work to create the matrix, before we can train which takes $O(n^3)$ work.
  - A very cool idea is that something called the Representer Theorem tells us that this will be the case, without even going through all the maths of coming up with the equivalent dual program. It can be used to study other new learning algorithms, and to tell whether they are kernel methods
- How should you interpret the kernel matrix? Well dot products are a kind of similarity measure: if you take unit length vectors u, v then $u \cdot v$ corresponds to the cosine of the angle between u, v and clearly the angle is a measure of similarity. The kernel is just the dot product in feature space, so it measures similarity of the feature-mapped examples
- It kind of sucks that you have to compute the kernel matrix, am I right? The good news is that in many cases you don't have to compute it directly by first mapping by ϕ then taking dot product. In many cases you can use the 'kernel trick' which lets you compute the dot product in feature space in terms of the dot product in input space. For example the above feature mapping essentially gives you the degree two polynomial which can be computed in terms of the input space dot product: $(1 + u \cdot v)^2$which is only $O(d)$ work. The cool thing is, computing dot product in feature space naively for degree p polynomial kernel is $O(d^p)$ but we can do it indirectly as $(1 + u \cdot v)^p$which still costs only $O(d)$. Computing the kernel matrix is then just $O(n^2d)$ no matter what p is. For our $p = 2$ example, all up primal costs about $O(nd^2 + d^6)$ and dual costs about $O(n^2d + n^3)$ - which is better depends on n vs d.
- So are feature mappings and kernels "the same"? Not quite: design a feature mapping and it is easy to compute the kernel as just dot products - and you could be lucky and take a short-cut via the kernel trick. For many kernels you can write down the feature mapping, although it may not be easy. However for some kernels you can't even write the feature mapping down - for example the RBF kernel $k(u, v) = \exp(-\|u - v\|^2/\sigma)$ has no explicit feature mapping (its feature mapping can be proved to be infinite dimensional in fact $f = \infty$). So, sometimes it may be easy to think about kernels and if so you don't need the feature mapping necessarily as you can use the dual which only requires the kernel. But if you want, you can design feature mappings and use those with the primal or dual.
  - One final thing: if you have a $k(u, v)$ that you'd like to use for your kernel similarity, the so-called Mercer's Theorem can be used to tell you if you have a "legal" kernel function that can be used with the SVM. This is great news, as it means you can go ahead and design just kernels and not feature mappings (if you want) and just use a simple theorem to check before inputting into the SVM
- These ideas hold true for pretty much all kernel methods. The Representer Theorem can tell you whether you have a learning algorithm that is a kernel method and can be made non-linear by kernelisation. So you can design lots of different kernel learning algorithms. On the other hand, Mercer's Theorem let's you design any kernel you like. You can usually solve kernel learning algorithms via either the primal or dual and which you choose depends on the various size parameters of your problem

There are lots of details of SVM not discussed above, but these are certainly some of the main ones. This summary should help better understand some of the key points, making the lecture slides and recording much easier to follow!