**Elements of Statistical Learning**

**Chapter 12 – Support Vector Machines and Flexible Discriminants**

**12.1 Introduction**

Support Vector Machines are generalizations of linear decision boundaries for classification. Optimal separating hyperplanes are used when the classes can be linearly separable, but in the cases where there is overlap, SVMs are used. SVMs produces non-linear boundaries by constructing a linear boundary in a transformed version of the input space.

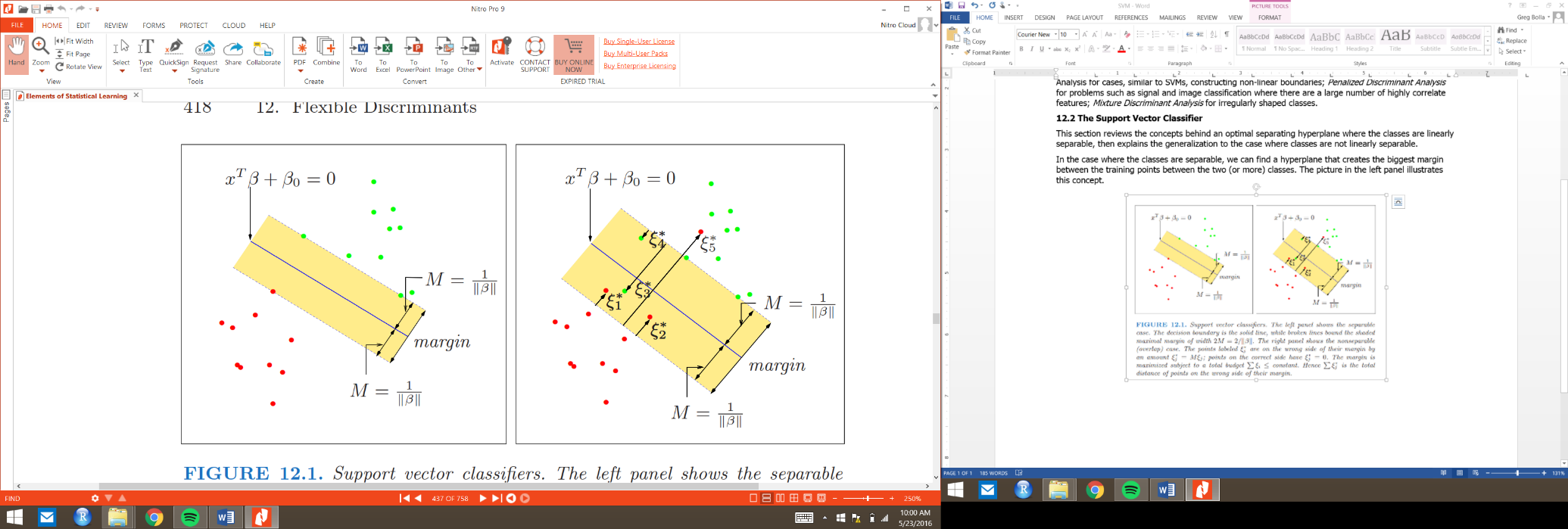
This chapter also discusses generalizes Fisher’s Linear Discriminant Analysis (LDA). *Flexible Discriminant* Analysis for cases, similar to SVMs, constructing non-linear boundaries; *Penalized Discriminant Analysis* for problems such as signal and image classification where there are a large number of highly correlate features; *Mixture Discriminant Analysis* for irregularly shaped classes.

SVMs are an approach that is designed by computer scientists; there are aren’t any probability models that make up the model, it simply searches for a hyperplane that separates the classes.

**12.2 The Support Vector Classifier**

This section reviews the concepts behind an optimal separating hyperplane where the classes are linearly separable, then explains the generalization to the case where classes are not linearly separable.

In the case where the classes are separable, we can find a hyperplane that creates the biggest margin between the training points between the two (or more) classes. The picture in the left panel illustrates this concept.



**Figure 1:** *Support vector classifiers. The left panel shows the separable case. The decision boundary is the solid line, while broken lines bound the shaded maximal margin of width . The right panel shows the nonseparable (overlap) case. The points labeled are on the wrong side of their margin by an amount . Points on the correct side have . The margin is maximized subject to a total budget some constant. Hence is the total distance of points on the wrong side of their margin.*

In the case where the classes are separable, we can find a hyperplane that creates the biggest margin between the training points between the two (or more) classes. The picture in the left panel illustrates this concept.

We’ve mentioned the concept of a hyperplane without much explanation as to what it is. The following is found from the ISL video here: <https://www.youtube.com/watch?v=QpbynqiTCsY>.

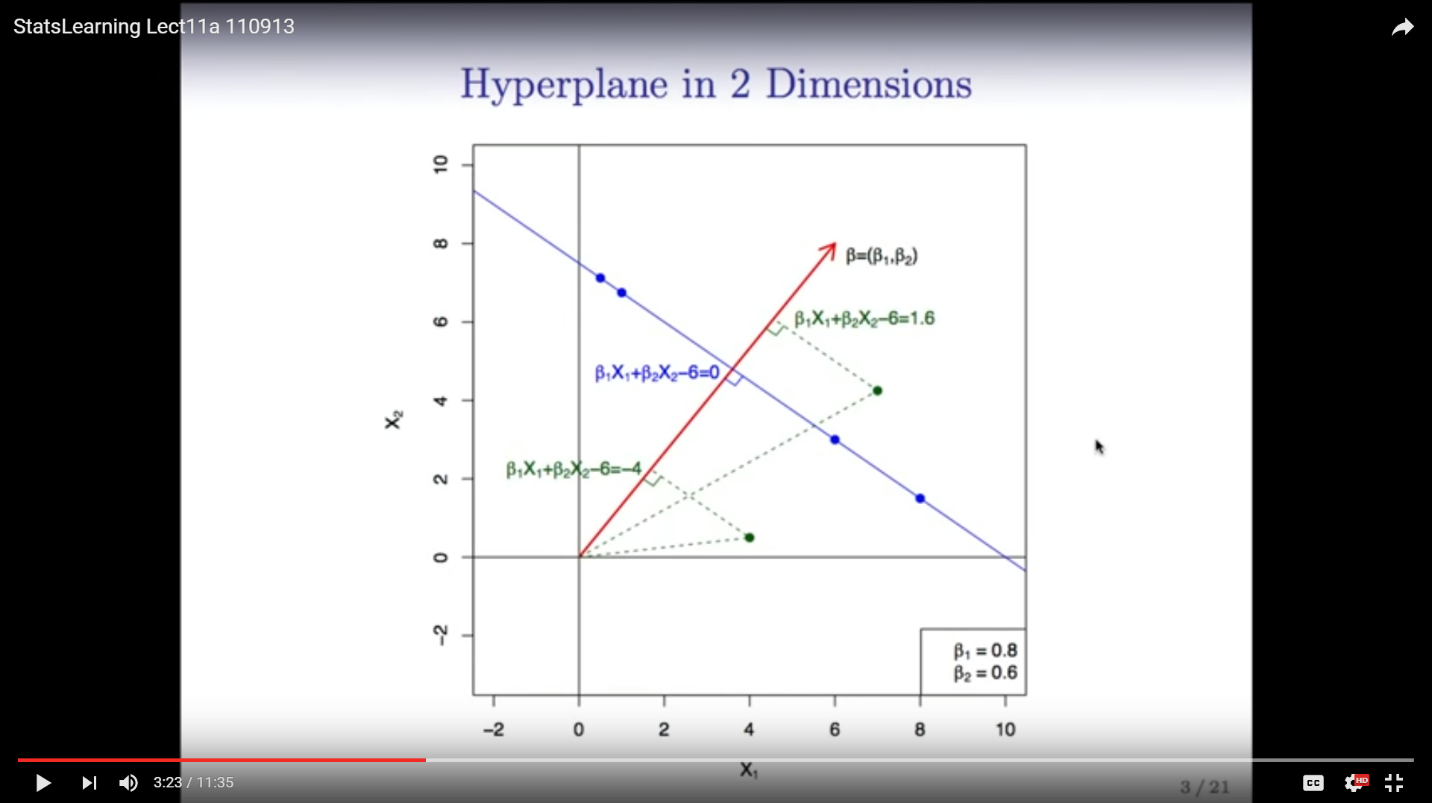
To give a technical definition, a hyperplane, in dimensions, is a *flat affine subspace* of dimension . Let’s split that phrase flat affine subspace down since it’s a lot to chew on. The input variable space is of dimensions. We call a hyperplane *affine* because we simply do not care where the origin is, we only care about where the observations within that variable space fall relative to one another.

Chewing on the *flat* portion of that phrase, that means that we are dealing with a congruent subspace of the original -dimension space. The flats of a two-dimensional space are points and lines. The flats of a three-dimensional space are points, lines, and planes. In this case, with dimensional space, there are flats of every dimension .

In general, the equation of the hyperplane has the form shown below. Notice that we set it equal to 0.

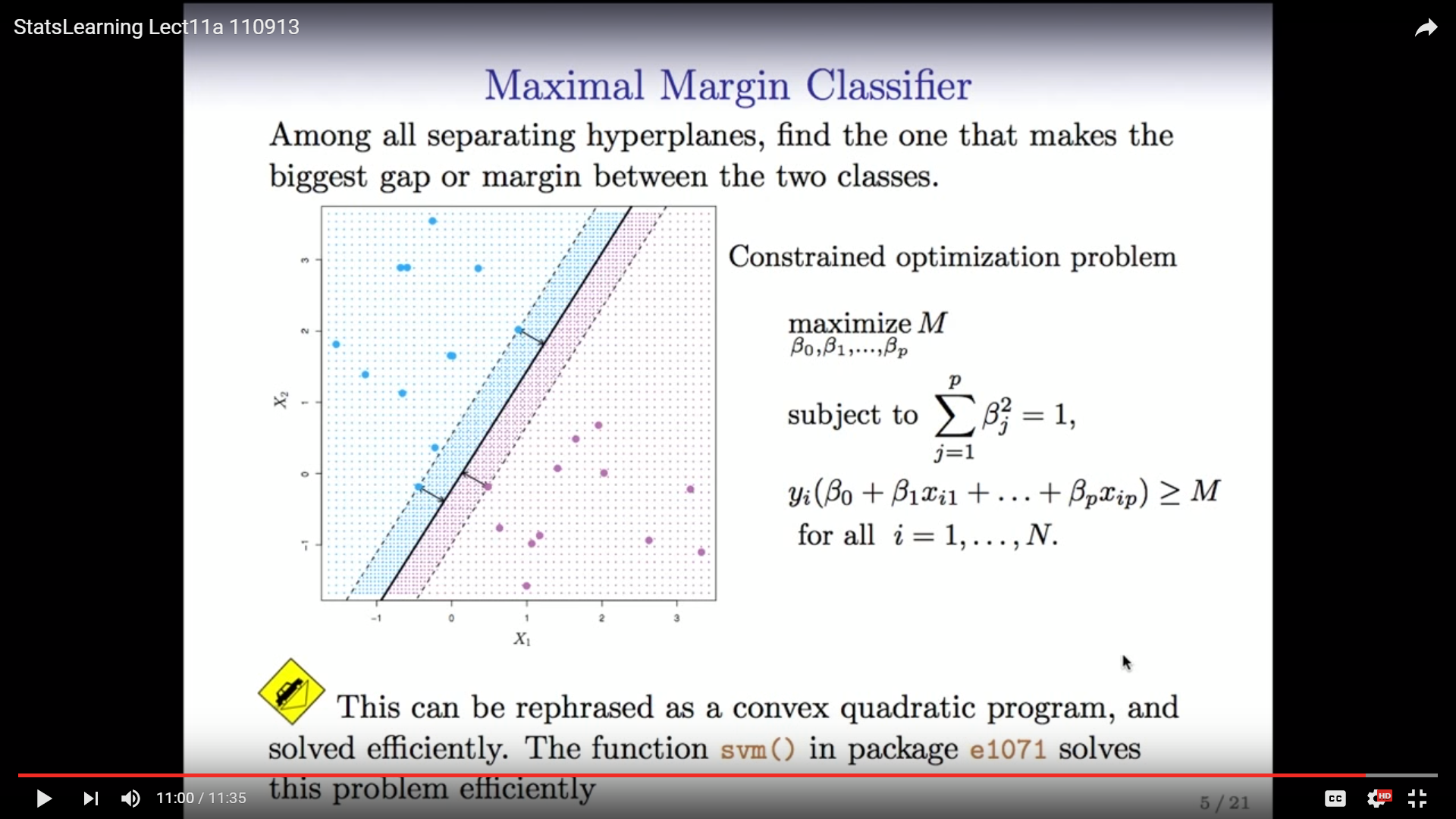
Another important thing to remember re: hyperplanes is that the vector of betas (excluding ) from the hyperplane equation above, , is called the *normal vector*, and runs orthogonal to the surface of the hyperplane.

The below picture shows a hyperplane and its normal vector in two dimensions. The blue line is the hyperplane, while the red line is the normal vector of the hyperplane. Notice that when we project any points above/below the hyperplane (green in the picture below), the result of the hyperplane equation is either positive (for points above the hyperplane), negative (for points below the hyperplane), or 0 (for points on the hyperplane). This becomes important because in SVMs we use the sign of any given point’s projection to classify that observation.



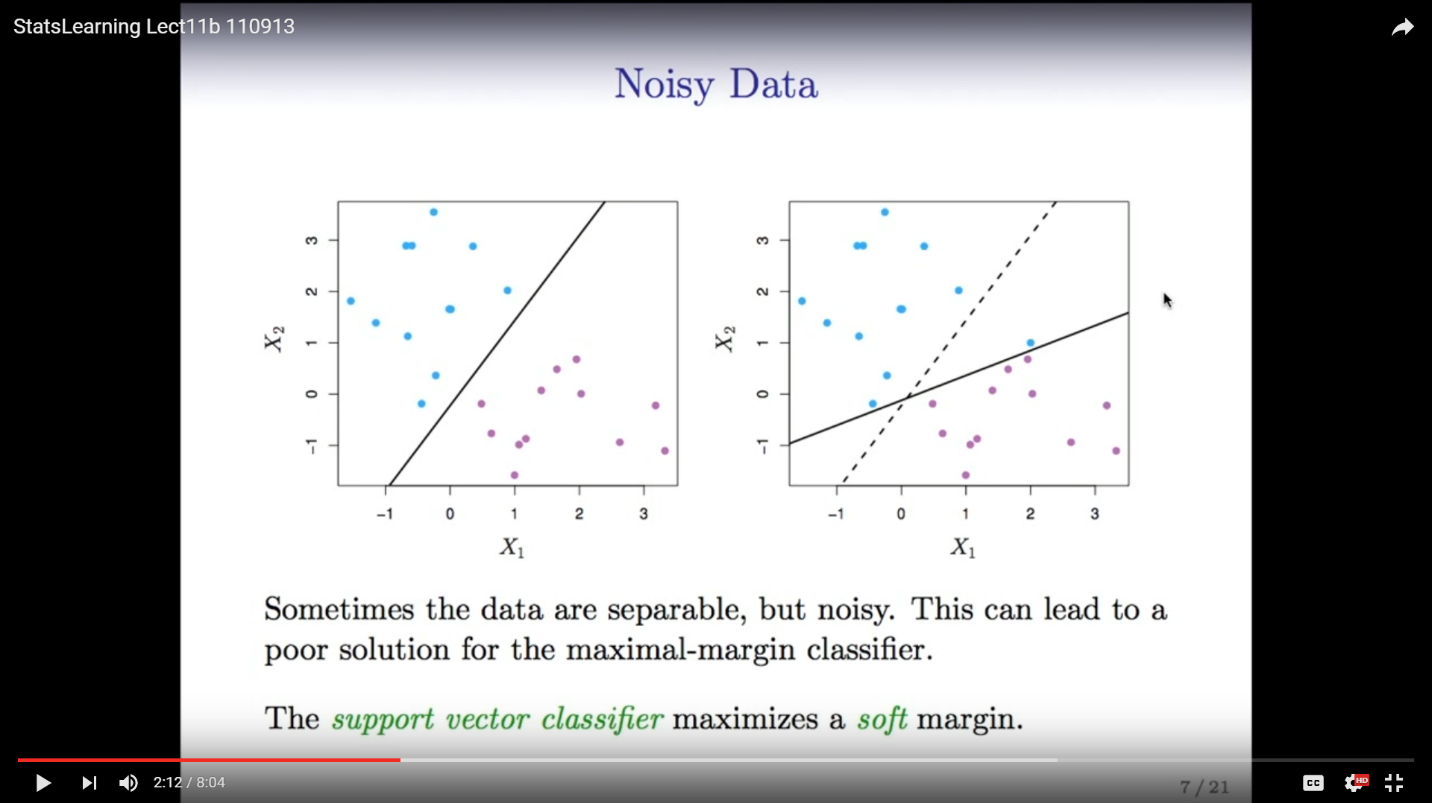
When a problem is linearly separable, you can have many different hyperplanes. The trick then becomes finding the optimal hyperplane for the given points, aka the *maximal margin classifier*, illustrated below.

The problem to solve for finding the maximal margin classifier can be done through software, however what it does, in a general sense, is constrain the sum of the squared betas (excluding the intercept term ) equal to 1.

The problem to solve for finding the maximal margin classifier can be done through software, however what it does, in a general sense, is first constrain the sum of the squared betas (excluding the intercept term ) equal to 1. In other words, is a *unit vector*, with the notation being . Obviously there’s a ton of choices/combinations of betas available which meets this criteria. The algorithm goes through each possible combination of betas (and their resulting hyperplane equations), then multiplies this equation by each training points (either 1 or -1 since it is a binary classifier).

The evaluation of this function for all points classified correctly will be positive (alternatively, all points classified incorrectly will be negative). Therefore if we sum up the evaluation of each hyperplane’s functions for each of the training points, the hyperplane that results in the largest sum is the optimal hyperplane for the training data.

Notice that this classifier is very reliant on a relative few number of points (namely, the two points closest to the margin/decision boundary). Therefore, this classifier is not robust against noisy data. The graph below depicts this situation; the presence of one noisy point (which could be an outlier) changes the chosen hyperplane quite dramatically.



In addition to this problem of lack of robustness, maximal margin classifiers also don’t take into account situations where the data is linearly separable (which is a vast majority - if not all - of the time in the real world).

To account for both of these issues, the support vector classifier relaxes the idea of a linearly separable hyperplane and implements a so-called *soft margin*.

The idea behind the soft margin is to allow the classifier to account for more points than just the relative few closest to the decision boundary. In other words, the parameter for the margin (which is chosen by the analyst) is a way of regularization to account for potentially noisy data. The underlying logic on how to find the optimal hyperplane remains the same, with one difference. Instead of maximizing in the equation below, as in the maximal margin classifier:

We add a slack variable, , that discounts by the given discount factor.

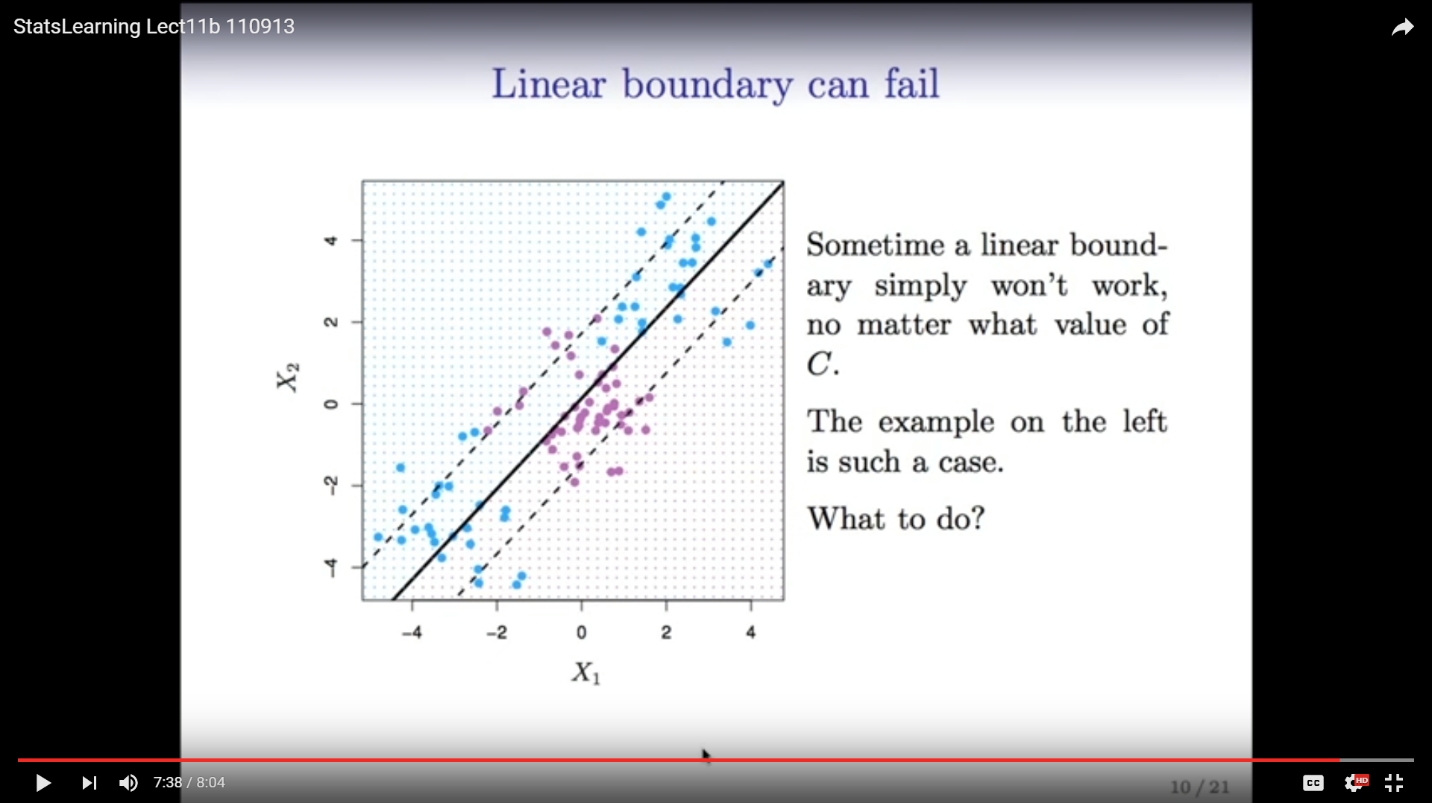
In other words, if a classifier results in a hyperplane that isn’t bigger than the margin, that particular classifier may not be penalized for misclassifying the given observation if the observation falls within the slack () of the margin. A constraint is put on the slack variable, or to be more precise, the sum of the epsilons/slack variables across all observations. This sum of the epsilons is called the cost parameter, or budget, often denoted by . The margin will increase or decrease depending on the analyst’s choice of ; Increasing your choice for will increase the margins, while decreasing will likewise decrease your margin.

In effect, there is a bias/variance tradeoff with your choice of . The larger the budget you allot for , the more stable the classifier becomes, while the smaller the budget for , the more precise your classifier, although the more prone it is to overfitting (since it takes into account fewer observations).

Another important note is that the units of the input variables matter to the support vector classifier. In other words, you must standardize the variables prior to running the SVM model.

*Kernels and Support Vector Machines -* [*https://www.youtube.com/watch?v=dm32QvCW7wE*](https://www.youtube.com/watch?v=dm32QvCW7wE)

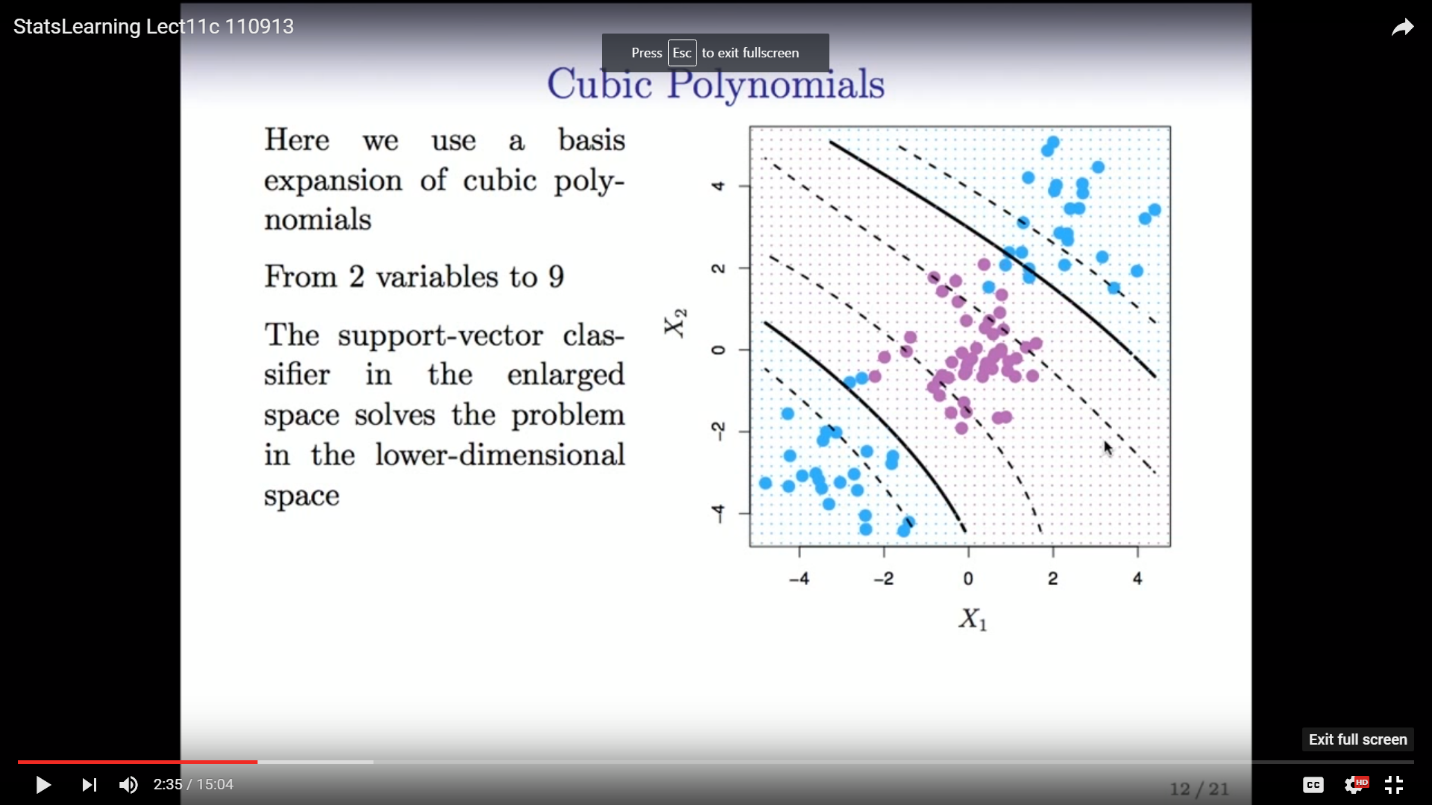
We have, up to now, only spoken about using the standardized inputs without further transformation. In some cases, however, it does not work to have a linear decision boundary in the original (standardized) input space. Below illustrates this issue; there is no linear boundary that can separate the data well.



In these cases, we can enlarge the number of features by including transformations such as polynomials (, etc.), interactions (), or a combination of both ().

The result of including these transformed variables in the SVM model is a nonlinear decision boundary, even though the underlying equation of the hyperplane is linear.

On the same problem above, where the linear boundary wasn’t appropriate, the transformations lead to a “quadratic conic” boundary (which performs quite well) shown below.



While this can lead to an accurate solution, the number of features can be blown up considerably if a transformation (or transformations) are applied to each original feature. To efficiently accommodate expanding the feature space without adding cumbersome computational burdens, we can increase the feature space in an efficient way using *kernels*.

*9.3.2 Support Vector Machines* (from Introduction to Statistical Learning textbook)

To counteract the computational considerations that come along with expanding the feature space to account for polynomials, interactions, etc. for every input, it was found that it is more efficient to calculate the inner products between all pairs of the training set. The equation for this inner product is given below:

Although there are more details I’m not quite grasping right now, we can use the inner products between all observations to represent the linear support vector classifier as:

You will notice the additional parameter included in the above equation. In order to estimate this alpha for each observation (along with ) all we need are the inner products between all pairs of training observations.

However, it turns out that the is non-zero only for the support vectors in the solution. The support vectors are the points of the support vector machine which fall within the specified margin (or on the wrong side of the decision boundary). In other words, all points that fall on the correct side of the decision boundary outside of the specified margin, do not need to be incorporated into the final support vector machine equation, thus easing the computational considerations.

OK, so we talked about inner products but haven’t touched much on the concept of kernels. Turns out that kernels are just generalizations of inner products – they are a function that describes the similarity between points/vectors. The equations above represent a linear kernel, which is essential the similarity between a pair of observations using the Pearson/standard correlation. There is also the *polynomial kernel:*

In the equation above, represents the degree of the polynomial that you’re trying to calculate the kernel for. Whenever the support vector classifier is combined with a non-linear kernel like the polynomial kernel above, this results in the support vector machine.

Another very popular kernel is the *radial kernel:*

In essence, what the radial kernel does is de-emphasize training points that are far from the given test point to be predicted. The thinking is that only points that are relatively close to the test point are incorporated when assigning that test point to a class. As such, the radial kernel is very *local* in its behavior.

Given all that has been discussed, the support vector machine function can be shown as below:

The represents that the only points that are taken into account are the support vectors, while the represents the specified kernel.

**12.2 The Support Vector Classifier (ESL Book, continued)**

OK, back to the ESL book after a (not-so-brief) foray into more introductory materials to understand some of the underlying concepts. Remember the optimization problem to find the maximal margin classifier:

with (the ’s, excluding the intercept )

Turns out that you can rewrite the above equation and drop the norm constraint on the betas. Instead of maximizing the margin , you minimize the unit vector (something I’m unclear on is how you minimize , considering that – based on my understanding – a unit vector must equal 1) subject to the following equation:

Note that in this case, the margin .

In the nonseparable case, we still want to maximize the margin, while allowing for some points to be on the wrong side of the margin. There are two ways to modify the constraint listed above:

or

Note that this applies to all training points ( in mathematical notation), with for all training points, and , which is the choice of budget mentioned above. The two choices lead to differenct solutions.

The first equation appears more straightforward/natural since it measures overlap in actual distance from the margin (i.e., in the same units), but this has the drawback of being a nonconvex optimization problem.

The second solution measures the amount of overlap in relative distance, which therefore changes with the width of the margin. However, this is a convex optimization problem and the one version that the authors of ESL prefer.

**12.3 Support Vector Machines and Kernels**

Much of this section is a reiteration of above from the ISL, but I wanted to call out a few important notes:

* In many cases, when the feature space is enlarged through polynomials, interactions, etc., the data can become perfectly separable.
* The optimal choice for can be found through cross-validation
* The neural network is a version of the Support Vector Machine classifier, with a different kernel. Instead of the popular polynomial or radial kernels, the neural network kernel is:

I know this isn’t very helpful, but I don’t know what stands for, however.