SVMs in Practice

* The SVM algorithm poses a particular optimization problem, + we use some software to solve for SVM parameters Ө
* Even when using some software for SVM, , there are a few things *you* need to do
* 1st: Come up w/ some parameter C
* 2nd: Choose the kernel/similarity function you want to use.
* 1 choice might be to not use any kernel = a **linear kernel**.
* “I used an SVM w/ a linear kernel” = used an SVM w/out a kernel + just used Ө(t)\*X = Ө0 + Ө1x1 + Ө2x2 … + ӨnXn, which predicts 1 if Ө(t)\*X > 0
* Think of “linear kernel” as a version of SVM that just gives you a standard linear classifier.
* Why would you want to do this?
* If you have a large number of features, n, + the number of training examples, m, is small (have a small training set), maybe you want to just fit a non-complicated linear function/decision boundary b/c you might not have enough data
* Also might risk overfitting if trying to fit a very complicated function in a very high-dimensional feature space w/ a small training set sample
* A 2nd choice for the kernel = **Gaussian kernel**
* + if you do this, then the other choice you need to make is to choose this parameter sigma squared when we also talk a little bit about the bias variance tradeoffs
* 3:30
* of how, if sigma squared is large, then you tend to have a higher bias, lower variance classifier, but if sigma squared is small, then you have a higher variance, lower bias classifier.
* 3:43
* So when would you choose a Gaussian kernel? Well, if your omission of features X, I mean Rn, + if N is small, +, ideally, you know,
* 3:55
* if n is large, right,
* 3:58
* so that's if, you know, we have say, a two-dimensional training set, like the example I drew earlier. So n is equal to 2, but we have a pretty large training set. So, you know, I've drawn in a fairly large number of training examples, then maybe you want to use a kernel to fit a more complex nonlinear decision boundary, + the Gaussian kernel would be a fine way to do this. I'll say more towards the end of the video, a little bit more about when you might choose a linear kernel, a Gaussian kernel + so on.
* 4:27
* But if concretely, if you decide to use a Gaussian kernel, then here's what you need to do.
* 4:35
* Depending on what SVM software package you use, it may ask you to implement a kernel function, or to implement the similarity function.
* 4:45
* So if you're using an octave or MATLAB implementation of an SVM, it may ask you to provide a function to compute a particular feature of the kernel. So this is really computing f subscript i for one particular value of i, where
* 5:00
* f here is just a single real number, so maybe I should move this better written f(i), but what you need to do is to write a kernel function that takes this input, you know,
* 5:10
* a training example or a test example whatever it takes in some vector X + takes as input one of the landmarks + but only I've come down X1 + X2 here, because the landmarks are really training examples as well. But what you need to do is write software that takes this input, you know, X1, X2 + computes this sort of similarity function between them + return a real number.
* 5:36
* + so what some SVM packages do is expect you to provide this kernel function that take this input you know, X1, X2 + returns a real number.
* 5:45
* + then it will take it from there + it will automatically generate all the features, + so automatically take X + map it to f1, f2, down to f(m) using this function that you write, + generate all the features + train the SVM from there. But sometimes you do need to provide this function yourself. Other if you are using the Gaussian kernel, some SVM implementations will also include the Gaussian kernel + a few other kernels as well, since the Gaussian kernel is probably the most common kernel.
* 6:14
* Gaussian + linear kernels are really the two most popular kernels by far. Just one implementational note. If you have features of very different scales, it is important
* 6:24
* to perform feature scaling before using the Gaussian kernel. + here's why. If you imagine the computing the norm between X + l, right, so this term here, + the numerator term over there.
* 6:38
* What this is doing, the norm between X + l, that's really saying, you know, let's compute the vector V, which is equal to X minus l. + then let's compute the norm does vector V, which is the difference between X. So the norm of V is really
* 6:53
* equal to V1 squared plus V2 squared plus dot dot dot, plus Vn squared. Because here X is in Rn, or Rn plus 1, but I'm going to ignore, you know, X0.
* 7:06
* So, let's pretend X is an Rn, square on the left side is what makes this correct. So this is equal to that, right?
* 7:17
* + so written differently, this is going to be X1 minus l1 squared, plus x2 minus l2 squared, plus dot dot dot plus Xn minus ln squared.
* 7:29
* + now if your features
* 7:31
* take on very different ranges of value. So take a housing prediction, for example, if your data is some data about houses. + if X is in the range of thousands of square feet, for the first feature, X1. But if your second feature, X2 is the number of bedrooms. So if this is in the range of one to five bedrooms, then
* 7:57
* X1 minus l1 is going to be huge. This could be like a thousand squared, whereas X2 minus l2 is going to be much smaller + if that's the case, then in this term,
* 8:08
* those distances will be almost essentially dominated by the sizes of the houses
* 8:14
* + the number of bathrooms would be largely ignored.
* 8:16
* As so as, to avoid this in order to make a machine work well, do perform future scaling.
* 8:23
* + that will sure that the SVM gives, you know, comparable amount of attention to all of your different features, + not just to in this example to size of houses were big movement here the features.
* 8:34
* When you try a SVMs chances are by far the two most common kernels you use will be the linear kernel, meaning no kernel, or the Gaussian kernel that we talked about. + just one note of warning which is that not all similarity functions you might come up w/ are valid kernels. + the Gaussian kernel + the linear kernel + other kernels that you sometimes others will use, all of them need to satisfy a technical condition. It's called Mercer's Theorem + the reason you need to this is because SVM algorithms or implementations of the SVM have lots of clever numerical optimization tricks. In order to solve for the parameter's Ө efficiently + in the original design envisaged, those are decision made to restrict our attention only to kernels that satisfy this technical condition called Mercer's Theorem. + what that does is, that makes sure that all of these SVM packages, all of these SVM software packages can use the large class of optimizations + get the parameter Ө very quickly.
* 9:39
* So, what most people end up doing is using either the linear or Gaussian kernel, but there are a few other kernels that also satisfy Mercer's theorem + that you may run across other people using, although I personally end up using other kernels you know, very, very rarely, if at all. Just to mention some of the other kernels that you may run across.
* 9:57
* One is the polynomial kernel.
* 10:01
* + for that the similarity between X + l is defined as, there are a lot of options, you can take X(t)\*l squared. So, here's one measure of how similar X + l are. If X + l are very close w/ each other, then the inner product will tend to be large.
* 10:20
* + so, you know, this is a slightly
* 10:23
* unusual kernel. That is not used that often, but
* 10:26
* you may run across some people using it. This is one version of a polynomial kernel. Another is X(t)\*l cubed.
* 10:36
* These are all examples of the polynomial kernel. X(t)\*l plus 1 cubed.
* 10:42
* X(t)\*l plus maybe a number different then one 5 +, you know, to the power of 4 +
* 10:47
* so the polynomial kernel actually has two parameters. One is, what number do you add over here? It could be 0. This is really plus 0 over there, as well as what's the degree of the polynomial over there. So the degree power + these numbers. + the more general form of the polynomial kernel is X(t)\*l, plus some constant + then to some degree in the X1 + so both of these are parameters for the polynomial kernel. So the polynomial kernel almost always or usually performs worse. + the Gaussian kernel does not use that much, but this is just something that you may run across. Usually it is used only for data where X + l are all strictly non negative, + so that ensures that these inner products are never negative.
* 11:37
* + this captures the intuition that X + l are very similar to each other, then maybe the inter product between them will be large. They have some other properties as well but people tend not to use it much.
* 11:49
* + then, depending on what you're doing, there are other, sort of more
* 11:52
* esoteric kernels as well, that you may come across. You know, there's a string kernel, this is sometimes used if your input data is text strings or other types of strings. There are things like the chi-square kernel, the histogram intersection kernel, + so on. There are sort of more esoteric kernels that you can use to measure similarity between different objects. So for example, if you're trying to do some sort of text classification problem, where the input x is a string then maybe we want to find the similarity between two strings using the string kernel, but I personally you know end up very rarely, if at all, using these more esoteric kernels. I think I might have use the chi-square kernel, may be once in my life + the histogram kernel, may be once or twice in my life. I've actually never used the string kernel myself. But in case you've run across this in other applications. You know, if
* 12:42
* you do a quick web search we do a quick Google search or quick Bing search you should have found definitions that these are the kernels as well. So
* 12:51
* just two last details I want to talk about in this video. One in multiclass classification. So, you have four classes or more generally 3 classes output some appropriate decision bounday between your multiple classes. Most SVM, many SVM packages already have built-in multiclass classification functionality. So if your using a pattern like that, you just use the both that functionality + that should work fine. Otherwise, one way to do this is to use the one versus all method that we talked about when we are developing logistic regression. So what you do is you trade kSVM's if you have k classes, one to distinguish each of the classes from the rest. + this would give you k parameter vectors, so this will give you, upi lmpw. Ө 1, which is trying to distinguish class y equals one from all of the other classes, then you get the second parameter, vector Ө 2, which is what you get when you, you know, have y equals 2 as the positive class + all the others as negative class + so on up to a parameter vector Ө k, which is the parameter vector for distinguishing the final class key from anything else, + then lastly, this is exactly the same as the one versus all method we have for logistic regression. Where we you just predict the class i w/ the largest Ө(t)\*X. So let's multiclass classification designate. For the more common cases that there is a good chance that whatever software package you use, you know, there will be a reasonable chance that are already have built in multiclass classification functionality, + so you don't need to worry about this result. Finally, we developed SVMs starting off w/ logistic regression + then modifying the cost function a little bit. The last thing we want to do in this video is, just say a little bit about. when you will use one of these two algorithms, so let's say n is the number of features + m is the number of training examples.
* 14:43
* So, when should we use one algorithm versus the other?
* 14:47
* Well, if n is larger relative to your training set size, so for example,
* 14:52
* if you take a business w/ a number of features this is much larger than m + this might be, for example, if you have a text classification problem, where you know, the dimension of the feature vector is I don't know, maybe, 10 thousand.
* 15:05
* + if your training set size is maybe 10 you know, maybe, up to 1000. So, imagine a spam classification problem, where email spam, where you have 10,000 features corresponding to 10,000 words but you have, you know, maybe 10 training examples or maybe up to 1,000 examples.
* 15:22
* So if n is large relative to m, then what I would usually do is use logistic regression or use it as the m w/out a kernel or use it w/ a linear kernel. Because, if you have so many features w/ smaller training sets, you know, a linear function will probably do fine, + you don't have really enough data to fit a very complicated nonlinear function. Now if is n is small + m is intermediate what I mean by this is n is maybe anywhere from 1 - 1000, 1 would be very small. But maybe up to 1000 features + if the number of training examples is maybe anywhere from 10, you know, 10 to maybe up to 10,000 examples. Maybe up to 50,000 examples. If m is pretty big like maybe 10,000 but not a million. Right? So if m is an intermediate size then often an SVM w/ a linear kernel will work well. We talked about this early as well, w/ the one concrete example, this would be if you have a two dimensional training set. So, if n is equal to 2 where you have, you know, drawing in a pretty large number of training examples.
* 16:24
* So Gaussian kernel will do a pretty good job separating positive + negative classes.
* 16:29
* One third setting that's of interest is if n is small but m is large. So if n is you know, again maybe 1 to 1000, could be larger. But if m was, maybe 50,000 + greater to millions.
* 16:47
* So, 50,000, a 100,000, million, trillion.
* 16:51
* You have very very large training set sizes, right.
* 16:55
* So if this is the case, then a SVM of the Gaussian Kernel will be somewhat slow to run. Today's SVM packages, if you're using a Gaussian Kernel, tend to struggle a bit. If you have, you know, maybe 50 thousands okay, but if you have a million training examples, maybe or even a 100,000 w/ a massive value of m. Today's SVM packages are very good, but they can still struggle a little bit when you have a massive, massive trainings that size when using a Gaussian Kernel.
* 17:22
* So in that case, what I would usually do is try to just manually create have more features + then use logistic regression or an SVM w/out the Kernel.
* 17:33
* + in case you look at this slide + you see logistic regression or SVM w/out a kernel. In both of these places, I kind of paired them together. There's a reason for that, is that logistic regression + SVM w/out the kernel, those are really pretty similar algorithms +, you know, either logistic regression or SVM w/out a kernel will usually do pretty similar things + give pretty similar performance, but depending on your implementational details, one may be more efficient than the other. But, where one of these algorithms applies, logistic regression where SVM w/out a kernel, the other one is to likely to work pretty well as well. But along w/ the power of the SVM is when you use different kernels to learn complex nonlinear functions. + this regime, you know, when you have maybe up to 10,000 examples, maybe up to 50,000. + your number of features,
* 18:26
* this is reasonably large. That's a very common regime + maybe that's a regime where a SVM w/ a kernel kernel will shine. You can do things that are much harder to do that will need logistic regression. + finally, where do neural networks fit in? Well for all of these problems, for all of these different regimes, a well designed neural network is likely to work well as well.
* 18:50
* The one disadvantage, or the one reason that might not sometimes use the neural network is that, for some of these problems, the neural network might be slow to train. But if you have a very good SVM implementation package, that could run faster, quite a bit faster than your neural network.
* 19:05
* +, although we didn't show this earlier, it turns out that the optimization problem that the SVM has is a convex
* 19:12
* optimization problem + so the good SVM optimization software packages will always find the global minimum or something close to it. + so for the SVM you don't need to worry about local optima.
* 19:25
* In practice local optima aren't a huge problem for neural networks but they all solve, so this is one less thing to worry about if you're using an SVM.
* 19:33
* + depending on your problem, the neural network may be slower, especially in this sort of regime than the SVM. In case the guidelines they gave here, seem a little bit vague + if you're looking at some problems, you know,
* 19:46
* the guidelines are a bit vague, I'm still not entirely sure, should I use this algorithm or that algorithm, that's actually okay. When I face a machine learning problem, you know, sometimes its actually just not clear whether that's the best algorithm to use, but as you saw in the earlier videos, really, you know, the algorithm does matter, but what often matters even more is things like, how much data do you have. + how skilled are you, how good are you at doing error analysis + debugging learning algorithms, figuring out how to design new features + figuring out what other features to give you learning algorithms + so on. + often those things will matter more than what you are using logistic regression or an SVM. But having said that, the SVM is still widely perceived as one of the most powerful learning algorithms, + there is this regime of when there's a very effective way to learn complex non linear functions. + so I actually, together w/ logistic regressions, neural networks, SVM's, using those to speed learning algorithms you're I think very well positioned to build state of the art you know, machine learning systems for a wide region for applications + this is another very powerful tool to have in your arsenal. One that is used all over the place in Silicon Valley, or in industry + in the Academia, to build many high performance machine learning system.
* Downloads
* Lecture Videomp4
* Subtitles (English)
* WebVTT
* Transcript (English)
* txt