generative algorithms, SVMs

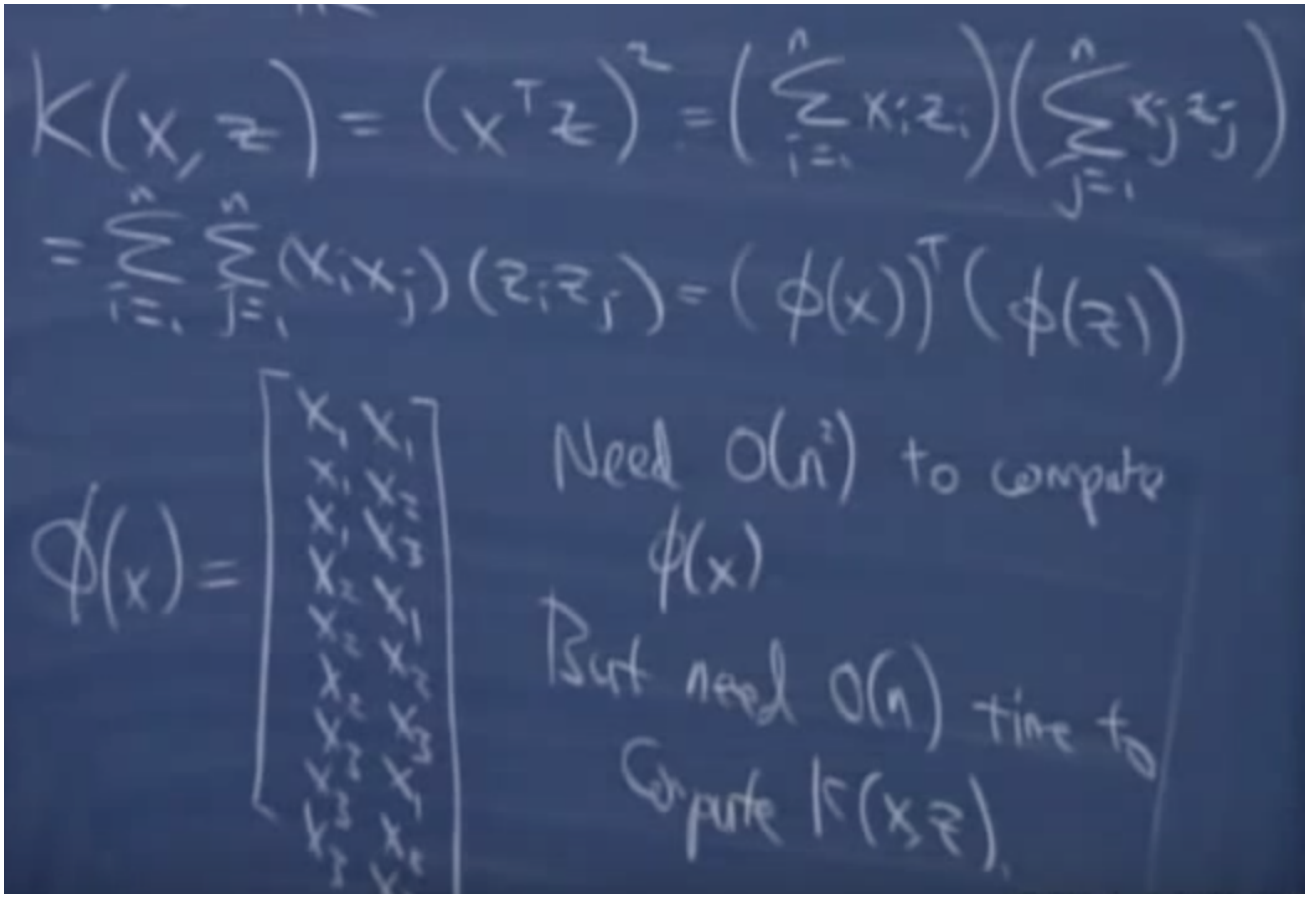
- the algorithms we've seen until now are called discriminative learning algorithms:
  - they either learn $p(y|x)$ or they learn $h_\theta(x)$ directly
- generative learning algorithms instead learn $p(x|y)$ and $p(y)$ and then get back $p(y|x)$ using bayes' rule. $p(x|y)$ is the probability of the features given the label and $p(y)$ is the probability of the label. we will now look at one such algorithm.
- gaussian discriminant analysis (GDA):
  - this is also used for binary classification. ng doesn't mention if this can be stretched to multinomial classification → i'd imagine it can.
  - each label type y has a different mean: μ_0, μ_1. $p(x|y)$ is given by using the multivariate gaussian formula where the input to the pdf is $\vec{x}$, with mean μ_0 for $y = 0$ and μ_1 for $y = 1$. we learn a covariance matrix Σ that is used for both pdfs (this model assumes same covariance matrix for both gaussians).
  - Now given this model, we want to derive the way to compute MLE estimates of these parameters.
  - for the marginal probability $p(y)$ in bayes' rule, we just compute that marginal from the frequencies of y's in the training data. μ_0 and μ_1 are also just the averages of the x's in the training data corresponding to $y = 0$ and $y = 1$ respectively.
  - because $p(y)$ depends on the frequency of y's in the training data, we have to be careful while choosing the number of different examples in the training data to reflect the true probability. same with the μ_k's. note that this is true, in general, for generative algorithms because they are trying to learn the probability distribution, the data we feed it must represent that distribution well for us to get a good model.
  - andrew does not mention how to learn the Σ matrix, but there is a MLE derivation for this.
  - the way to do prediction after learning is to simply use bayes rule to compute $p(y|x)$ and then choose the argmax. (the y that has the max $p(y|x)\ \alpha\ p(x|y)p(y)$)
- andrew ng states the following but doesn't prove it:
  - when $p(x|y)$ is modelled by a gaussian, like in GDA, the $p(y|x)$ turns out to be logistic. the reverse implication is not true.
  - even when $p(x|y)$ is poisson, $p(y|x)$ turns out to be logistic. should we use logistic regression or this? if you plot the data and see that its gaussian, then GDA might perform better. otherwise try logistic regression.
  - turns out if the data is any exponential distribution, then $p(y|x)$ is logistic, which means the logistic is pretty powerful → it captures a lot of distributions.
- another generative model is naive bayes':
  - we'll see an example of naive bayes' for email classification. we are going to represent an email as a vector $\vec{x}$ where x_i a 1 if it contains word i and 0 if not. so if there are n words in a language, the email vector $\epsilon\ R^n$. n is typically 10s of thousands.
  - now we want to learn $p(x|y)$. assume the x_i's are conditionally independent given y.
  - that means $p(x_1, x_2..x_n|y) = p(x_1|y)...p(x_2|y)$. in practice, this means once you know if an email is spam or not, then knowing the existence of one word does not affect the probability of existence of any other word. obviously this assumption isn't true for english sentences.

- using the frequencies seen in the training set, we compute $p(x_i|y=0)$ and $p(x_i|y=1)$ for each x_i. Given an email, we compute $p(x|y) = \prod p(x_i|y)$. then we apply bayes rule to arg max $p(x|y)p(y)$ to get the spam/no-spam classification.
- notice the limitation here: if a word has not been seen before, then we're screwed. also this does not take any advantage of sentence structure, which could def indicate spam/not spam. this model makes strong assumptions.
- another big problem with the above is that it is very easy to lead to $\frac{0}{0}$ forms, by virtue of not having seen words before.
- a way to get around the above problem is laplace smoothing. what it does is simple. when you use frequencies to compute a probability, instead of doing $\frac{num.1s}{num.0s+num.1s}$, we do $\frac{num.1s+1}{(num.0s+1)+(num.1s+1)}$. note how if we've never seen an x_i before, we assign set $p(x_i|y) = \frac{1}{2}$. Also if we've seen it 6 times, but all as not spam, then we set the probability of it $p(x_i|y=spam) = \frac{1}{7}$ and not 0. this is just a hack around the problem of "it is not impossible, so its not probability 0, we just have never seen it".
- similarly, this can be extended to the case where x_i can take on a few categories of values (multinomial) instead of just 2 categories. to apply bayes' to the continuous case, we can discretize the continuous inputs and outputs into buckets.
- naive bayes can also be modified to account for the number of times a word occurs. note that the version we presented does not account for this. this is a simple manipulation of the feature vector and i won't talk about that here. the math basically looks the same.

- we saw that the boundaries drawn by linear regression, logistic regression, softmax and gda were mostly linear boundaries. what if the data cannot be separated by a linear boundary? turns out, a lot of data can't be → which shouldn't be a surprise. so we will talk about a major non-linear classifier which is known to do really well on many problems: support vector machines.
- andrew briefly describes neural networks but we won't talk about that here since i've written extensive notes on neural nets and deep learning in the notes on 4 other courses!
- we will start off with describing a linear boundary version of SVMs and then go on to non-linear version of it. so our assumption for the linear version is that the training data is linearly separable. (only in that case is a linear SVM a good fit)
- Some change in notation, note that the labels $y\ \epsilon\ -1, 1$ so we will have our classifier either output, so $h\ \epsilon\ -1, 1$. instead of writing $h_\theta(x) = g(\theta^T x)$, we will write $h_\theta(x) = g(w^T x + b)$ and instead of having θ_0, we will have b.
- Some intuition: in logistic regression, we saw that if $\theta^T x >> 0$ then we have high confidence in our classification. ideally, we'd like to have it so that if $y = 1$, then $\theta^T x >> 0$ and same for the other case. this notion is called the functional margin, defined formally as $\hat{j}_i = y_i(w^T x_i + b)$ for training example x_i. this is called the functional margin of the hyperplane (hyperplane is just another term for the decision boundary in high dimensions)
- We desire that the functional margin is always high and positive. so if $y = 1$, then we want $w^T x + b >> 0$ and if $y = -1$, then we want $w^T x + b << 0$.
- Note that the condition for correct classification is that the functional margin is positive.
- the functional margin of a hyperplane with respect to an entire training set is the worst case, so $\hat{j} = min\ \hat{j}_i$. Turns out, simply making the functional margin large is easy, just multiply w, b by $2, 3, 4...$ etc and it can get arbitrarily large. so to compare different hyperplanes, we will first normalize the weights, so $||w|| = 1$ and then compare the functional margins of different hyperplanes. Note that scaling the weights does not change the hyperplane. $w^T x + b = 0$ is the same as $2w^T x + 2b = 0$.
- the geometric margin of a training example is the perpendicular distance between a hyperplane and the training example. From our calculus notes on planes, we know that if the equation of a plane is

$Ax + By + Cz + k = 0$ then the normal is A, B, C (see notes on vector arithmetic if you'd like a refresher of this)

- the unit vector perpendicular to the hyperplane is $\frac{w}{||w||}$. Now, if j_i is the geometric margin, let the point on the hyperplane that is perpendicular to our training example is $x_i - j_i \frac{w}{||w||}$. Now since this is on the hyperplane it satisfies $w^T(x_i - j_i \frac{w}{||w||}) + b = 0$. Solving for j_i, we get $j_i = \frac{w}{||w||}^T x_i + \frac{b}{||w||}$. To take into account the sign, we do $j_i = y_i(\frac{w}{||w||}^T x_i + \frac{b}{||w||})$.
- so we see that when the weights are normalized, the functional margin is the same as the geometric margin.
- Again, for the entire training set, we just take the min margin for the entire training set.
- the optimal margin classifier (a precursor to the SVM) chooses the weights so that the geometric margin over the training set (the worst case) is maximized. Turns out, stating the optimization problem this way, to maximize the min distance to any training example is not a convex optimization problem → we'd like to make it convex so we can just use gradient descent and find a global minima)
- Another way to formulate the above problem is to maximize $\frac{j}{||w||}$ where j is the functional margin, instead of the geometric margin. this is still not a convex problem, but will be useful as we will soon see.
- note that this same thing can be stated another way. we want to minimize $||w||$ so that $min_i\ y_i(w^T x_i + b) \geq 1$.
- Now this is basically a constrained optimization problem with $f(\mathbf{x}) = ||w||^2$ and inequality constraints $y_i(w^T x_i + b) \geq 1$. We can invoke the generalized lagrangian to solve this (see math notes from deep learning book for generalized lagrangian). The active constraints are the the cases where the functional margin is exactly equal to 1. These are called the support vectors. hence the name SVM.
- A big assumption here has been the points are clearly linearly separable and if there is noise that can mess up all this reasoning we've done so far. we will handle this case soon.
- Here we have assumed that $\mathbf{x}$ is finite dimensional and easily representable in memory. If $\mathbf{x}$ is super super high dimensional where we can't easily represent in memory, we will see a trick called kernels that will help us easily represent the inner product $\mathbf{x}_i^T \mathbf{x}_j$ efficiently even for infinite dimensional features. If you use the theory of KKT conditions and the formulation of the SVM problem as done here and work out the algebra, you will see that everything can be done in terms of these inner products so as long we can compute and represent this efficiently, we can handle super high dimensional feature spaces.
- Now lets look at what's called the kernel function → it gives us an efficient-to-compute representation of the inner product. Lets call our high dimensional feature space vector $\phi(x)$. We want to come up with a kernel function $K(x_i, x_j) =< \phi(x_i), \phi(x_j) >$. Here is an example feature vector $\phi(x)$ for which we can efficiently compute K. Note that this takes advantage of the example definition of ϕ here and is not a general trick for any ϕ.



- Note that the trick above and tricks like this make use of the fact that the high dimensional feature vectors are functions of a much smaller set of attributes. This lets us model non-linear things and learn non-linear boundaries. But if the features weren't all functions of a smaller set of attributes (like in the case of a text document), this trick doesn't really work so well.
- so really what we've done is by mapping a feature space where the decision boundary is not linear to a higher dimensional feature space, we just make the boundary linear in the higher dimensional space and then apply the SVM procedure we saw for linear boundaries.
- the above is just an example of a kernel but how do we come up with a kernel, given an ML problem? this is kind of an art. one trick is to realize that if things are "similar" the kernel (inner product of high dimensional feature vector) should be higher in magnitude and if dissimilar then lower in magnitude.
- Andrew mentions a test for valid kernels but doesn't really prove it so we leave it out here. Realize that if $K(x, x)$ is negative it can't be a valid kernel function because the kernel is basically a similarity measure and the similarity measure of something with itself should be non-negative.
- We have still made the assumption that the data is linearly separable in the high dimensional mapping. what if this is not true? we will see later, but this algorithm won't work if the data is not linearly separable in the high dimensional space.
- Notice that the beauty of the above is that we can come up with a kernel and not worry about the ϕ at all, because we can express our whole algorithm just in terms of the K's (inner products)
- also, turns out the idea of kernels can be used for linear regression, logistic regression etc so that they can be used for not just linear problems
- now lets address not perfectly linearly separable with something called L1 norm soft margin SVM. now we'd like to allow some data to "cross over" the decision boundary. translated to the constraints this means we can violate some of the current constraints, but not too much. a way to frame this is to modify

the function to be minimized from $f(x)$ to $f(x) + \beta \sum_i C_i$ and instead of $y_i(w^T x_i + b) \geq 1$, we set the constraint to $y_i(w^T x_i + b) \geq 1 - C_i^2$. This formulation means we pay a cost for each constraint that's violated (for each datapoint that crosses over the line). But we are less likely to get swayed by one bad point that moves it worse for all other points.

- Until now, we've framed the optimization problems involved in SVMs. Now we will look at a couple of algorithms that are useful in solving these optimization problems that come up in SVM.
- coordinate ascent: in coordinate ascent, we want to maximize something like $f(x_1, ..x_n)$ (pretty typical). we choose a x_i, maximize with respect to that, holding everything else fixed. then we choose a different x_j and so on. one idea is to keep cycling between all the x_i's but sometimes you can make progress faster by having some smart heuristics to choose which i to optimize over, while holding the rest fixed. Coordinate is the idea behind what's called the sequential minimal optimization, or SMO algorithm.
- "Sometimes there are research papers written on how to come up with a kernel for a given problem."