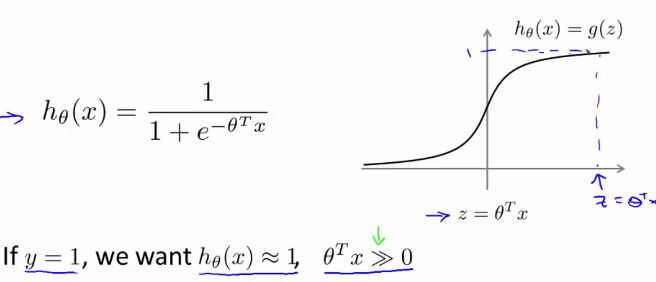
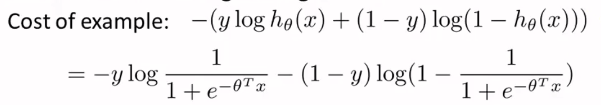
***Large Margin Classification***

**I. OPTIMIZATION OBJECTIVE**

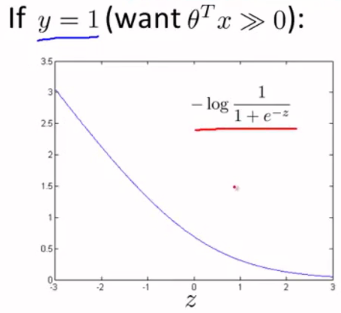
* The performance of many supervised learning algorithms will be pretty similar + often what matters less will be whether you use learning algorithm A or learning algorithm B
* What matters more will often be things like the *amount* of data you create algorithms on, as well as *skill* in applying these algorithms (choice/design of features to give to a learning algorithm, + how we choose the regularization parameter, etc.)
* But, there's 1 more very powerful + very widely used algorithm (both in industry + academia), the **support vector machine (SVM)**
* Compared to both logistic regression + NNs, the SVM sometimes gives a cleaner + more powerful way of learning complex non-linear functions
* To describe the SVM, start w/ logistic regression + modify it a bit to get what is essentially the SVM
* In logistic regression, we have the familiar form of the hypothesis hϴ(x) + the sigmoid activation function, z, denoted as ϴ(t)\*X



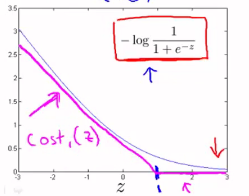
* We want logistic regression to perform such that when an example has y = 1, we hope hϴ(x) will be close to 1 (hoping to correctly classify that example)
* Having hϴ(x) close to 1 means ϴ(t)\*x must be must larger than 0, as we can see on the graph
* Conversely, for an example where y = 0, we hope hϴ(x) will be close to 0 (hoping to correctly classify that new example), which mean corresponds to ϴ(t)\*will be < 0
* If you look at the cost function of logistic regression:



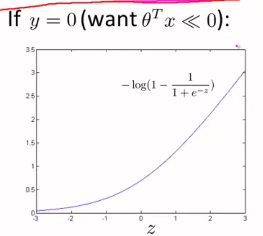
* We find each example (x, y) contributes a term to the overall cost function
* So, for the *overall* cost function, we sum over all examples + the 1/m term
* The expression above is the cost for a *single* training example, which contributes to the overall objective function.
* If we take the definition of the hypothesis + plug it, we get the bottom expression
* Now consider 2 cases of when y = 1 and y = 0
* When y = 1, only the 1st term in the expression matters, b/c the (1 – y) term = 0 and cancels out the 2nd term
* If we plot the resulting function as a function of z, we see that when z is large (ϴ(t)x is large), it gives us a small value for the cost function of that example



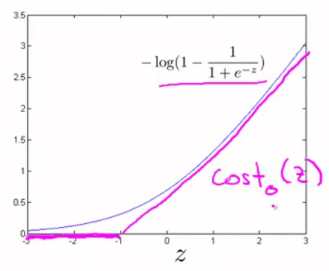
* This explains why, when logistic regression sees a positive example (y = 1), it tries to set z to be very large, b/c doing so corresponds to cost function being small.
* Now, to build an SVM, we take this cost + modify it a little bit by making the cost = 0 for z values > 1



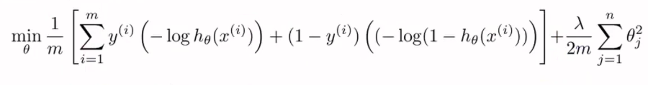
* This new curve is a pretty close approximation to the cost function used by logistic regression, except it is now made up of 2 line segments
* This is the new cost function we're going to use for y = 1
* You can imagine it should do something pretty similar to logistic regression, this will give the SVM computational advantages +, later on, give us an easier optimization problem for software to solve.
* The other case is if y = 0, in which only the 2nd term in the cost function matters as the 1st term is zeroed out



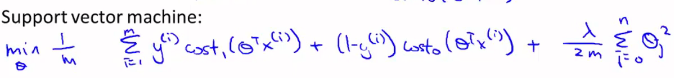
* For the SVM, again, plug in the new line



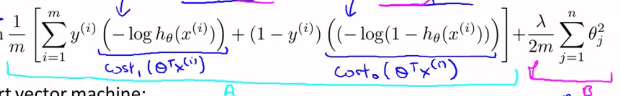
* So we have cost(z)1 and cost(z)0 for the costs corresponding to when y = 1 vs. when y = 0
* Armed w/ these definitions, we're now ready to build a SVM.
* Remember the cost function, J(ϴ), we have for logistic regression:



* For the SVM, we replace our log and h(ϴ)x terms with cost(z)1 + cost(z)0, where z = ϴ(t)\*x



* Now, by convention, for the SVM, we *re-parameterize* this slightly differently by removing the 1/m terms (1/m and **λ**/2m)
* This should give you us the same optimal value of ϴ b/c 1/m is just as constant, whether we solve this minimization problem w/ 1/m in front or not.
* Suppose I had a minimization problem of minimizing (U – 5)^2 + 1, whose minimum is U = 5
* If we multiply this objective function by 10, we get (10U – 50)^2 + 10, + the value of U that minimizes this is still U = 5
* The 2nd bit of notational change in the more standard convention when using SVMs instead of logistic regression, is as follows:
* For logistic regression, by setting different **λ** values, we control a trade-off between fitting the training set well (minimizing A) vs. how much we care about keeping values of parameters small

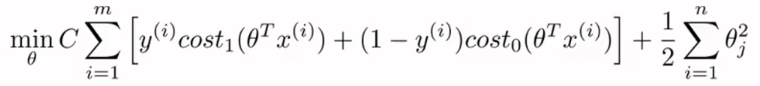




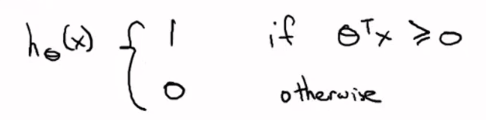
* For SVM, we have a different notation w/ a different parameter
* Instead of using **λ** to control the relative waiting between the 1st + 2nd (A + B) terms, we use C



* For logistic regression, if we set a very *large value of* ***λ***, we give B a very high weight
* If we set *C to be a very small value*, B again gets a much larger weight than A.
* So, this is just a different way of controlling the tradeoff + prioritizing how much we care about optimizing the 1st vs the 2nd term
* Think of this as the parameter C playing a role similar to 1/**λ**
* if C = 1/**λ**, the 2 optimization objectives should give the same value the same optimal value for
* **Overall Optimization Objective Function for the SVM.**

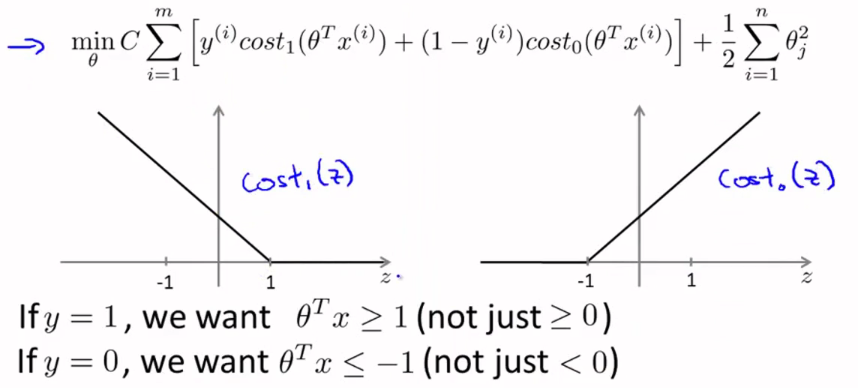


* If you minimize this, we get the parameters learned by the SVM.
*    
* 
* Finally, unlike logistic regression, the SVM doesn't output the probability
* It just makes a prediction of y = 1 or = 0 *directly*.
* Having learned the parameters ϴ, the **hypothesis for the SVM** is:

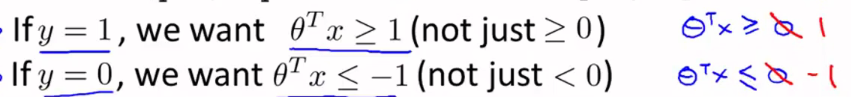


**II. LARGE MARGIN INTUITION**

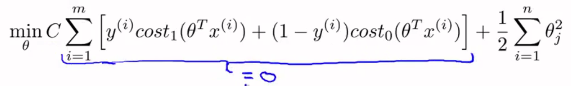
* Sometimes people talk about SVMs as **large margin classifiers**
* Here's my cost functions for the SVM



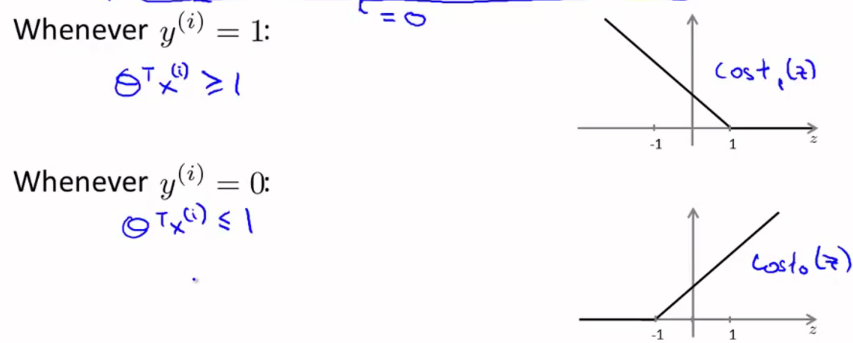
* Let's think about what it takes to make these cost functions small.
* For a positive example (y = 1), then cost(z)1 is = 0 only when z >= 1.
* In other words, if you have a positive example, we really want ϴ(t)\*x to be >= 1
* Conversely if y = 0, cost(z)0 is = 0 only when z <= -1
* This is an interesting property of the SVM right, which is that, w/ a positive example (y = 1), all we really need is that ϴ(t)\*x >= 0
* This means we classify the example correctly b/c if ϴ(t)\*x > 0 our hypothesis hϴ(x) will predict 0
* Similarly, w/ a negative example, really all we want is that ϴ(t)\*x < 0 to make sure we got the example right.
* *But the SVM wants a bit more than that*.
* It says “don't just barely get the example right” 🡪 don't just have ϴ(t)\*x just a little bit bigger than 0
* What I really want is for ABS(ϴ(t)\*x) to be quite a lot bigger than 0 + have it be >= 1 or <= -



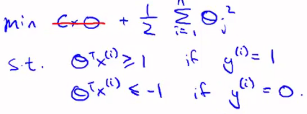
* This builds in an extra safety factor or **safety margin factor** into the SVM.
* Logistic regression does something similar too
* Consider a case where we set this constant C to be a very large value + see what a SVM will do.
* If C is very, very large, then when minimizing the optimization objective, we're going to be highly motivated to choose a value such that the 1st term = 0.



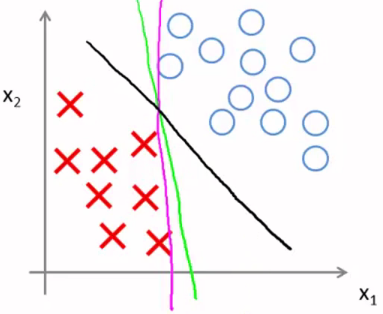
* We saw already that w/ a training example w/ a label of y = 1, in order to make that 1st term = 0, you need is to find a value of ϴ so that ϴ(t)\*x >= 1
* Similarly, w/ an example, w/ label y = 0, in order to make sure cost(z)0 = 0, we need ϴ(t)\*x <= -1



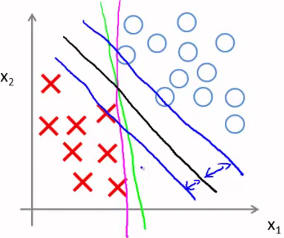
* So, if we now think of our optimization problem as choosing parameters to ensure show cost(z)1 = 0, 0, what we're left w/ is the following optimization problem:
* Minimize C\*0 (b/c we're going to choose parameters so it’s = 0) + 1/2\*cost(z)0
* This will be subject to the constraints that ϴ(t)\*x is >= 1 if y = 1 and ϴ(t)\*x <= -1 when y = 0

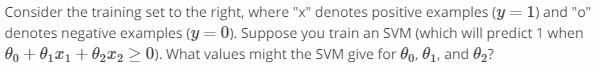


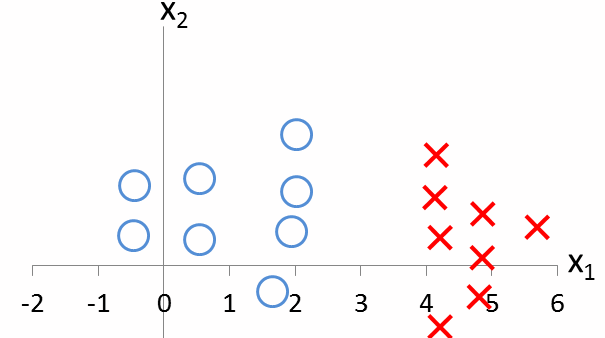
* It turns out that when you solve this optimization problem (when we minimize this as a function of the parameters ϴ), you get a very interesting **decision boundary**.
* If you look at a data set this w/ positive + negative examples that are linearly separable, but the decision boundaries (pink + green) to separate them won’t look very natural
* SVMs will instead choose the decision boundary (black).



* The black line seems like a more robust separator + does a better job of separating the examples
* Mathematically, this black decision boundary has a larger minimum distance (**margin**) from any of the training examples, whereas the magenta + the green lines they come awfully close to them

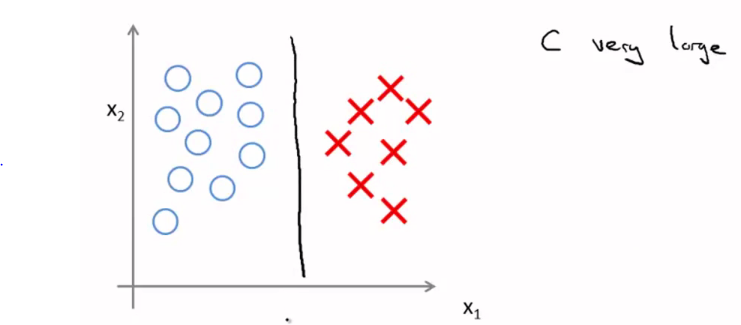


* This distance is called the **margin of the SVM** + it gives the SVM a certain robustness b/c it tries to separate the data w/ *as a large a margin as possible.*
* The SVM is sometimes also called a large margin classifier, which is a consequence of the optimization problem from before.
* 

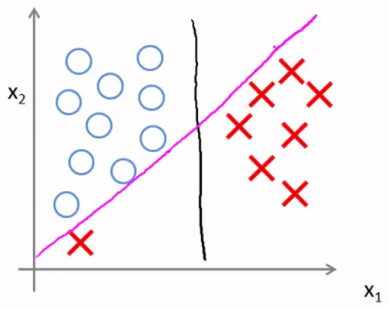




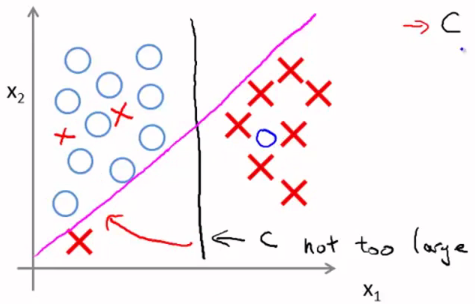
* We worked out this large margin classification setting in the case of when C was very large
* Given a dataset, maybe we'll choose a decision boundary that separate the positive + negative examples on large margin.



* The SVM is actually slightly more sophisticated than this large margin view might suggest
* In particular, if *ALL* you're doing is use a large margin classifier, your learning algorithms can be sensitive to outliers



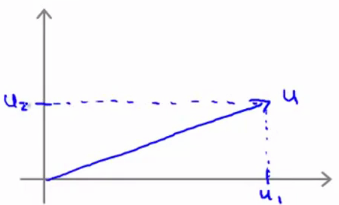
* It's clear that this is actually a good idea (to change my decision boundary from black to pink)
* So, if the regularization parameter C were very large, then this is actually what SVM will do
* But if C were reasonably small + not too large, you’d still end up w/ the black decision boundary
* Of course, if the data were not linearly separable, the SVM will also do the right thing



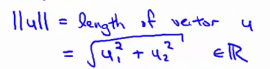
* This picture of a large margin classifier gives better intuition for the case of when the regularization parameter C is very large
* Remember: C plays a role similar to 1/**λ**, where **λ** is the regularization parameter we had previously
* So it's only when 1/**λ** is very large or **λ** is very small that you end up w/ things like the pink decision boundary
* BUT, in practice when applying SVMs, when C is not very large like that, it can do a better job ignoring the few outliers + do reasonable things even if data is not linearly separable.

**III. MATHEMATICS BEHINDLARGE MARGIN CLASSIFICATION**

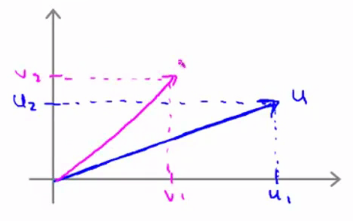
* Reminder of a couple of properties of **vector inner products**
* Have 2 vectors U + V, two 2-dimensional vectors.
* U(t)\*V = the **inner product** between the vectors U + V.
* U a 2D vector, so we can plot it



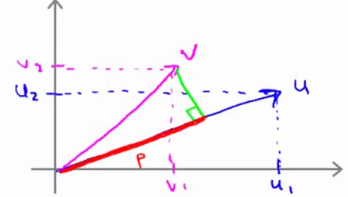
* 1 quantity that will be nice to have is the **norm of the vector U 🡪 ||U||,** or the **Euclidean length** of the vector U, which derives from the Pythagorean theorem for a real number U



* This gives the length of this vector/arrow on the plot = the norm of U
* Look at the vector V, some other vector

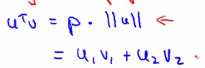


* Now compute the inner product between U + V
* Take V + *project down* onto U = take an **orthogonal projection**/90**-**degree projection onto U



* Measure length of the red line, P, which is the **magnitude of the projection of the vector V onto the vector U.**
* It’s possible to show that **U(t)\*V is going to be = P \* ||U||**
* This is 1 way to compute the inner product
* If you actually do the geometry figure out what P + the norm of U is, it should give you the same answer as the other way of computing unit product, U(t)\*V
* *\*\*\*\*||U|| and P are both real numbers*

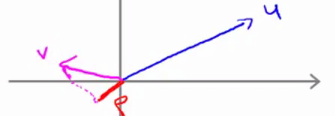




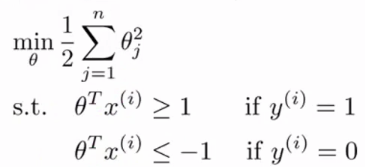
* ***NOTE:*** U(t)V is also equal to V(t)U
* So if you were to do the same process in reverse, instead of projecting V onto U, you could project U onto V 🡺 do the same process, but w/ the rows of U + V reversed
* You should actually get the same number
* Just 1 last detail, for the norm of P, ||P||, P is actually **signed** (can be positive OR negative)
* If U is a positive vector + V is a negative vector:



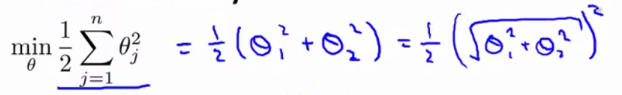
* And if the angle between U + V is > 90, then if I project V onto U, we get this a projection, P:



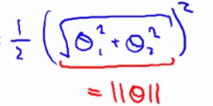
* In this case, we still have that U(t)V = P\*||U||, except in this example P is negative.
* So, for inner products, if the angle between U + V is < 90, P is a positive length + if the angle is > 90, P will be negative
* So the inner product between 2 vectors can also be negative if the angle between them is > 90
* We're going to use these properties of vector inner product to try to understand the SVM optimization objective:



* Simplification 🡪 ignore the intercept terms (ϴ0 = 0) + to make things easier to plot, set n (number of features) = 2
* When n= 2, the optimization objective of the SVM can be written as:



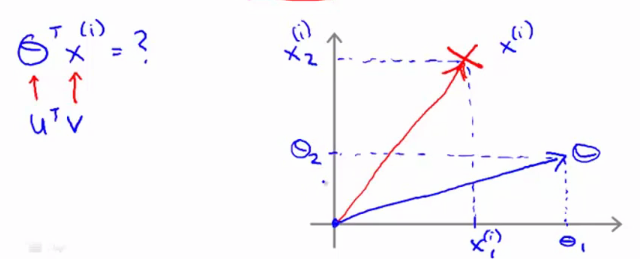
* Notice this final term inside the square root is equal to the norm/length of the vector ϴ



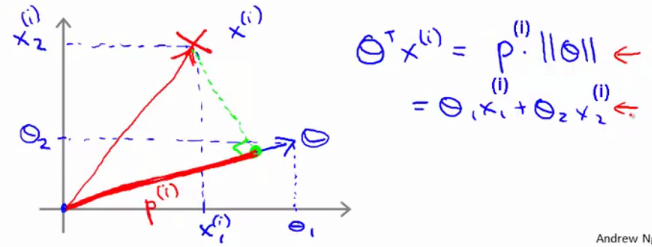
* The math works out whether we include ϴ0 here or not (won’t matter for the rest of the derivation)
* Finally, this means my optimization objective is equal to 1/2 ||ϴ||^2



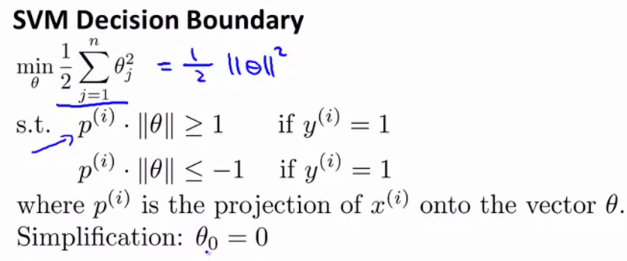
* So all the SVM is doing in the optimization objective is minimizing the squared norm of the squared length of the parameter vector ϴ.
* Now look at the terms, ϴ(t)X + understand better what they're doing.
* So given the parameter vector ϴ + given + example x(i), what is ϴ(t)X(i) equal to?
* Try using the strategy from U(t)\*V 🡪 plot the 2 features of x (n = 2) and the parameters ϴ



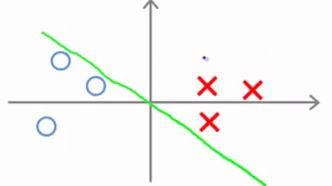
* *What is the inner product ϴ(t)X(i)?*
* Take x(i) + project it onto ϴ + look at the length of this segment, P(i), a projection of the ith training example
* So **ϴ(t)X(i) = P(i)\*||ϴ||**



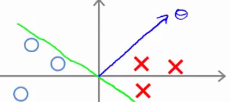
* Remember our constraints that ϴ(t)X(i) >= 1 or <= -1
* We can replace these of constraints by saying P(i)\*||ϴ|| must be >= 1 or <= -1



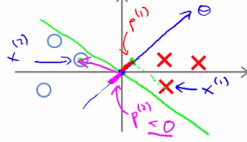
* Consider the training example + SVM-chosen decision boundary:



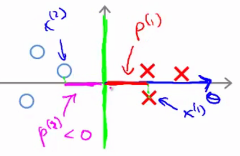
* This is not a very good choice b/c it has *very small margins* 🡪 decision boundary comes very close to the training examples.
* Let's see why the SVM will actually not do this.
* For this choice of parameters, it's possible to show that the parameter vector ϴ is actually at 90 degrees to the decision boundary (perpendicular)



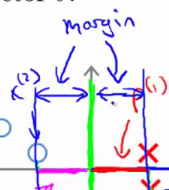
* \*\*\*Simplification of ϴ0 = 0 just means the decision boundary must pass through the origin
* Let's look at examples projected onto the parameters ϴ, P1 + P2



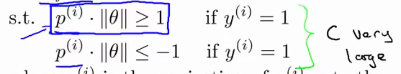
* What we find is that the terms P(i) are pretty small numbers
* Looking at the optimization objective, we that for positive examples (y = 1), we need P(i)\*||ϴ|| to be >= 1
* But if P1 over here is small, it means we need ||ϴ|| to be pretty large
* Similarly for our negative example, we need P2\*||ϴ||to be <= -1, and since P2 is small + negative, the only way for that to happen as well is for ||ϴ|| o be large
* *But what we’re doing in the optimization objective trying to find a setting of parameters where the ||ϴ|| is small*
* So, this doesn't seem like such a good direction for the parameter vector + ϴ
* In contrast, just look at a different decision boundary chosen by the SVM, its corresponding perpendicular vector ϴ, + the projections on it

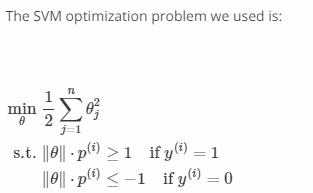
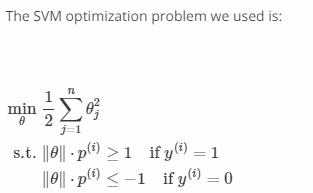
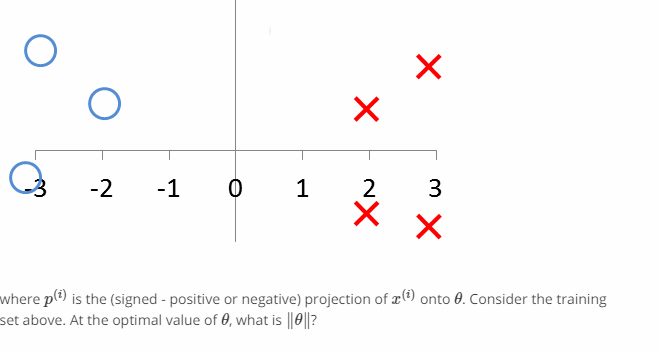
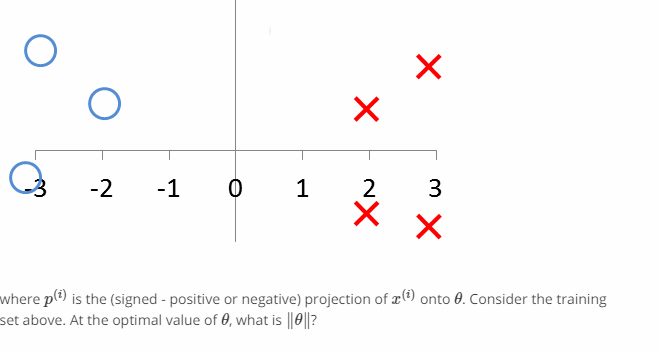


* Notice that now P1 + P2 are much bigger, so if we still need to enforce our constraints, b/c P1 and P2 are much bigger, ||ϴ|| can be smaller.
* By choosing *this* decision boundary, the SVM can make the ||ϴ|| much smaller + therefore make the squared ||ϴ|| smaller
* This is why the SVM would choose this 2nd hypothesis + how the SVM gives rise to the **large margin certification effect.**
* We want the projections of both positive + negative examples onto ϴ to be large, + the only way for that to hold true this is if there's a large gap/margin that separates positive + negative examples:



* The magnitude of this margin is exactly the values of P1, P2, P3 + so on
* By making the margin large (by making P1, P2, P3 + so on large), the SVM can end up w/ a smaller value for the ||ϴ||, which it’s doing in the objective
* This is why this SVM ends up w/ a large margin classifiers, b/c its trying to maximize the ||P(i)||’s, or the distance from the training examples to the decision boundary.
* We did this whole derivation using the simplification that parameter ϴ0 must be = 0.
* The effect of that is that we’re only entertaining decision boundaries that pass through the origin
* If you allow ϴ0 to be != 0, you can entertain decision boundaries that don’t cross through the origin
* It turns out that this same large margin proof works in pretty much in exactly the same way
* *Generally,* even when ϴ0 is != 0, the SVM is trying to find the large margin separator between the positive + negative examples (when you have this optimization objective, which corresponds to the case of when C is very large)



* 
* 
* **1/2**