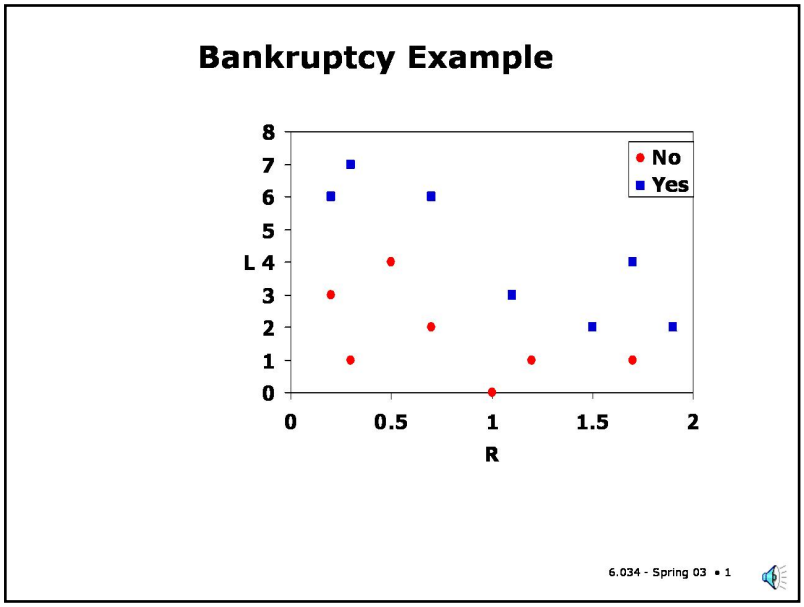
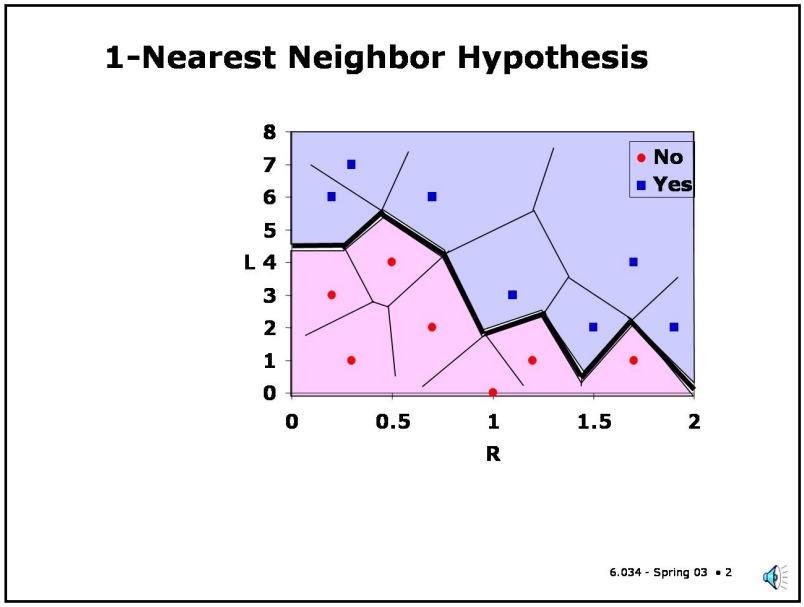
6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

6.034 Notes: Section 11.1

Slide 11.1.1
We have been using this simulated bankruptcy data set to illustrate the different learning algorithms that operate on continuous data. Recall that R is supposed to be the ratio of earnings to expenses while L is supposed to be the number of late payments on credit cards over the past year. We will continue using it in this section where we look at a new hypothesis class, **linear separators**.

One key observation is that each hypothesis class leads to a distinctive way of defining the **decision boundary** between the two classes. The decision boundary is where the class prediction changes from one class to another. Let's look at this in more detail.

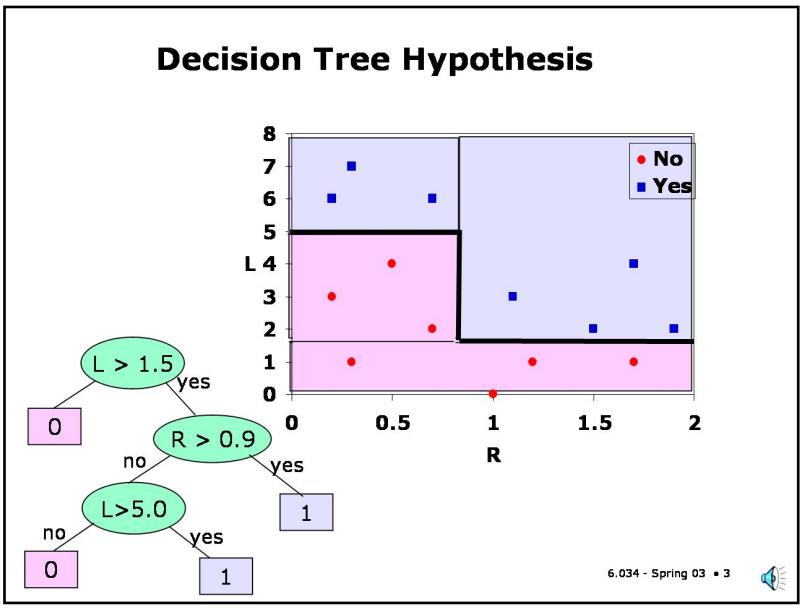


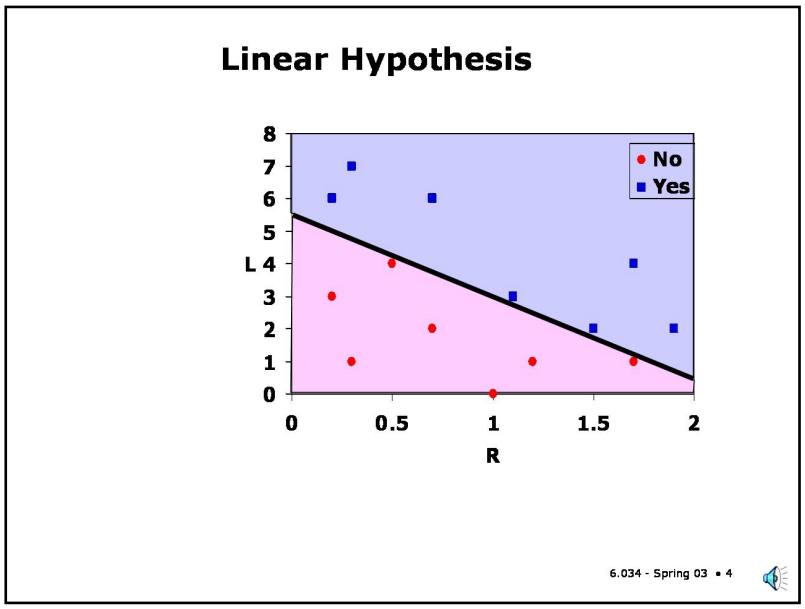


Slide 11.1.2
We mentioned that a hypothesis for the 1-nearest neighbor algorithm can be understood in terms of a Voronoi partition of the feature space. The cells illustrated in this figure represent the feature space points that are closest to one of the training points. Any query in that cell will have that training point as its nearest neighbor and the prediction will be the class of that training point. The decision boundary will be the boundary between cells defined by points of different classes, as illustrated by the bold line shown here.

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Slide 11.1.3
Similarly, a decision tree also defines a decision boundary in the feature space. Note that although both 1-NN and decision trees agree on all the training points, they disagree on the precise decision boundary and so will classify some query points differently. This is the essential difference between different learning algorithms.

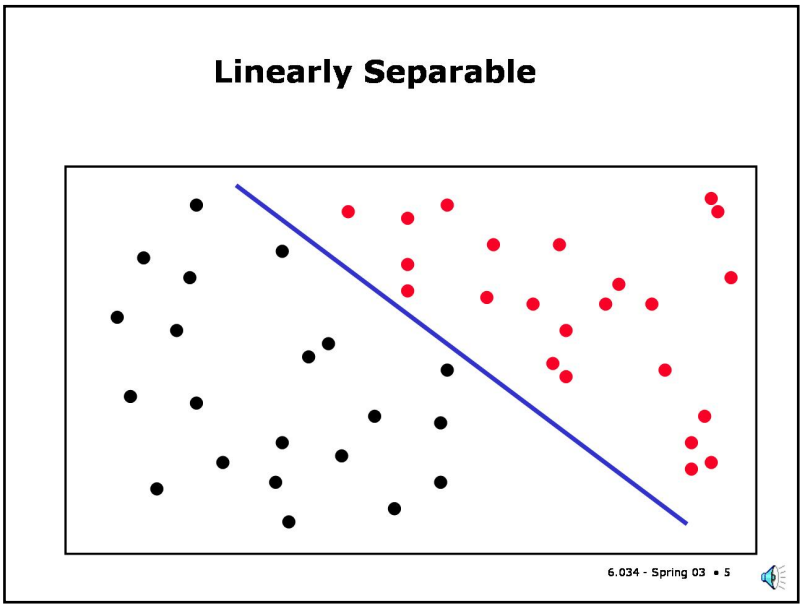




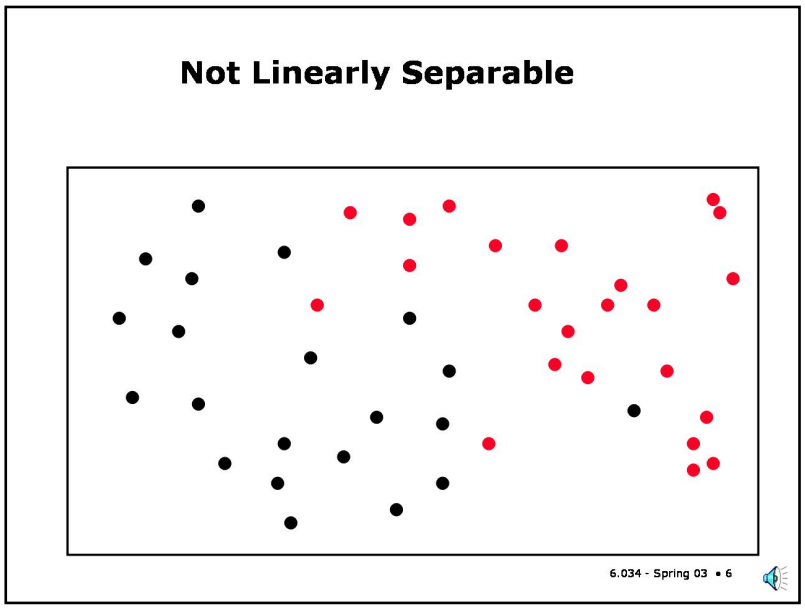
Slide 11.1.4
In this section we will be exploring **linear separators** which are characterized by a single linear decision boundary in the space. The bankruptcy data can be successfully separated in that manner. But, notice that in contrast to 1-NN and decision trees, there is no guarantee that a single linear separator will successfully classify any set of training data. The linear separator is a very simple hypothesis class, not nearly as powerful as either 1-NN or decision trees. However, as simple as this class is, in general, there will be many possible linear separators to choose from.

Also, note that, once again, this decision boundary disagrees with that drawn by the previous algorithms. So, there will be some data sets where a linear separator is ideally suited to the data. For example, it turns out that if the data points are generated by two Gaussian distributions with different means but the same standard deviation, then the linear separator is optimal.

Slide 11.1.5
A data set that can be successfully split by a linear separator is called, not surprisingly, **linearly separable**.



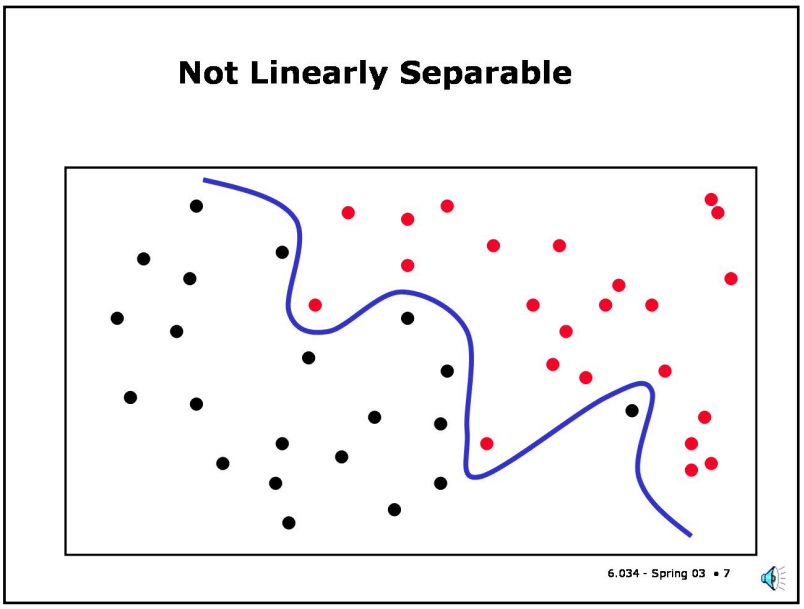
6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

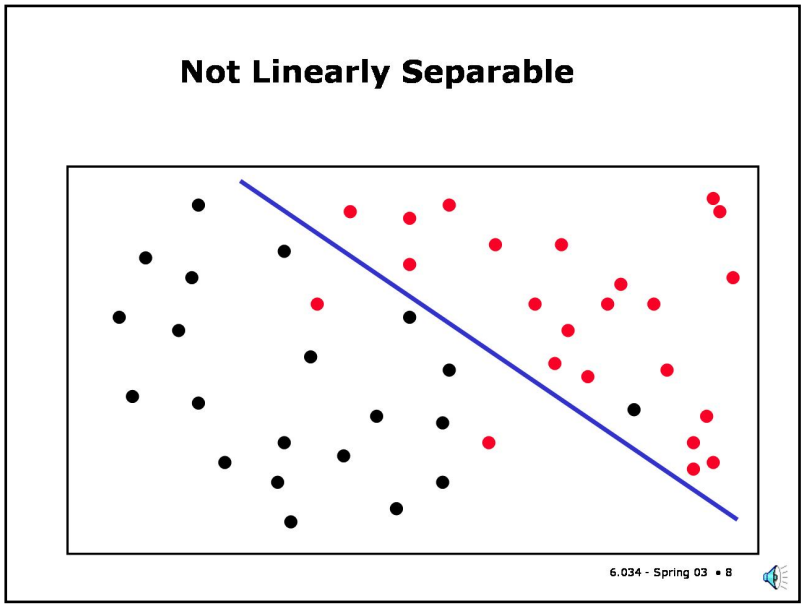


Slide 11.1.6
As we've mentioned, not all data sets are linearly separable. Here's one for example. Another classic non-linearly-separable data set is our old nemesis XOR.

It turns out, although it's not obvious, that the higher the dimensionality of the feature space, the more likely that a linear separator exists. This will turn out to be important later on, so let's just file that fact away.

Slide 11.1.7
When faced with a non-linearly-separable data set, we have two options. One is to use a more complex hypothesis class, such as shown here.





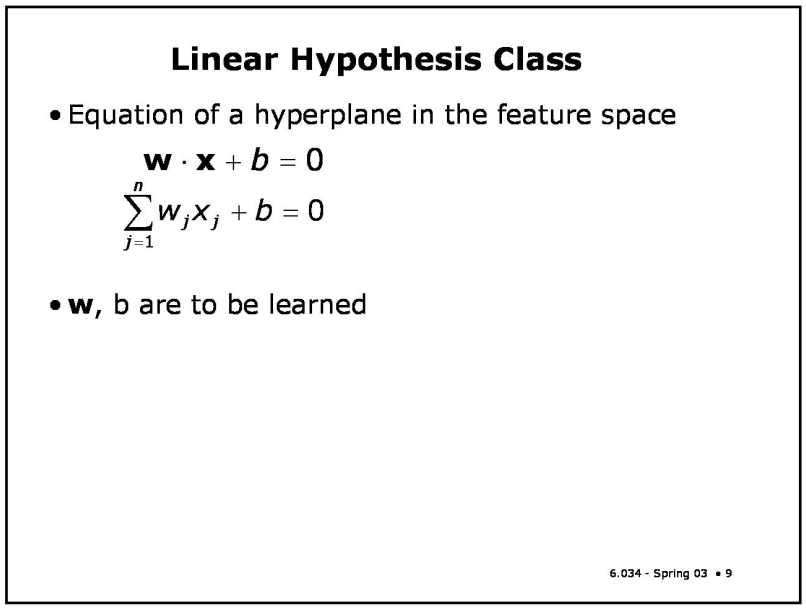
Slide 11.1.8
Or, keep the simple linear separator and accept some errors. This is the classic bias/variance tradeoff. Use a more complex hypothesis with greater variance or a simpler hypothesis with greater bias. Which is more appropriate depends on the underlying properties of the data, including the amount of noise. We can use our old friend cross-validation to make the choice if we don't have much understanding of the data.

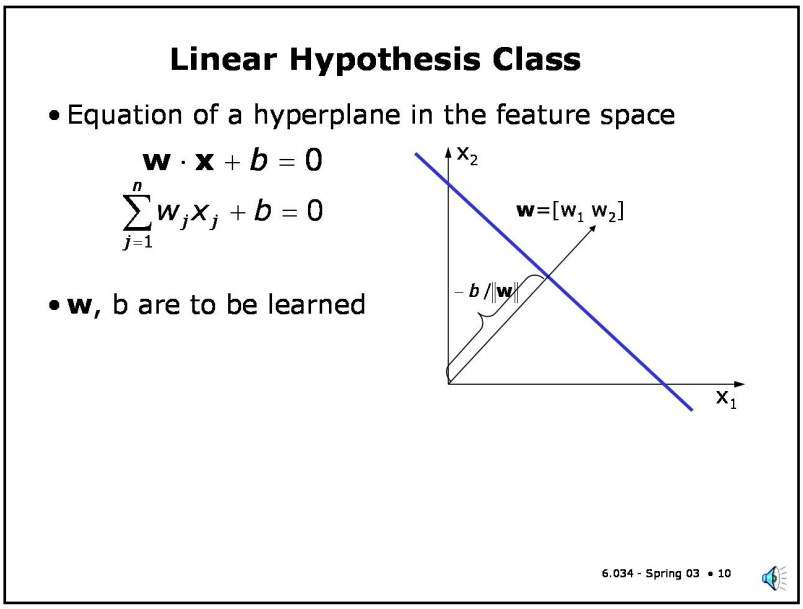
6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Slide 11.1.9
So, let's look at the details of linear classifiers. First, we need to understand how to represent a particular hypothesis, that is, the equation of a linear separator. We will be illustrating everything in two dimensions but all the equations hold for an arbitrary number of dimensions.

The equation of a linear separator in an n-dimensional feature space is (surprise!) a linear equation which is determined by n+1 values, the components of an n-dimensional coefficient vector **w** and a scalar value *b*. These n+1 values are what will be learned from the data. The **x** will be some point in the feature space.

We will be using dot product notation for compactness and to highlight the geometric interpretation of this equation (more on this in a minute). Recall that the dot product is simply the sum of the componentwise products of the vector components, as shown here.

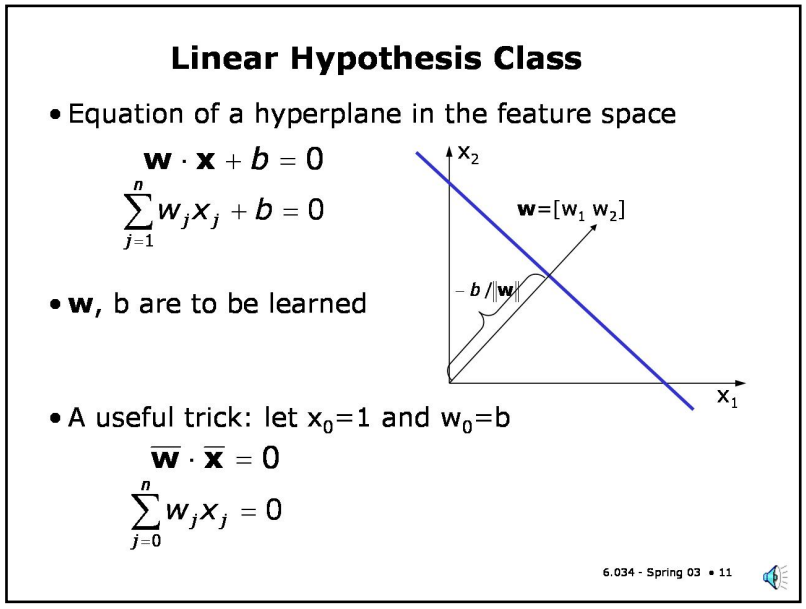




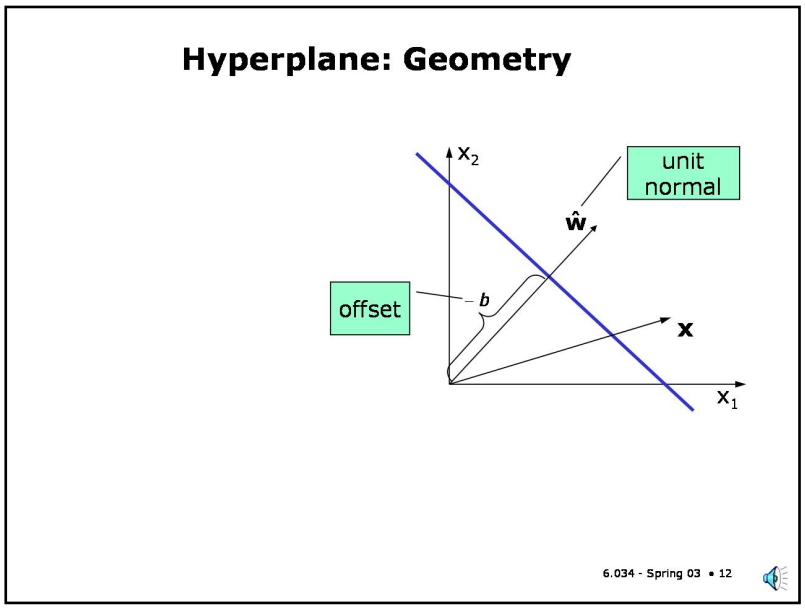
Slide 11.1.10
In two dimensions, we can see the geometric interpretation of **w** and *b*. The vector **w** is perpendicular to the linear separator; such a vector is known as the **normal** vector. Often we say "the vector normal to the surface". The scalar *b*, which we will call the **offset**, is proportional to the perpendicular distance from the origin to the linear separator. The constant of proportionality is the negative of the reciprocal of the magnitude of the normal vector. We'll examine this in more detail soon.

By the way, the choice of the letter "w" is traditional and meant to suggest "weights", we'll see why when we look at neural nets. The choice of "b" is meant to suggest "bias" - which is the third different connotation of this word in machine learning (the bias of a hypothesis class, bias vs variance, bias of a separator). They are all fundamentally related; they all refer to a difference from a neutral value. To keep the confusion down to a dull roar, we won't call *b* a bias term but are telling you about this so you won't be surprised if you see it elsewhere.

Slide 11.1.11
Sometimes we will use the following trick to simplify the equations. We'll treat the offset as the 0th component of the weight vector **w** and we'll augment the data vector **x** with a 0th component that will always be equal to 1. Then we can write a linear equation as a dot product. When we do this, we will indicate it by using an overbar over the vectors.



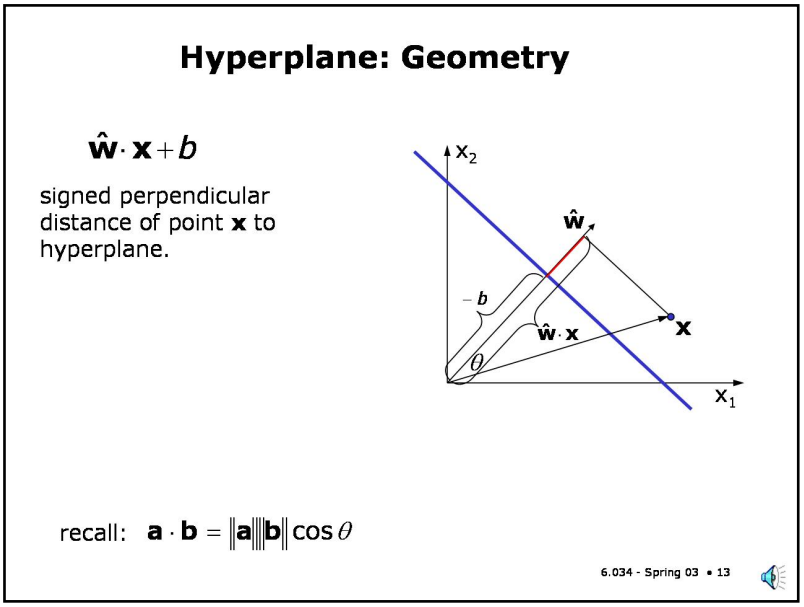
6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved



Slide 11.1.12
First a word on terminology: the equations we will be writing apply to linear separators in *n* dimensions. In two dimensions, such a linear separator is refered to as a "line". In three dimensions, it is called a "plane". These are familiar words. What do we call it in higher dimensions? The usual terminology is **hyperplane**. I know that sounds like some type of fast aircraft, but that's the accepted name.

Let's look at the geometry of a hyperplane a bit more closely. We saw earlier that the offset *b* in the linear separator equation is proportional to the perpendicular distance from the origin to the linear separator and that the constant of proportionality is the reciprocal of the magnitude of the **w** vector (negated). Basically, we can multiply both sides of the equation by any number without affecting the equality. So, there are an infinite set of equations all of which represent the same separator.

If we divide the equation through by the magnitude of **w** we end up with the situation shown in the figure. The normal vector is now unit length (denoted by the hat on the **w**) and the offset *b* is now equal to the perpendicular distance from the origin (negated).



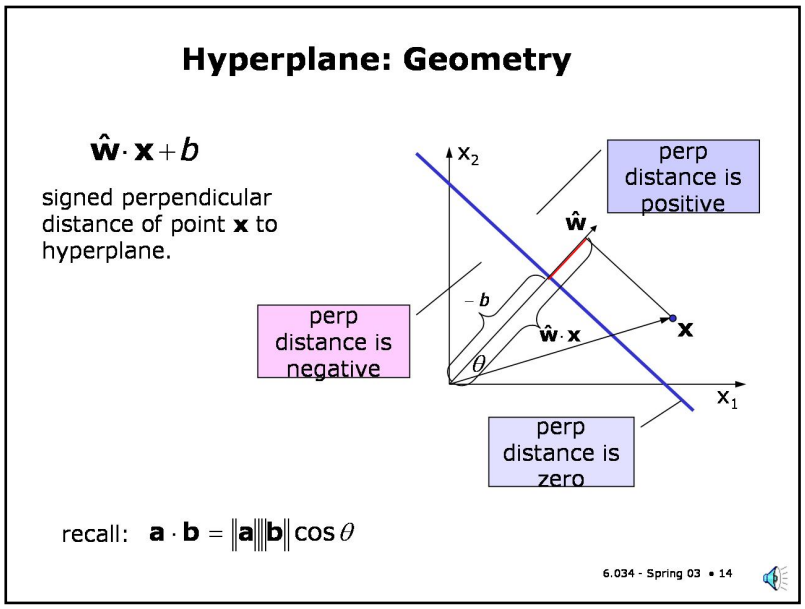
Slide 11.1.13
It's crucial to understand that the quantity **w**-hat dot **x** plus b is the perpendicular distance of point **x** to the linear separator.

If you recall, the geometric interpretation of a dot product **a . b** is that it is a number which is the magnitude of **a** times the magnitude of **b** times the cosine of the angle between the vectors. If one of the vectors, say **a**, has unit magnitude then what we have is precisely the magnitude of the projection of the **b** vector onto the direction defined by **a**. Thus **w**-hat dot **x** is the distance from **x** to the origin measured perpendicular to the hyperplane.

Looking at the right triangle defined by the **w**-hat and the **x** vector, both emanating from the origin, we see that the projection of **x** onto **w**-hat is the length of the base of the triangle, where **x** is the hypotenuse and the base angle is theta.

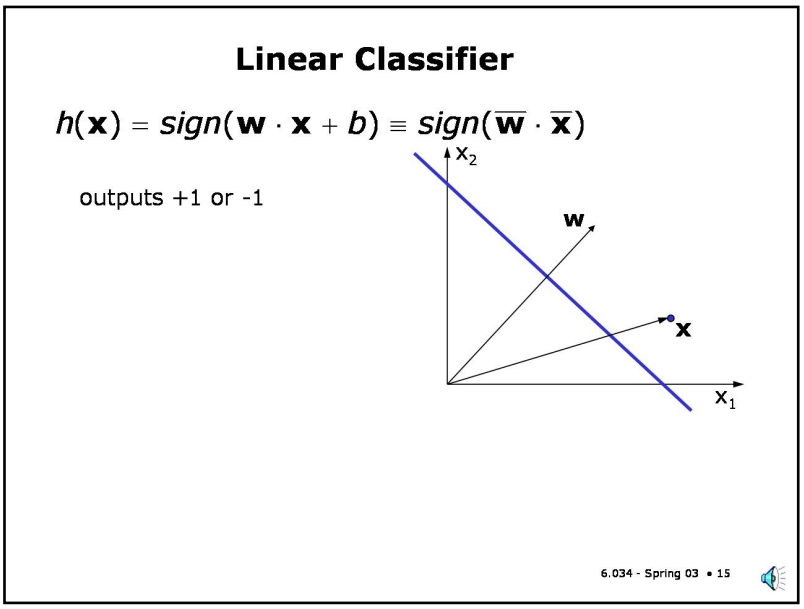
Now, if we subtract out the perpendicular distance to the origin we get the distance of **x** from the hyperplane (rather than from the origin). Note that when theta is 90 degrees (that is, **w** and **x** are perpendicular), the cosine is equal to 0 and the distance is precisely *b* as we expect.

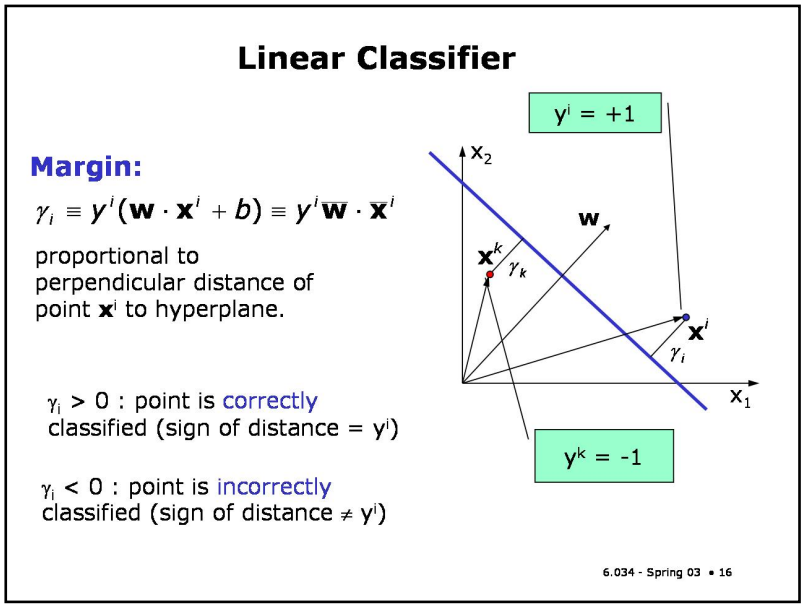
6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved



Slide 11.1.14
This distance measure from the hyperplane is **signed**. It is zero for points on the hyperplane, it is positive for points in the side of the space towards which the normal vector points, and negative for points on the other side. Notice that if you multiply the normal vector **w** and the offset *b* by -1, you get an equation for the same hyperplane but you switch which side of the hyperplane has positive distances.

Slide 11.1.15
We can now exploit the sign of this distance to define a linear classifier, one whose decision boundary is a hyperplane. Instead of using 0 and 1 as the class labels (which was an arbitrary choice anyway) we use the sign of the distance, either +1 or -1 as the labels (that is the values of the y_i).





Slide 11.1.16
A variant of the signed distance of a training point to a hyperplane is the **margin** of the point. The margin (gamma) is the product of the actual signed distance for th epoint and the desired sign of the distance, y_i. If they agree (the point is correctly classified), then the margin is positive; if they disagree (the classification is in error), then the margin is negative.

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

6.034 Notes: Section 11.2

Slide 11.2.1

So far we've talked about how to represent a linear hypothesis but not how to find one. In this slide is the perceptron algorithm, developed by Rosenblatt in the mid 50's. This is not exactly the original form of the algorithm but it is equivalent and it will help us later to see it in this form.

This is a greedy, "mistake driven" algorithm not unlike the Boolean function learning algorithms we saw earlier. We will be using the extended form of the weight and data-point vectors in this algorithm. The extended weight vector is what we are trying to learn.

The first step is to start with an initial value of the weight vector, usually all zeros. Then we repeat the inner loop until all the points are correctly classified using the current weight vector. The inner loop is to consider each point. If the point's margin is positive then it is correctly classified and we do nothing. Otherwise, if it is negative or zero, we have a mistake and we want to change the weights so as to increase the margin (so that it ultimately becomes positive).

Perceptron Algorithm
Rosenblatt, 1956

- Pick initial weight vector (including b), e.g. [0 … 0]
- Repeat until all points correctly classified
  - Repeat for each point
    - Calculate margin ($y^i \overline{\mathbf{w}}\overline{\mathbf{x}}^i$) for point i
    - If margin > 0, point is correctly classified
    - Else change weights to increase margin; change in weight proportional to $y^i \overline{\mathbf{x}}^i$

6.034 - Spring 03 • 1

The trick is how to change the weights. It turns out that using a value proportional to $y\mathbf{x}$ is the right thing. We'll see why, formally, later. For now, let's convince ourselves that it makes sense.

Perceptron Algorithm
Rosenblatt, 1956

- Pick initial weight vector (including b), e.g. [0 … 0]
- Repeat until all points correctly classified
  - Repeat for each point
    - Calculate margin ($y^i \overline{\mathbf{w}}\overline{\mathbf{x}}^i$) for point i
    - If margin > 0, point is correctly classified
    - Else change weights to increase margin; change in weight proportional to $y^i \overline{\mathbf{x}}^i$

- Note that, if $y^i = 1$
  if $x_j^i > 0$ then w_j increased (increases margin)
  if $x_j^i < 0$ then w_j decreased (increases margin)
- And, similarly for $y^i = -1$

6.034 - Spring 03 • 2

Slide 11.2.2

Consider the case in which y is positive; the negative case is analogous. If the jth component of $\mathbf{x}$ is positive then we will increase the corresponding component of $\mathbf{w}$. Note that the resulting effect on the margin is positive. If the jth component of $\mathbf{x}$ is negative then we will decrease the corresponding component of $\mathbf{w}$, and the resulting effect on the margin is also positive.

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Slide 11.2.3

So, each change of **w** increases the margin on a particular point. However, the changes for the different points interfere with each other, that is, different points might change the weights in opposing directions. So, it will not be the case that one pass through the points will produce a correct weight vector. In general, we will have to go around multiple times.

The remarkable fact is that the algorithm is guaranteed to terminate with the weights for a separating hyperplane as long as the data is linearly separable. The proof of this fact is beyond our scope.

Notice that if the data is not separable, then this algorithm is an infinite loop. It turns out that it is a good idea to keep track of the best separator you've seen so far (the one that makes the fewest mistakes) and after you get tired of going around the loop, return that one. This algorithm even has a name (the **pocket** algorithm: see, it keeps the best answer in its pocket...).

Perceptron Algorithm

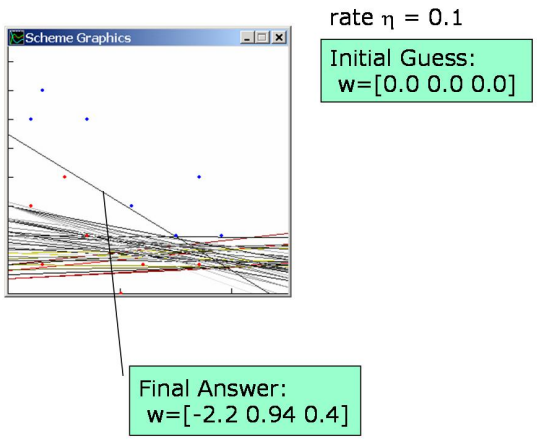
Rosenblatt, 1956

- Pick initial weight vector (including b), e.g. [0 … 0]
- Repeat until all points correctly classified
  - Repeat for each point
    - Calculate margin ($y^i \overline{\mathbf{w}}\overline{\mathbf{x}}^i$) for point i
    - If margin > 0, point is correctly classified
    - Else change weights to increase margin; change in weight proportional to $y^i \overline{\mathbf{x}}^i$

- Note that, if $y^i = 1$
  if $x_j^i > 0$ then w_j increased (increases margin)
  if $x_j^i < 0$ then w_j decreased (increases margin)
- And, similarly for $y^i = -1$
- Guaranteed to find separating hyperplane if one exists
- Otherwise, data are not linearly separable, loops forever

6.034 - Spring 03 • 3

Perceptron Algorithm
Bankruptcy Data



6.034 - Spring 03 • 4

Slide 11.2.4

This shows a trace of the perceptron algorithm on the bankruptcy data. Here it took 49 iterations through the data (the outer loop) for the algorithm to stop. The hypothesis at the end of each loop is shown here. Recall that the first element of the weight vector is actually the offset. So, the normal vector to the separating hyperplane is [0.94 0.4] and the offset is -2.2 (recall that is proportional to the negative perpendicular distance from origin to the line).

Note that the units in the horizontal and vertical directions in this graph are not equal (the tick marks along the axes indicate unit distances). We did this since the range of the data on each axis is so different.

One usually picks some small "rate" constant to scale the change to **w**. It turns out that for this algorithm the value of the rate constant does not matter. We have used 0.1 in our examples, but 1 also works well.

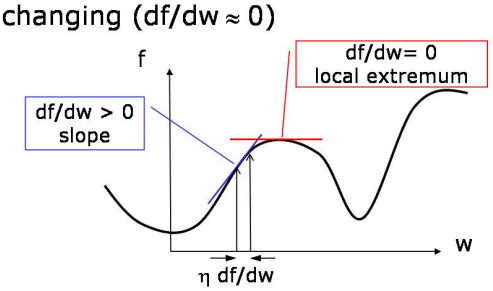
Slide 11.2.5

Let's revisit the issue of why we picked $y\mathbf{x}$ to increment **w** in the perceptron algorithm. It might have seemed arbitrary but it's actually an instance of a general strategy called **gradient ascent** for finding the input(s) that maximize a function's output (or gradient descent when we are minimizing).

The strategy in one input dimension is shown here. We guess an initial value of the input. We calculate the slope of the function at that input value and we take a step that is proportional to the slope. Note that the sign of the slope will tell us whether an increase of the input variable will increase or decrease the value of the output. The magnitude of the slope will tell us how fast the function is changing at that input value. The slope is basically a linear approximation of the function which is valid "near" the chosen input value. Since the approximation is only valid locally, we want to take a small step (determined by the rate constant eta) and repeat.

Gradient Ascent

- Why pick $y^i \overline{\mathbf{x}}^i$ as increment to weights?
- To maximize scalar function of one variable f(w)
  - Pick initial w
  - Change w to w + η df/dw (η > 0, small)
  - until f stops changing (df/dw ≈ 0)



6.034 - Spring 03 • 5

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

We want to stop when the output change is zero (or very small). This should correspond to a point where the slope is zero, which should be a local extremum of the function. This strategy will not guarantee finding the global maximal value, only a local one.

Gradient Ascent/Descent

- To maximize f(**w**) $\nabla_{\mathbf{w}} f = \left[\frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_n}\right]$
  - Pick initial **w**
  - Change **w** to **w** + $\eta \nabla_{\mathbf{w}} f$ ($\eta > 0$, small)
  - until f stops changing ($\nabla_{\mathbf{w}} f \approx 0$)
- Finds local maximum; global maximum if function is globally convex.

6.034 - Spring 03 • 6

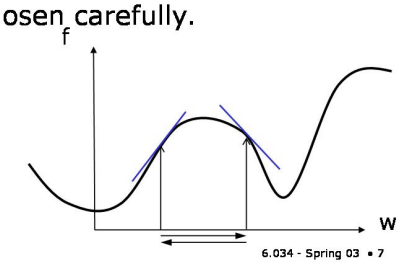
Slide 11.2.6
The generalization of this strategy to multiple input variables is based on the generalization of the notion of slope, which is the **gradient** of the function. The gradient is the vector of first (partial) derivatives of the function with respect to each of the input variables. The gradient vector points in the direction of steepest increase of the function output. So, we take a small step in that direction, recompute the gradient and repeat until the output stops changing. Once again, this will only find us a local maximum of the function, in general. However, if the function is globally convex, then it will find the global optimum.

Slide 11.2.7
In general, the choice of the rate constant (eta), which determines the step size, is fairly critical. Unfortunately, no single value is appropriate for all functions. If one chooses a very conservative small rate, it can take a long time to find a minimum, if one takes too big steps there is no guarantee that the algorithm will even converge to a minimum; it can oscillate as shown in the figure here where the sign of the slope changes and causes a back-and-forth search.

In more sophisticated search algorithms one does a search along the specified direction looking for a value of the step size that guarantees an increase in the function value.

Gradient Ascent/Descent

- To maximize f(**w**) $\nabla_{\mathbf{w}} f = \left[\frac{\partial f}{\partial w_1}, \ldots, \frac{\partial f}{\partial w_n}\right]$
  - Pick initial **w**
  - Change **w** to **w** + $\eta \nabla_{\mathbf{w}} f$ ($\eta > 0$, small)
  - until f stops changing ($\nabla_{\mathbf{w}} f \approx 0$)
- Finds local maximum; global maximum if function is globally convex
- Rate (η) has to be chosen carefully.
  - Too small – slow convergence
  - Too big – oscillation



6.034 - Spring 03 • 7

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Perceptron Training via Gradient Descent

- Maximize sum of margins of misclassified points

$$f(\mathbf{w}) = \sum_{i \text{ misclassified}} y^i \overline{\mathbf{w}\mathbf{x}}^i$$

$$\nabla_{\mathbf{w}} f = \sum_{i \text{ misclassified}} y^i \overline{\mathbf{x}}^i$$

6.034 - Spring 03 • 8

Slide 11.2.8
Now we can see that our choice of increment in the perceptron algorithm is related to the gradient of the sum of the margins for the misclassified points.

Slide 11.2.9
If we actually want to maximize this sum via gradient descent we should sum all the corrections for every misclassified point using a single **w** vector and then apply that correction to get a new weight vector. We can then repeat the process until convergence. This is normally called an **off-line** algorithm in that it assumes access to all the input points.

What we actually did was a bit different, we modified **w** based on each point as we went through the inner loop. This is called an **on-line** algorithm because, in principle, if the points were arriving over a communication link, we would make our update to the weights based on each arrival and we could discard the points after using them, counting on more arriving later.

Another way of thinking about the relationship of these algorithms is that the on-line version is using a (randomized) approximation to the gradient at each point. It is randomized in the sense that rather than taking a step based on the true gradient, we take a step based on an estimate of the gradient based on a randomly drawn example point. In fact, the on-line version is sometimes called "stochastic (randomized) gradient ascent" for this reason. In some cases, this randomness is good because it can get us out of shallow local minima.

Perceptron Training via Gradient Descent

- Maximize sum of margins of misclassified points

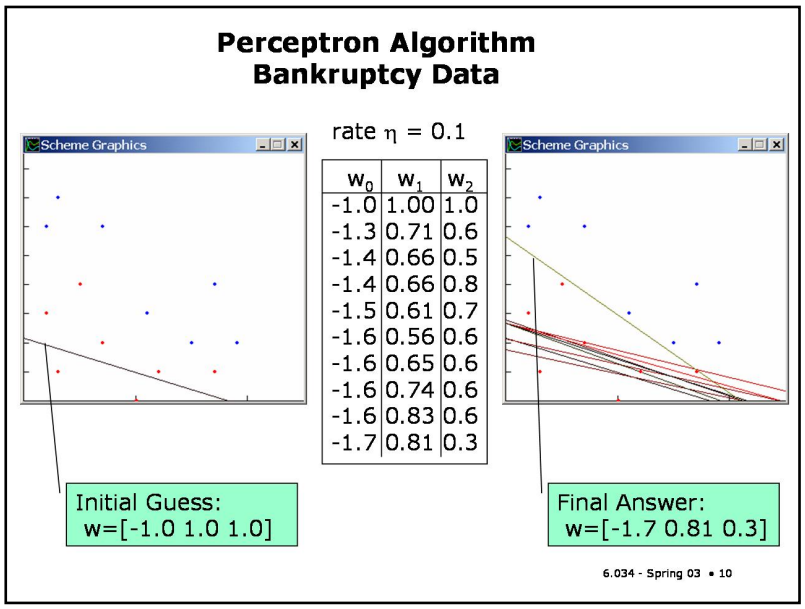
$$f(\mathbf{w}) = \sum_{i \text{ misclassified}} y^i \overline{\mathbf{w}\mathbf{x}}^i$$

$$\nabla_{\mathbf{w}} f = \sum_{i \text{ misclassified}} y^i \overline{\mathbf{x}}^i$$

- Off-line training: Compute gradient as sum over all training points.
- On-line training: Approximate gradient by one of the terms in the sum: $y^i \overline{\mathbf{x}}^i$

6.034 - Spring 03 • 9

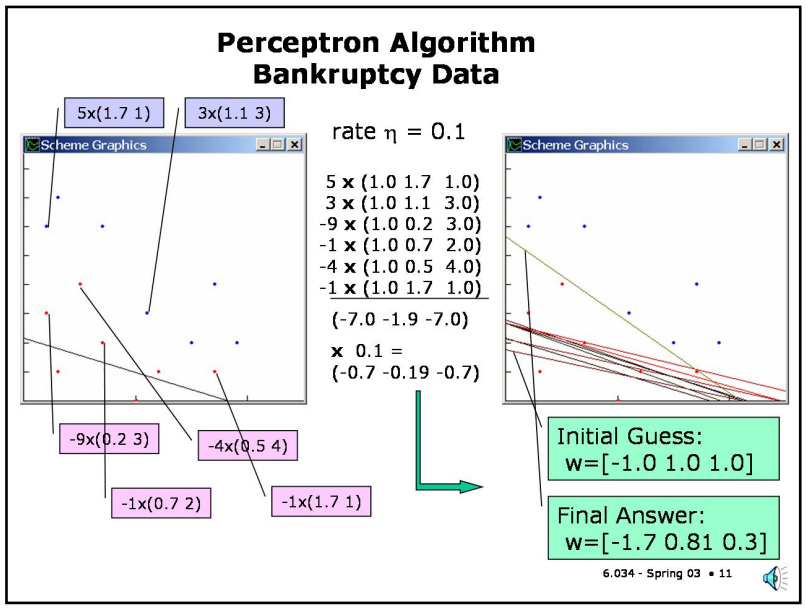
6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

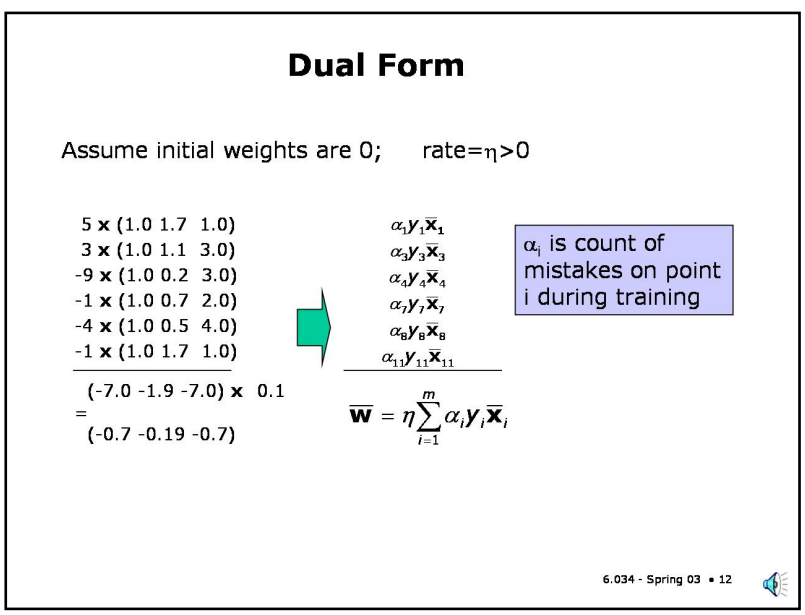


Slide 11.2.10
Here's another look at the perceptron algorithm on the bankruptcy data with a different initial starting guess of the weights. You can see the different separator hypotheses that it goes through. Note that it converges to a different set of weights from our previous example. However, recall that one can scale these weights and get the same separator. In fact these numbers are approximately 0.8 of the ones we got before, but only approximately; this is a slightly different separator.

The perceptron algorithm can be described as a gradient ascent algorithm, but its error criterion is slightly unusual in that there are many separators that all have zero error.

Slide 11.2.11
Recall that the perceptron algorithm starts with an initial guess for the weights and then adds in scaled versions of the misclassified training points to get the final weights. In this particular set of 10 iterations, the points indicated on the left are misclassified some number of times each. For example, the leftmost negative point is misclassified in each iteration except the last one. If we sum up the coordinates of each of these points, scaled by how many times each is misclassified and by the rate constant we get the total change in the weight vector.





Slide 11.2.12
This analysis leads us to a somewhat different view of the perceptron algorithm, usually called the **dual form** of the algorithm. Call the count of how many times point *i* is misclassified, $alpha_i$. Then, assuming the weight vector is initalized to 0s, we can write the final weight vector in terms of these counts and the input data (as well as the rate constant).

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Slide 11.2.13
Since the rate constant does not change the separator we can simply assume that it is 1 and ignore it. Now, we can substitute this form of the weights in the classifier and we get the classifier at the bottom of the slide, which has the interesting property that the data points only appear in dot-products with other data points. This will turn out to be extremely important later; file this one away.

Dual Form

Assume initial weights are 0; rate=η>0

5 **x** (1.0 1.7 1.0)
3 **x** (1.0 1.1 3.0)
-9 **x** (1.0 0.2 3.0)
-1 **x** (1.0 0.7 2.0)
-4 **x** (1.0 0.5 4.0)
-1 **x** (1.0 1.7 1.0)

(-7.0 -1.9 -7.0) **x** 0.1
=
(-0.7 -0.19 -0.7)

$\alpha_1 y^1 \bar{\mathbf{x}}^1$
$\alpha_3 y^3 \bar{\mathbf{x}}^3$
$\alpha_4 y^4 \bar{\mathbf{x}}^4$
$\alpha_7 y^7 \bar{\mathbf{x}}^7$
$\alpha_8 y^8 \bar{\mathbf{x}}^8$
$\alpha_{11} y^{11} \bar{\mathbf{x}}^{11}$

$$\bar{\mathbf{w}} = \eta \sum_{i=1}^{m} \alpha_i y_i \bar{\mathbf{x}}_i$$

α_i is count of mistakes on point i during training

η just scales answer, set to 1

$$h(\mathbf{x}) = sign(\bar{\mathbf{w}} \cdot \bar{\mathbf{x}}) = sign(\sum_{i=1}^{m} \alpha_i y^i \bar{\mathbf{x}}^i \cdot \bar{\mathbf{x}})$$

6.034 - Spring 03 • 13

Perceptron Training
Dual Form

- $\alpha = 0$
- Repeat until all points correctly classified
  - Repeat for each point i
    - Calculate margin $\sum_{j=1}^{m} \alpha_j y^j \bar{\mathbf{x}}^j \cdot \bar{\mathbf{x}}^i$
    - If margin > 0, point is correctly classified
    - Else increment α_i
- Return $\bar{\mathbf{w}} = \sum_{j=1}^{m} \alpha_j y^j \bar{\mathbf{x}}^j$
- If data is not linearly separable, the α_i grow without bound

6.034 - Spring 03 • 14

Slide 11.2.14
We can now restate the perceptron algorithm in this interesting way. The separator is described as a weighted sum of the input points, with alpha$_i$ the weight for point *i*. Initially, set all of the alphas to zero, so the separator has all zero's as coefficients.

Then, for each point, compute its margin with respect to the current separator. If the margin is positive, the point is classified correctly, so do nothing. If the margin is negative, add that point into the weights of the separator. We can do that simply by incrementing the associated alpha.

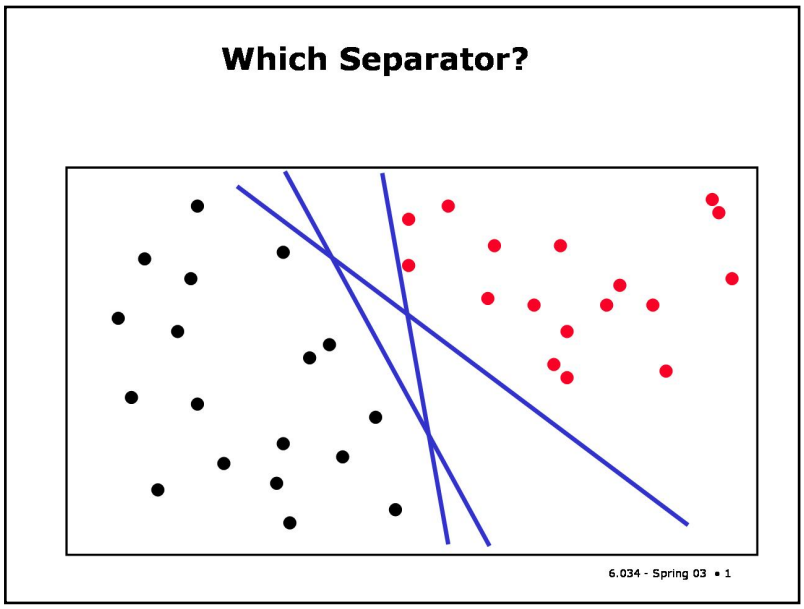
Finally, when all of the points are classified correctly, we return the weighted sum of the inputs as the coefficients fo the separator. Note that if the data is not linearly separable, then the algorithm will loop forever, the alphas growing without bound.

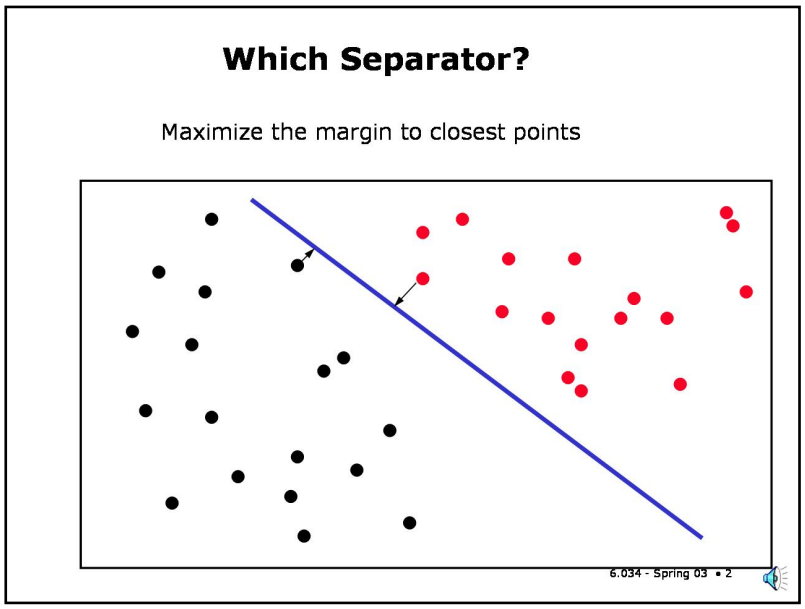
You should convince yourself that this dual form is equivalent to the original. Once again, you may be wondering...so what? I'll say again; file this away. It has surprising consequences.

6.034 Notes: Section 11.3

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Slide 11.3.1
There is no easy way to characterize which particular separator the perceptron algorithm will end up with. In general, there can be many separators for a data set. Even in the tightly constrained bankruptcy data set, we saw two runs of the algorithm with different starting points ended up with slightly different hypotheses. Is there any reason to prefer one separator over the others?



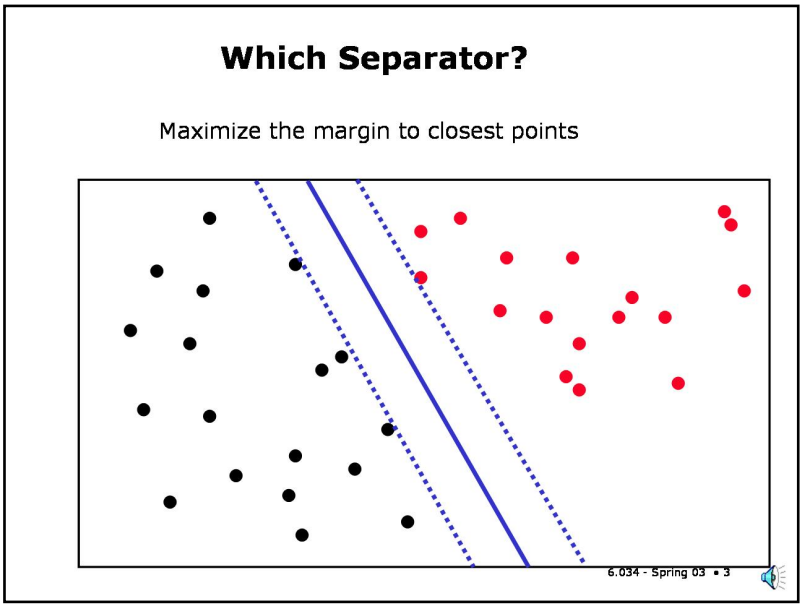


Slide 11.3.2
Yes. One natural choice is to pick the separator that has the maximal margin to its closest points on either side. This is the separator that seems most conservative. Any other separator will be "closer" to one class than to the other. The one shown in this figure, for example, seems like it's closer to the black points on the lower left than to the red ones.

Slide 11.3.3
This one seems safer, no?

Another way to motivate the choice of the maximal margin separator is to see that it reduces the "variance" of the hypothesis class. Recall that a hypothesis has large variance if small changes in the data result in a very different hypothesis. With a maximal margin separator, we can wiggle the data quite a bit without affecting the separator. Placing the separator very close to positive or negative points is a kind of overfitting; it makes your hypothesis very dependent on details of the input data.

Let's see if we can figure out how to find the separator with maximal margin as suggested by this picture.

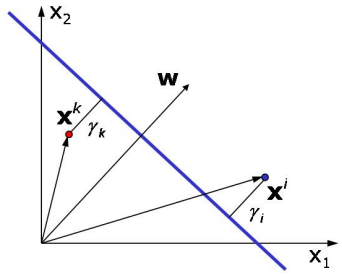


6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Margin of a point

$$\gamma^i \equiv y^i(\mathbf{w} \cdot \mathbf{x}^i + b)$$

- proportional to perpendicular distance of point $\mathbf{x}^i$ to hyperplane



6.034 - Spring 03 • 4

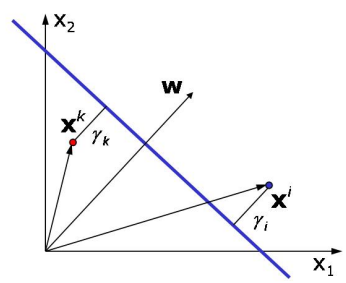
Slide 11.3.4
First we have to define what we are trying to optimize. Clearly we want to use our old definition of margin, but we'll have to deal with a couple of issues first. Note that we're using the **w**, *b* notation instead of **w** bar, because we will end up giving *b* special treatment in the future.

Slide 11.3.5
Remember that any scaling of **w** and *b* defines the same line; but it will result in different values of gamma. To get the actual geometric distance from the point to the separator (called the *geometric margin*), we need to divide gamma through by the magnitude of **w**.

Margin of a point

$$\gamma^i \equiv y^i(\mathbf{w} \cdot \mathbf{x}^i + b)$$

- proportional to perpendicular distance of point $\mathbf{x}^i$ to hyperplane
- geometric margin is $\gamma^i / \|\mathbf{w}\|$



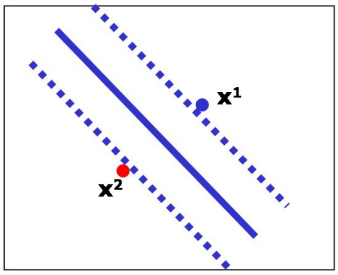
6.034 - Spring 03 • 5

Margin

$$\gamma^i \equiv y^i(\mathbf{w} \cdot \mathbf{x}^i + b)$$

- Scaling **w** changes value of margin but not actual distances to separator (geometric margin)
- Pick the margin to closest positive and negative points to be 1

$$+1(\mathbf{w} \cdot \mathbf{x}^1 + b) = 1$$
$$-1(\mathbf{w} \cdot \mathbf{x}^2 + b) = 1$$



6.034 - Spring 03 • 6

Slide 11.3.6
The next issue is that the we have defined the margin for a point relative to a separator but we don't want to just maximize the margin of some particular single point. We want to focus on one point on each side of the separator, each of which is closest to the separator. And we want to place the separator so that the it is as far from these two points as possible. Then we will have the maximal margin between the two classes.

Since we have a degree of freedom in the magnitude of **w** we're going to just define the margin for each of these points to be 1. (You can think of this 1 as having arbitrary units given by the magnitude of **w**.)

You might be worried that we can't possibly know which will be the two closest points until we know what the separator is. It's a reasonable worry, and we'll sort it out in a couple of slides.

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Slide 11.3.7
Having chosen these margins, we can add the two equations to get that the projection of the weight vector on the difference between the two chosen data points has magnitude 2. This is obvious from the setup, but it's nice to see it follows.

Then, we divide through by the magnitude of the weight vector and we have a simple expression for the margin, simply 2 over the magnitude of **w**.

Margin

- Pick the margin to closest positive and negative points to be 1

$$+1(\mathbf{w}\cdot\mathbf{x}^1+b)=1$$
$$-1(\mathbf{w}\cdot\mathbf{x}^2+b)=1$$

- Combining these

$$\mathbf{w}\cdot(\mathbf{x}^1-\mathbf{x}^2)=2$$

- Dividing by length of **w** gives perpendicular distance between lines (2 **x** geometric margin)

$$\frac{\mathbf{w}}{\|\mathbf{w}\|}\cdot(\mathbf{x}^1-\mathbf{x}^2)=\frac{2}{\|\mathbf{w}\|}$$

6.034 - Spring 03 • 7

Picking w to Maximize Margin

- Pick **w** to maximize geometric margin

$$\frac{2}{\|\mathbf{w}\|}$$

- or, equivalently, minimize

$$\|\mathbf{w}\|=\sqrt{\mathbf{w}\cdot\mathbf{w}}$$

- or, equivalently, minimize

$$\frac{1}{2}\|\mathbf{w}\|^2=\frac{1}{2}\mathbf{w}\cdot\mathbf{w}=\frac{1}{2}\sum_j w_j^2$$

6.034 - Spring 03 • 8

Slide 11.3.8
So, we want to pick **w** to maximize the geometric margin, that is, to maximize 2 over the magnitude of **w**. To maximize this expression, we want to minimize the magnitude of **w**. If we minimize 1/2 the magnitude squared that is completely equivalent in effect but simpler analytically.

Of course, this is not enough, since we could simply pick **w** = 0 which would be completely useless.

Slide 11.3.9
We'd like to find the **w** that specifies the maximum margin separator. To be a separator, **w** needs to classify the points correctly. So, we'll maximize the margin, subject to a set of constraints that require the points to be classified correctly. We will require each point in the training set to have a margin greater than or equal to 1. Requiring the margins to be positive will ensure that they are classified correctly. Requiring them to be greater than or equal to 1 will ensure that the margin of the closest points wil be greater than or equal to 1. The fact that we are minimizing the magnitude of **w** will force the margins to be as small as possible, so that in fact the margins of the closest points will equal 1.

Picking w to Maximize Margin

- Pick **w** to maximize geometric margin

$$\frac{2}{\|\mathbf{w}\|}$$

- or, equivalently, minimize

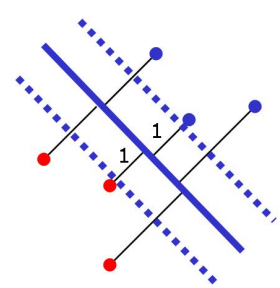
$$\frac{1}{2}\|\mathbf{w}\|^2=\frac{1}{2}\mathbf{w}\cdot\mathbf{w}=\frac{1}{2}\sum_j w_j^2$$

- while classifying points correctly

$$y^i(\mathbf{w}\cdot\mathbf{x}^i+b)\geq 1$$

- or, equivalently,

$$y^i(\mathbf{w}\cdot\mathbf{x}^i+b)-1\geq 0$$



6.034 - Spring 03 • 9

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Constrained Optimization

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 \text{ subject to } y^i(\mathbf{w}\cdot\mathbf{x}^i + b) - 1 \geq 0, \ \forall_i$$

6.034 - Spring 03 • 10

Slide 11.3.10
So, to summarize, we have defined a constrained optimization problem as shown here. It involves minimizing a quadratic function subject to a set of linear constraints. These kinds of optimization problems are very well studied. When the function to be minimized is linear, it is a particularly easy case that can be solved by a "linear programming" algorithm. In our case, it's a bit more complicated.

Slide 11.3.11
The standard approach to solving this type of problem is to convert it to an unconstrained optimization problem by incorporating the constraints as additional terms in the function to be minimized. Each of the constraints is multiplied by a weighting term $alpha_i$. Think of these terms as penalty terms that will penalize values of **w** that do not satisfy the constraints.

Constrained Optimization

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 \text{ subject to } y^i(\mathbf{w}\cdot\mathbf{x}^i + b) - 1 \geq 0, \ \forall_i$$

Convert to unconstrained optimization by incorporating the constraints as an additional term

$$\min_{\mathbf{w}}\left(\frac{1}{2}\|\mathbf{w}\|^2 - \sum_i \alpha_i\left[y^i(\mathbf{w}\cdot\mathbf{x}^i + b) - 1\right]\right) \quad \alpha_i \geq 0, \forall_i$$

6.034 - Spring 03 • 11

Constrained Optimization

$$\min_{\mathbf{w}} \frac{1}{2}\|\mathbf{w}\|^2 \text{ subject to } y^i(\mathbf{w}\cdot\mathbf{x}^i + b) - 1 \geq 0, \ \forall_i$$

Convert to unconstrained optimization by incorporating the constraints as an additional term

$$\min_{\mathbf{w}}\left(\frac{1}{2}\|\mathbf{w}\|^2 - \sum_i \alpha_i\left[y^i(\mathbf{w}\cdot\mathbf{x}^i + b) - 1\right]\right) \quad \alpha_i \geq 0, \forall_i$$

To minimize expression:
- minimize first (original) term, and
- maximize second (constraint) term
  - since $\alpha_i > 0$, encourages constraints to be satisfied
  - but we want least "distortion" of original term…

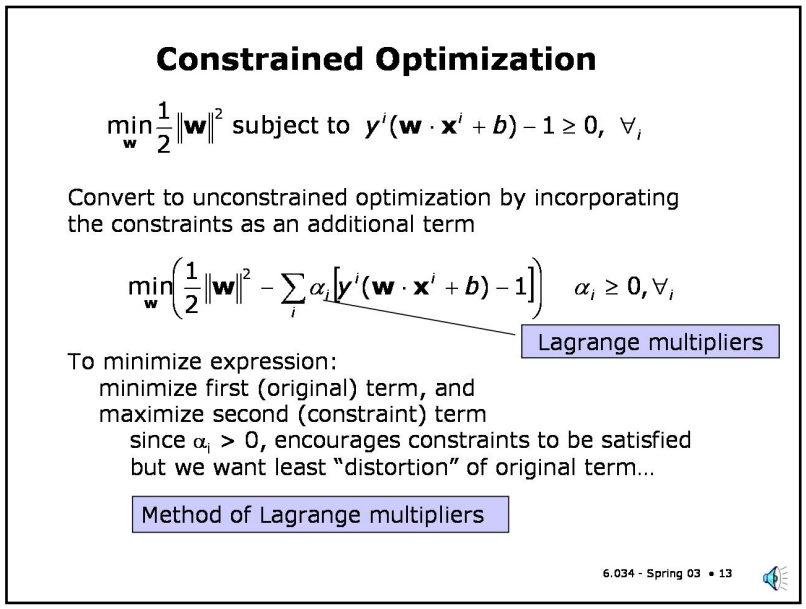
6.034 - Spring 03 • 12

Slide 11.3.12
To minimize the combined expression we want to minimize the first term (the magnitude of the weight vector squared) but we want to maximize the constraint term since it is negated. Since $alpha_i > 0$, making the constraint terms bigger encourages them to be satisfied (we want the margins to be bigger than 1).

However, the bigger the constraint term, the farther we move from the original minimal value of **w**. In general we want to minimize this "distortion" of the original problem. We want to introduce just enough distortion to satisfy the constraints. We'll look at this in more detail momentarily.

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

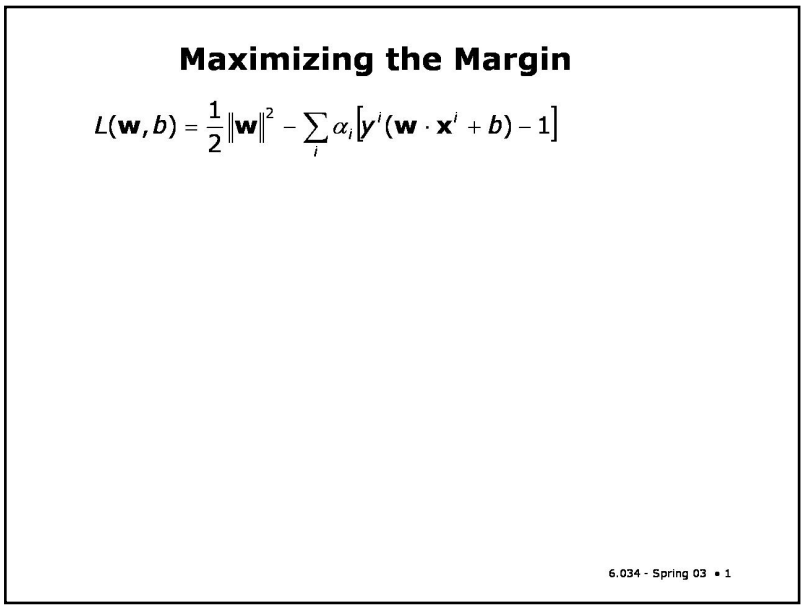
Slide 11.3.13
This method we have begun to outline here is called the **method of Lagrange multipliers** and the $alpha_i$ are the individual Lagrange multipliers.



6.034 Notes: Section 11.4

Slide 11.4.1
The details of solving a Lagrange multiplier problem are a little bit complicated. We've described the process in quite a bit of detail in an optional section of this chapter. But we are going to go through the derivation at a somewhat abstract level here, because it gives us some insights and intuitions about the resulting solution.

We have an expression, $L(\mathbf{w},b)$, that also involves parameters alpha. If we knew what the values of alpha should be, we could just fix them, minimize L with respect to $\mathbf{w}$ and b, and be done. The big problem is that we don't know what the alphas are supposed to be.



6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Maximizing the Margin

$$L(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i \alpha_i \left[y^i(\mathbf{w} \cdot \mathbf{x}^i + b) - 1\right]$$

Minimized when: $\mathbf{w}^* = \sum_i \alpha_i y^i \mathbf{x}^i$ $\quad \sum_i \alpha_i y^i = 0$

6.034 - Spring 03 • 2

Slide 11.4.2
So, we're going to start by imagining that we know what we want the alphas to be. We'll hold them constant for now, and figure out what values of **w** and *b* would optimize *L* for those fixed alphas. We can do this by taking the partial derivatives of *L* with respect to **w** and *b* and setting them to zero, getting two constraints. We find that the best value of **w**, **w*** is a weighted sum of the input points (in the same form as the dual perception); and we get an extra constraint that the sum of the alphas for the positive points has to equal the sum of the alphas for the negative points.

Slide 11.4.3
We can substitute this expression for the optimal **w**'s back into our original expression for L, getting L as a function of alpha. Now we have an expression involving only alphas, which we don't know, and x's and y's, which we do know. This function is known as the *dual Lagrangian*. One of the most important things about it, from our perspective, is that the feature vectors only appear in dot products with other feature vectors. We'll come back to this point later on.

Maximizing the Margin

$$L(\mathbf{w}, b) = \frac{1}{2}\|\mathbf{w}\|^2 - \sum_i \alpha_i \left[y^i(\mathbf{w} \cdot \mathbf{x}^i + b) - 1\right]$$

Minimized when: $\mathbf{w}^* = \sum_i \alpha_i y^i \mathbf{x}^i$ $\quad \sum_i \alpha_i y^i = 0$

Substituting **w*** into L yields dual Lagrangian:

$$L(\alpha) = \sum_{i=1}^{m} \alpha_i - \frac{1}{2}\sum_{i=1}^{m}\sum_{k=1}^{m} \alpha_i \alpha_k y_i y_k \mathbf{x}_i \mathbf{x}_k$$

Only dot products of the feature vectors appear

6.034 - Spring 03 • 3

Dual Lagrangian

$$\max_{\alpha} L(\alpha) \text{ subject to } \sum_i \alpha_i y^i = 0 \text{ and } \alpha_i \geq 0, \forall i$$

6.034 - Spring 03 • 4

Slide 11.4.4
Now, it's time to pick the best values for the alphas. We do so (for reasons that you'll have to learn in a math class) by choosing the alpha values that maximize this expression. We will retain the constraints that the sum of the alpha values for positive points is equal to the sum of the alpha values for negative points, and that the alphas must be positive.

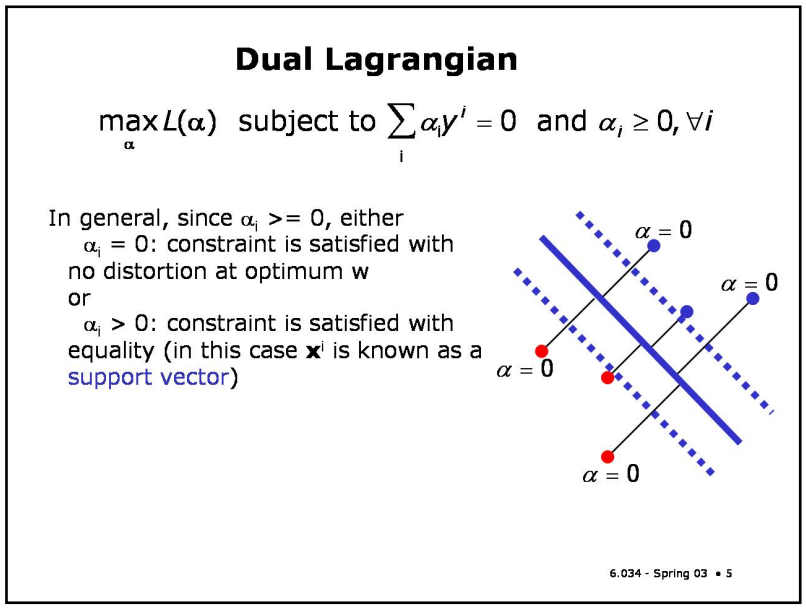
Note that we will be solving for *m* alphas. We started with *n+1* (the number of features, plus one) variables in the original Lagrangian and now we have *m* (the number of data points) variables in the dual Lagrangian. For the low-dimensional examples we have been dealing with this seems like a horrible tradeoff. We will see in the next lecture that this can be a very good tradeoff in some circumstances.

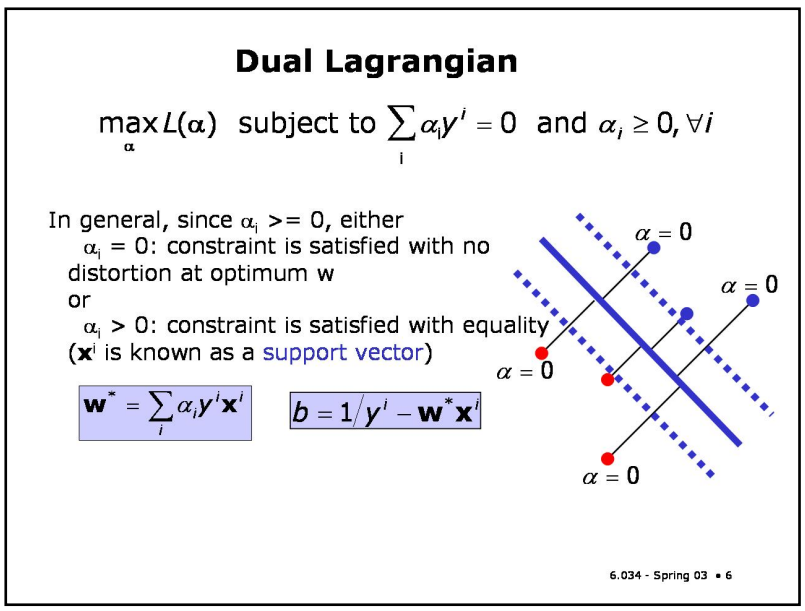
We have two constraints, but they are much simpler. One constraint is simply that the alphas be non-negative---this is required because our original constraints were >= inequalities. The constraint on the alphas comes from the setting to zero the derivative of the Lagrangian with respect to the offset *b*.

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

This problem is not trivial to solve in general; we'll talk more about this later. For now, let us assume that we can solve it and get the optimal values of alphas.

Slide 11.4.5
In the solution, most of the alphas will be zero, corresponding to data points that do not provide binding constraints on the choice of the weights. A few of the data points will have their alphas be nonzero; they will all satisfy their constraints with equality (that is, their margin is equal to 1). These are called **support vectors** and they are the ones used to define the maximum margin separator. You could remove all the other data points and still get the same separator. Because the sparsity of support vectors is so important, this learning method is called a **support vector machine**, or SVM.





Slide 11.4.6
Given the optimal alphas, we can compute the weights. but this time, the coefficients in the sum are the Lagrange multipliers, the alphas, which are mostly zero. This means that the equation of the maximum margin separator depends only on the handful of data points that are closest to it. It makes sense that all the rest of the points would be irrelevant.

We can use the fact that at the support vectors the constraints hold with equality to solve for the value of the offset *b*. Each such constraint can be used to solve for this scalar.

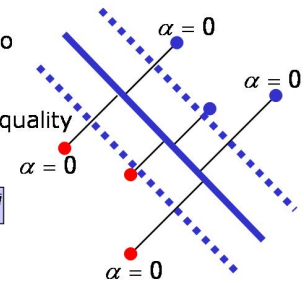
6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Slide 11.4.7
We have not discussed actual algorithms for finding the maxima of the dual Lagrangian. It turns out that the optimization problem we defined is a relatively simple form of the general class of **quadratic programming** problems, which are known to (a) have a unique maximum and (b) can be found using existing algorithms. A number of variations on these algorithms exist but they are beyond our scope.

Dual Lagrangian

$$\max_{\alpha} L(\alpha) \text{ subject to } \sum_i \alpha_i y^i = 0 \text{ and } \alpha_i \geq 0, \forall i$$

In general, since α_i >= 0, either
α_i = 0: constraint is satisfied with no distortion at optimum w
or
α_i > 0: constraint is satisfied with equality
($\mathbf{x}^i$ is known as a support vector)



$$\mathbf{w}^* = \sum_i \alpha_i y^i \mathbf{x}^i \qquad b = 1/y^i - \mathbf{w}^* \mathbf{x}^i$$

- Has a unique maximum vector
- Can be found using quadratic programming or gradient ascent

6.034 - Spring 03 • 7

SVM Classifier

- Given unknown vector **u**, predict class (1 or -1) as follows:

$$h(\mathbf{u}) = sign\left(\sum_{i=1}^{k} \alpha_i y^i \mathbf{x}^i \cdot \mathbf{u} + b\right)$$

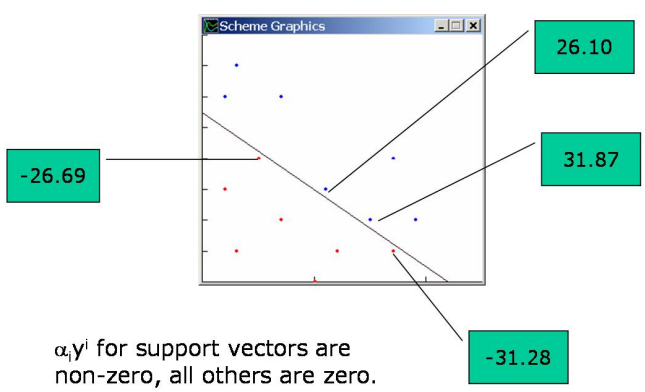
- The sum is over *k* support vectors

6.034 - Spring 03 • 8

Slide 11.4.8
With the values of the optimal alpha$_i$'s and *b* in hand, and the knowledge of how **w** is defined, we now have a classifier that we can use on unknown points. Crucially, notice that once again, the only thing we care about are the dot products of the unknown vector with the data points.

Slide 11.4.9
Here's the result of running a quadratic programming algorithm to find the maximal margin separator for the bankruptcy example. Note that only four points have non-zero alpha's. They are the closest points to the line and are the ones that actually define the line.

Bankruptcy Example



$\alpha_i y^i$ for support vectors are non-zero, all others are zero.

6.034 - Spring 03 • 9

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Key Points

- Learning depends only on dot products of sample pairs. Recognition depends only on dot products of unknown with samples.

6.034 - Spring 03 • 10

Slide 11.4.10

Let's highlight once again a few of the key points about SVM training and classification. First and foremost, and at the risk of repeating myself, recall that the training and classification of SVMs depends only on the value of the dot products of data vectors. That is, if we have a way of getting the dot products, the computation does not otherwise depend explicitly on the dimensionality of the feature space.

Slide 11.4.11

The fact that we only need dot products (as we will see next) means that we will be able to substitute more general functions for the traditional dot product operation to get more powerful classifiers without really changing anything in the actual training and classification procedures.

Key Points

- Learning depends only on dot products of sample pairs. Recognition depends only on dot products of unknown with samples.
- Exclusive reliance on dot products enables approach to non-linearly-separable problems.

6.034 - Spring 03 • 11

Key Points

- Learning depends only on dot products of sample pairs. Recognition depends only on dot products of unknown with samples.
- Exclusive reliance on dot products enables approach to non-linearly-separable problems.
- The classifier depends only on the support vectors, not on all the training points.

6.034 - Spring 03 • 12

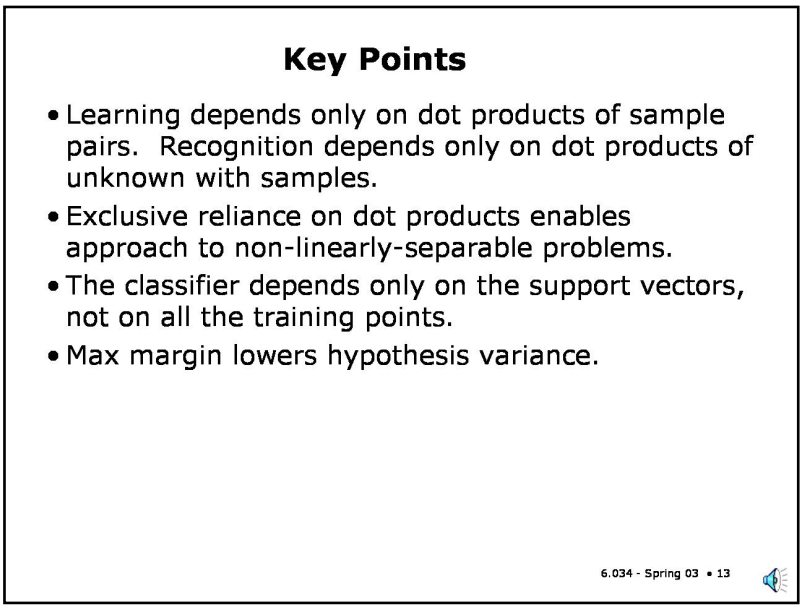
Slide 11.4.12

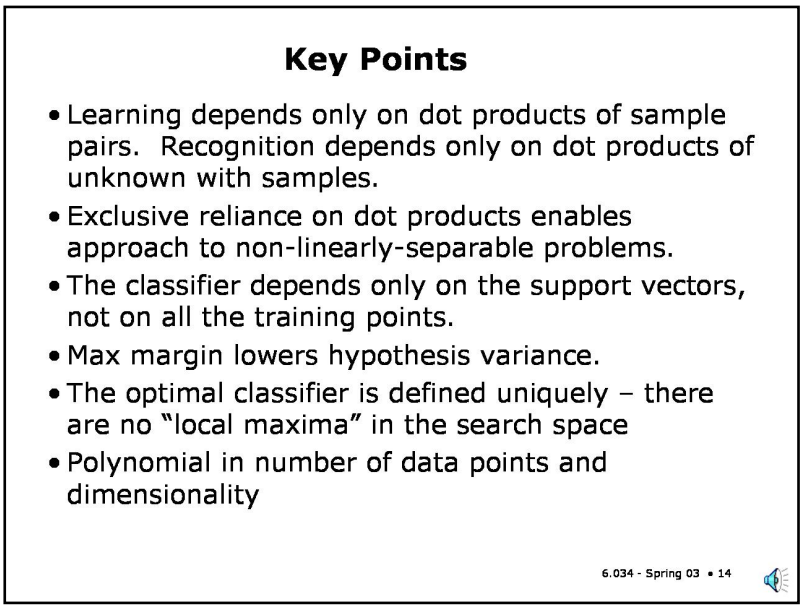
Another point to remember is that the resulting classifier does not (in general) depend on all the training points but only on the ones "near the margin", that is, those that help define the boundary between the two classes.

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Slide 11.4.13

The maximum margin constraint helps reduce the variance of the SVM hypotheses. Insisting on a minimum magnitude weight vector drastically cuts down on the size of the hypothesis class and helps avoid overfitting.





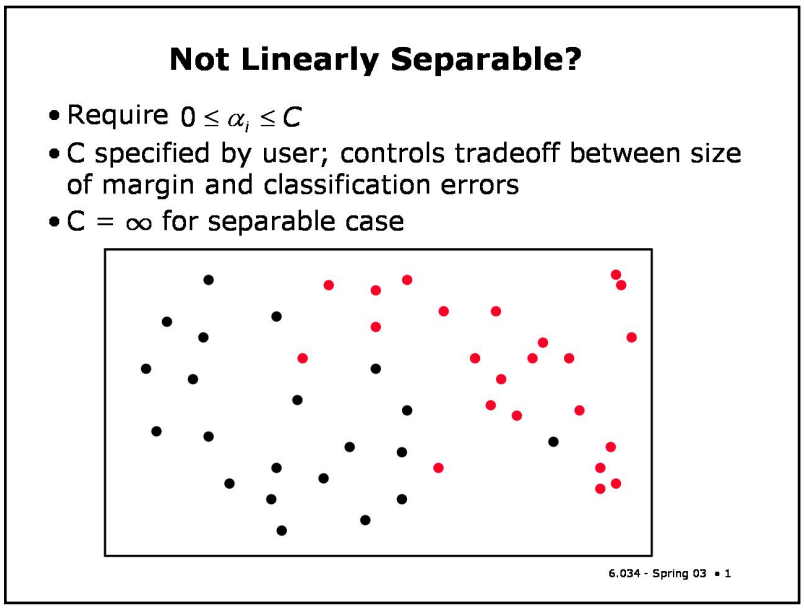
Slide 11.4.14

Finally, we should keep firmly in mind that the SVM training process guarantees a unique global maximum. And it runs in time polynomial in the number of data points and the dimensionality of the data.

6.034 Notes: Section 11.5

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Slide 11.5.1

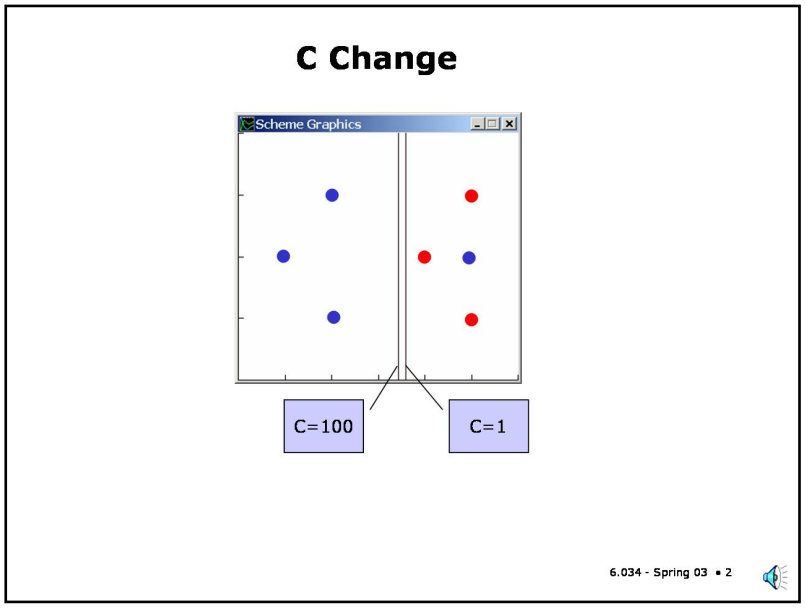


Thus far, we have only been talking about the linearly separable case. What happens for the case in which we have a "nearly separable" problem? That is, some "noise points" that are bound to be misclassified by a linear separator.

It is useful to think about the behavior of the dual perceptron In this algorithm, the value of the Lagrange multiplier, $alpha_i$ for a point is incremented proportionally to its distance to the separator. In fact, if the point is classified correctly, no change is made to the multiplier. We can see that if point i stubbornly resists being classified, then the value of $alpha_i$ will continue to grow without bounds.

So, one strategy for dealing with these noise points is to limit the maximal value of any of the $alpha_i$'s to some C. And, furthermore, to ignore the points with this maximal value when computing the margin. Clearly, if we ignore enough points, we can always get back to a linearly separable problem. By choosing a large value of C, we will work very hard at correctly classifying all the points, a low value of C will allow us to give up more easily on many of the points.

We can get some intuition about C by thinking about the dual perceptron. Remember that in the perceptron, the magnitude of alpha is related to how many times a point has been misclassified. By putting a limit on alpha, we limit the effect any individual "outlier" point can have on the definition of the separator. We say that we are willing to live with some misclassifications.



Slide 11.5.2

This simple example shows how changing C causes the geometric margin to change. For low values of C the margin between positives and negatives is reduced as the algorithm tries to "capture" the wayward point off on the right. For high values of C, the separator is closer to where it would be if the "outlier" were not there.

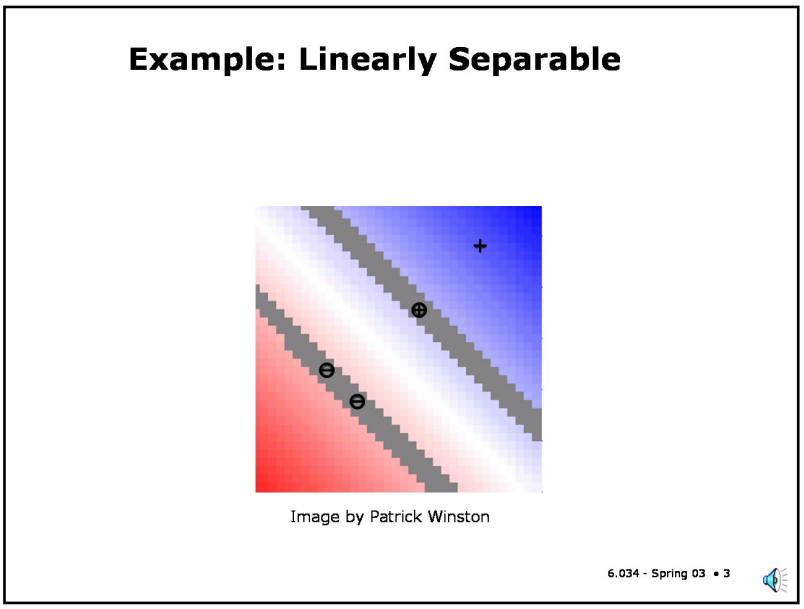
6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

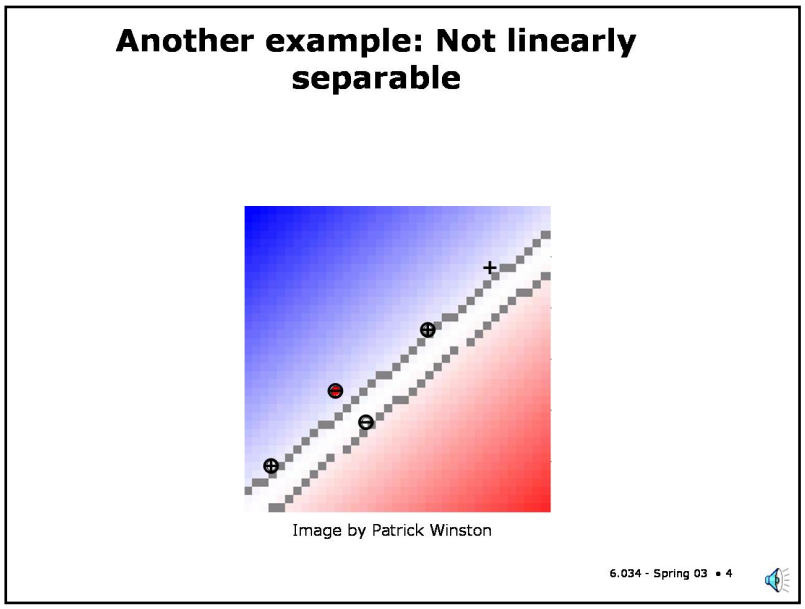
Slide 11.5.3

Here is an example of a separator on a simple data set with four points, which are linearly separable. The colors show the result returned by the classification function on each point in the space. Gray means near 1 or -1. The more intense the blue, the more positive the result; the more intense the red, the more negative. Points lying between the two gray lines return values between -1 and +1.

Note that only three of the four samples are actually used to define **w**, the ones circled. The other plus sample might as well not be there; its coefficient alpha is zero.

The samples that are actually used are the **support vectors**.





Slide 11.5.4

The next example is the same as the previous example, but with the addition of another plus sample in the lower left corner. There are several points of interest.

First, the optimization has failed to find a separating line, as indicated by the minus sample surrounded by a red disk. The alphas were bounded and so the contribution of this misclassified point is limited and the algorithm converges to a global optimum.

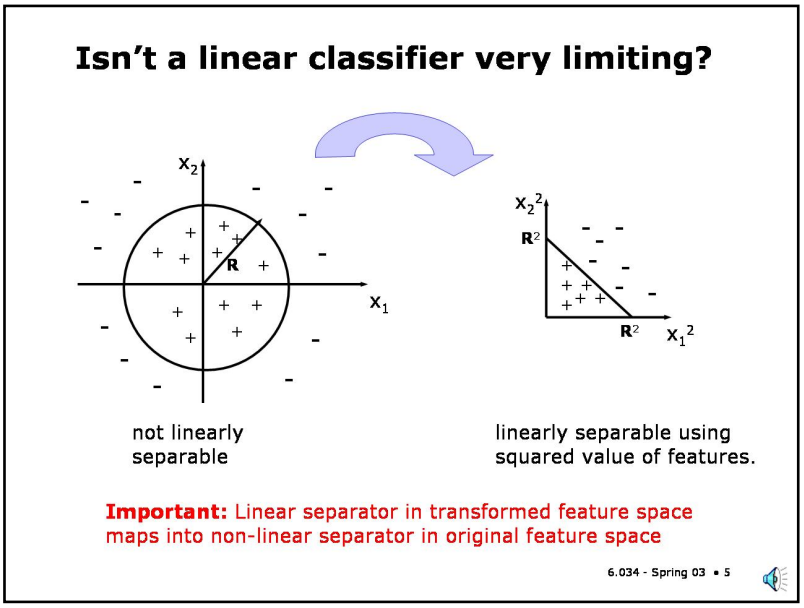
Second, the added point produced quite a different solution. The algorithm is looking for best possible dividing line; a tradeoff between margin and classification error defined by C. If we had kept a solution close to the one in the previous slide, the rogue plus point would have been misclassified by a lot, while with this solution we have reduced the misclassification margin substantially.

Slide 11.5.5

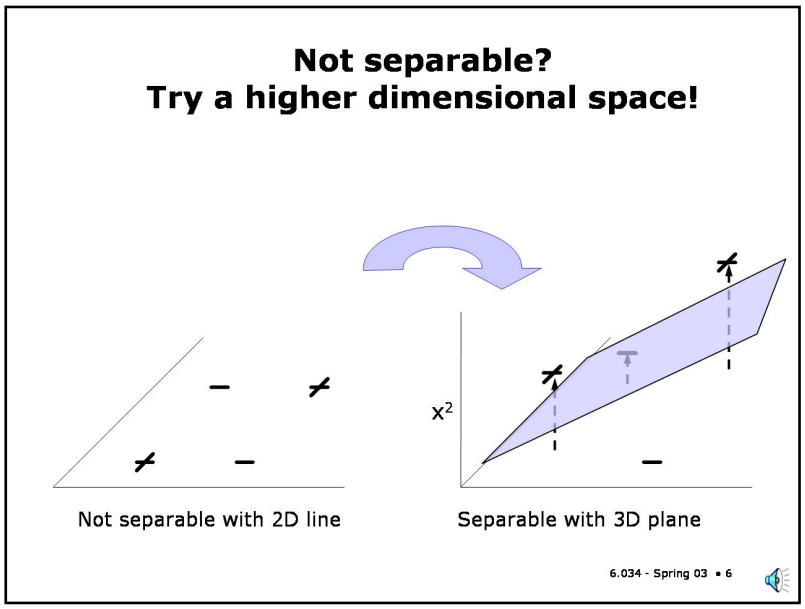
However, even if we provide a mechanism for ignoring noise points, aren't we really limited by a linear classifier? Well, yes.

However, in many cases, if we transform the feature values in a non-linear way, we can transform a problem that was not linearly separable into one that is. This example, shows that we can create a circular separator by finding a linear classifier in a feature space defined by the squares of the original feature values. That is, we can obtain a non-linear classifier in the original space by finding a linear classifier in a transformed space.

Hold that thought.



6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved



Slide 11.5.6

Furthermore, when training samples are not separable in the original space they may be separable if you perform a transformation into a higher dimensional space, especially one that is a non-linear transformation of the input space.

For the example shown here, in the original feature space, the samples all lie in a plane, and are not separable by a straight line. In the new space, the samples lie in a three dimensional space, and happen to be separable by a plane.

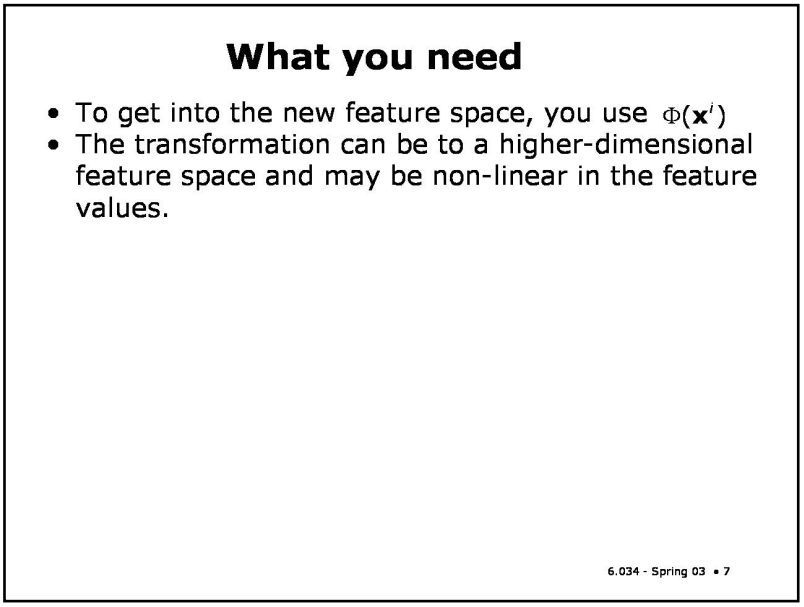
The heuristic of moving to a higher dimensional space is general, and does not depend on using SVMs.

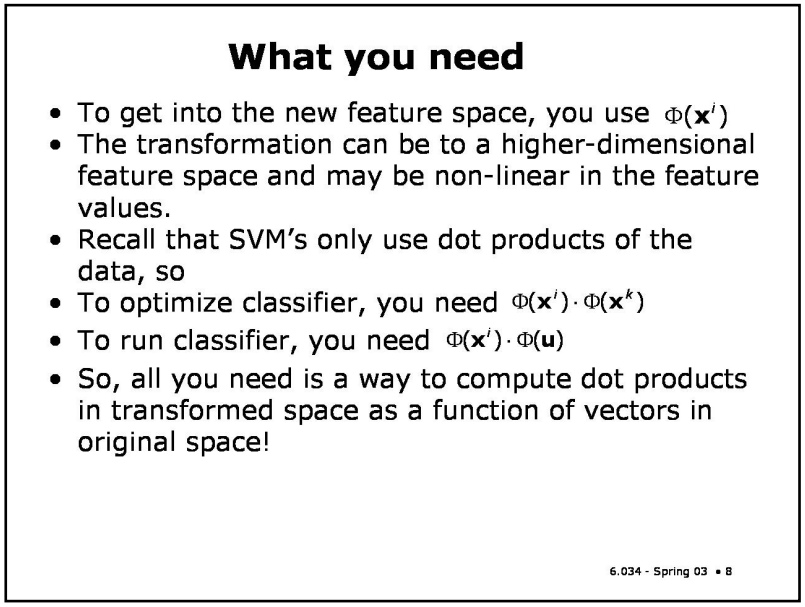
However, we will see that the support vector approach lends itself to movement into higher dimensional spaces because of the exclusive dependence of the support vector approach on dot products for learning and subsequent classification.

Slide 11.5.7

First, suppose there is a function, phi, that puts the vectors into another, higher-dimensional space, which will also typically involve a non-linear mapping of the feature values. In general, the higher the dimensionality, the more likely there will be a separating hyperplane.

By moving to a higher-dimensional feature space, we are also moving to a bigger hypothesis class, and so we might be worried about overfitting. However, because we are finding the maximum margin separator, the danger of overfitting is greatly reduced.





Slide 11.5.8

Even if we aren't in danger of overfitting, there might be computational problems if we move into higher dimensional spaces. In real applications, we might want to move to orders of magnitude more features, or even (in some sense) infinitely many features! We'll need a clever trick to manage this...

You have learned that to work in any space with the support vector approach, you will need (only) the dot products of the samples to train and you will need the dot products of the samples with unknowns to classify.

Note that you don't need anything else. So, all we need is a way of computing the dot product between the transformed feature vectors.

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Slide 11.5.9

Let's assume that we have a function that allows us to compute the dot products of the transformed vectors in a way that depends only on the original feature vectors and not directly on the transformed vectors. We will call this the **kernel function**. (This usage of the term "kernel" is related to kernel functions we saw in regression; they are both about measuring effective distances between points in different spaces.)

Then you do not need to know how to do the transformations themselves! This is why the support-vector approach is so appealing. The actual transformations may be computationally intractable, or you may not even know how to do the transformations at all, but you can still learn and classify without ever moving explicitly up into the high-dimensional space.

The "Kernel Trick"

- If dot products can be efficiently computed by
  $\Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{x}^k) = K(\mathbf{x}^i, \mathbf{x}^k)$
- Then, all you need is a function on low-dim inputs
  $K(\mathbf{x}^i, \mathbf{x}^k)$
- You don't need ever to construct high-dimensional
  $\Phi(\mathbf{x}^i)$

6.034 - Spring 03 • 9

Standard Choices For Kernels

- No change (linear kernel)

$$\Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{x}^k) = K(\mathbf{x}^i, \mathbf{x}^k) = \mathbf{x}^i \cdot \mathbf{x}^k$$

6.034 - Spring 03 • 10

Slide 11.5.10

So now we need to find some phis (mappings from low to high-dimensional space) that have a convenient kernel function associated with them. The simplest case is one where phi is the identity function and K is just the dot product.

Slide 11.5.11

One such other kernel function is the dot product raised to a power; the actual power is a parameter of the learning algorithm that determines the properties of the solution.

Standard Choices For Kernels

- No change (linear kernel)

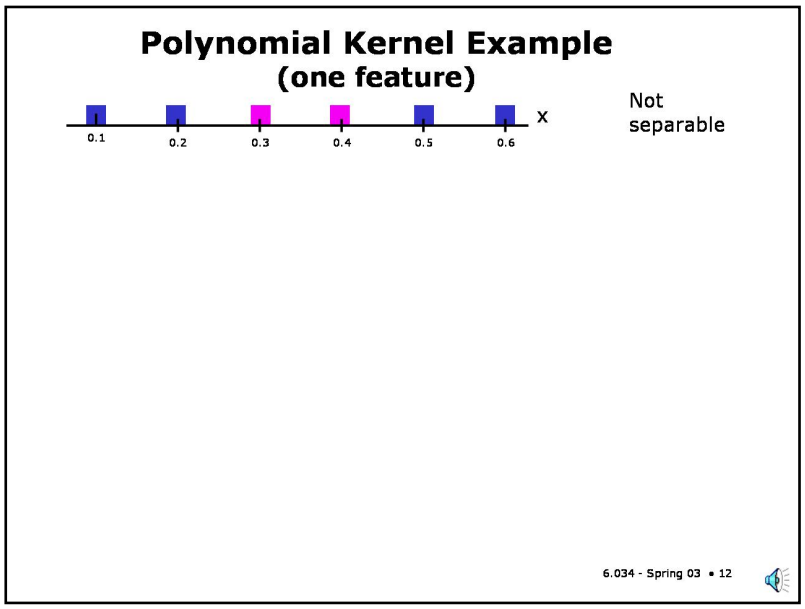
$$\Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{x}^k) = K(\mathbf{x}^i, \mathbf{x}^k) = \mathbf{x}^i \cdot \mathbf{x}^k$$

- Polynomial kernel (n^{th} order)

$$K(\mathbf{x}^i, \mathbf{x}^k) = (1 + \mathbf{x}^i \cdot \mathbf{x}^k)^n$$

6.034 - Spring 03 • 11

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved



Slide 11.5.12

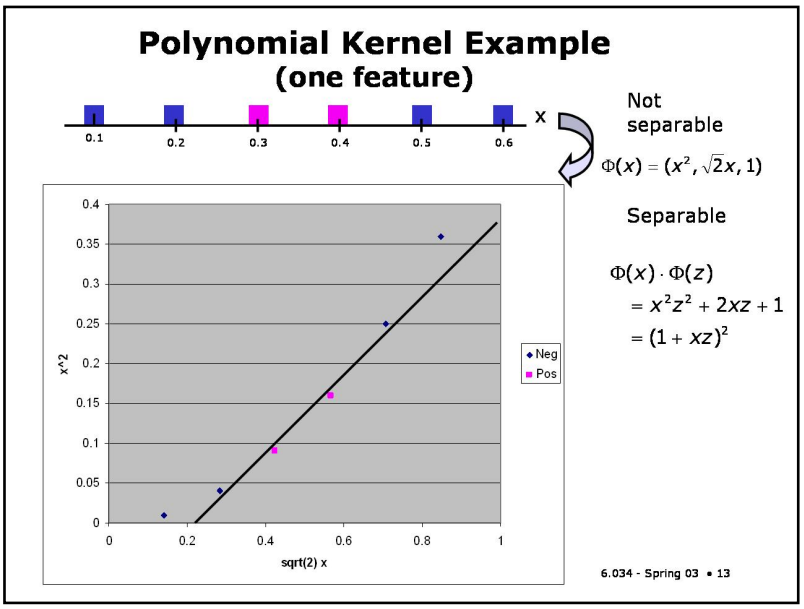
Let's look at a simple example of using a polynomial kernel. Consider the one dimensional problem shown here, which is clearly not separable. Let's map it into a higher dimensional feature space using the polynomial kernel of second degree (n=2).

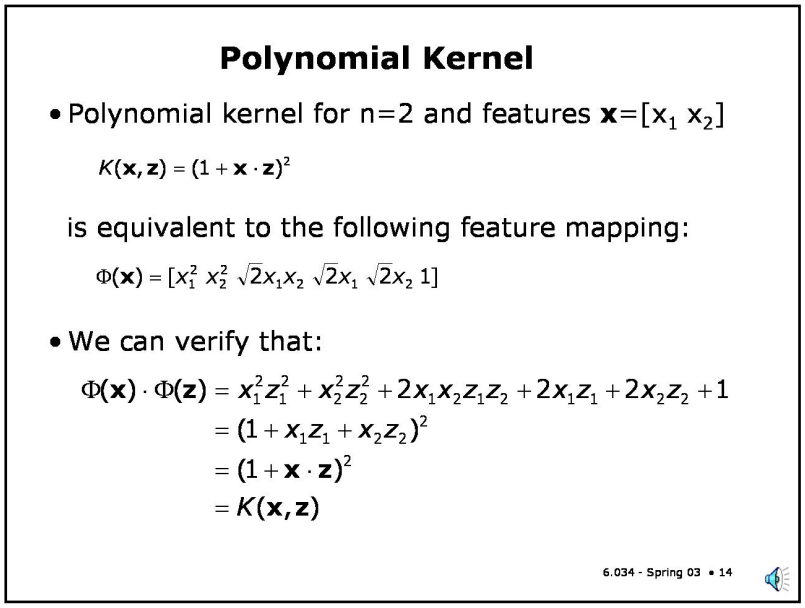
Slide 11.5.13

Note that a second degree polynomial kernel is equivalent to mapping the single feature value x to a three dimensional space with feature values x^2, sqrt(2)x, and 1. You can see that the dot product of two of these feature vectors is exactly the value computed by the polynomial kernel function.

If we plot the original points in the transformed feature space (using just the first two features), we see in fact that the two classes are linearly separable. Clearly, the third feature value (equal to 1) will be irrelevant in finding a separator.

The important aspect of all of this is that we can find and use such a separator without ever explicitly computing the transformed feature vectors - only the kernel function values are required.





Slide 11.5.14

Here is a similar transformation for a two dimensional feature vector. Note that the dimension of the transformed feature vector is now 6. In general, the dimension of the transformed feature vector will grow very rapidly with the dimension of the input vector and the degree of the polynomial.

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

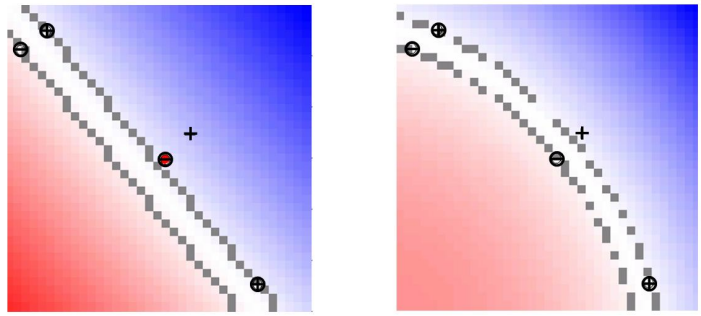
Slide 11.5.15

Let's look at the behavior of these non-linear kernels.

The decision surface produced by the non-linear kernels is curved. Here is an example for which the (unsuccessful) attempt on the left is with a simple dot product; the attempt on the right is done with a polynomial kernel of degree 3. Note the curve in the solution, and note that four of the samples have become support vectors.

Generally, the higher-dimensional the transformed space, the more complex the separator is in the original space, and the more support vectors will be required to specify it.

Polynomial Kernel



Images by Patrick Winston

6.034 - Spring 03 • 15

Standard Choices For Kernels

- No change (linear kernel)

$$\Phi(\mathbf{x}^i)\cdot\Phi(\mathbf{x}^k) = K(\mathbf{x}^i,\mathbf{x}^k) = \mathbf{x}^i\cdot\mathbf{x}^k$$

- Polynomial kernel (n^{th} order)

$$K(\mathbf{x}^i,\mathbf{x}^k) = (1+\mathbf{x}^i\cdot\mathbf{x}^k)^n$$

- Radial basis kernel (σ is standard deviation)

$$K(\mathbf{x}^i,\mathbf{x}^k) = e^{\frac{-\|\mathbf{x}^i-\mathbf{x}^k\|^2}{2\sigma^2}} = e^{\frac{-(\mathbf{x}^i-\mathbf{x}^k)\cdot(\mathbf{x}^i-\mathbf{x}^k)}{2\sigma^2}}$$

6.034 - Spring 03 • 16

Slide 11.5.16

Another popular kernel function is an exponential of the square of the distance between vectors, divided by sigma squared. This is the formula for a Gaussian bump in the feature space, where sigma is the standard deviation of the Gaussian. Sigma is a parameter of the learning that determines the properties of the solution.

Slide 11.5.17

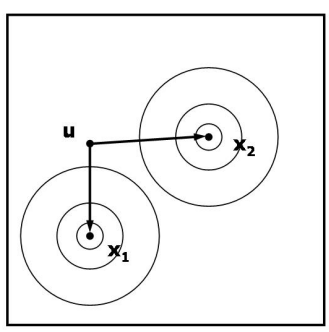
You can get a curved separator if you use radial basis functions, which give us a classifier that is a sum of the values of several Gaussian functions.

Let's pause a minute to observe something that should strike you as a bit weird. When we used the polynomial kernels, we could see that each input feature vector was being mapped into a higher-dimensional, possibly very high dimensional, feature vector. With the radial-basis kernel each input feature vector is being mapped into a **function** that is defined over the whole feature space! In fact, each input feature point is being mapped into a point in an infinite-dimensional feature space (known as a Hilbert space). We then build the classifier as sum of these functions. Whew!

The actual operation of the process is less mysterious than this "infinite-dimensional" mapping view, as we will see by a very simple example.

Radial-basis kernel

- Classifier based on sum of Gaussian bumps with standard deviation σ, centered on support vectors.



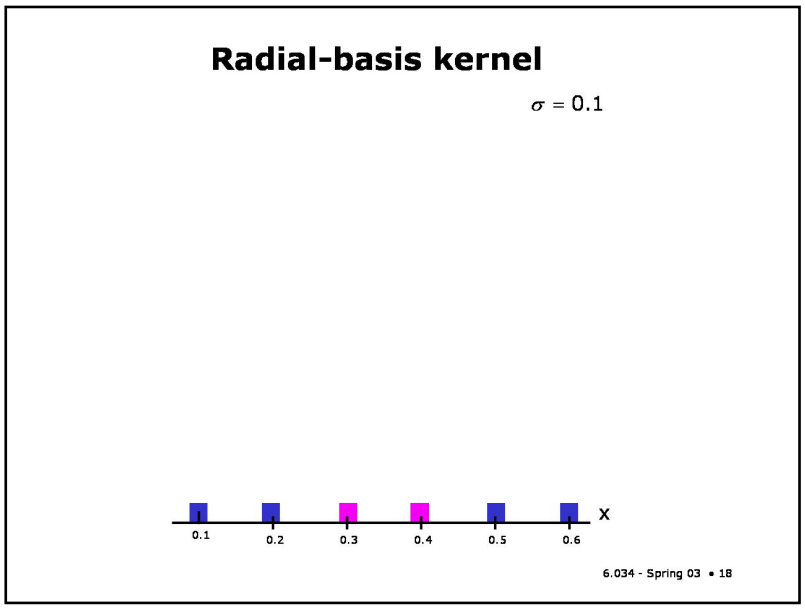
$$h(\mathbf{u}) = sign[h'(\mathbf{u})]$$

$$h'(\mathbf{u}) = \sum_{i=1}^{k}\alpha_i y^i K(\mathbf{x}^i,\mathbf{u}) + b$$

$$K(\mathbf{x}^i,\mathbf{u}) = e^{\frac{-\|\mathbf{x}^i-\mathbf{u}\|^2}{2\sigma^2}}$$

6.034 - Spring 03 • 17

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved



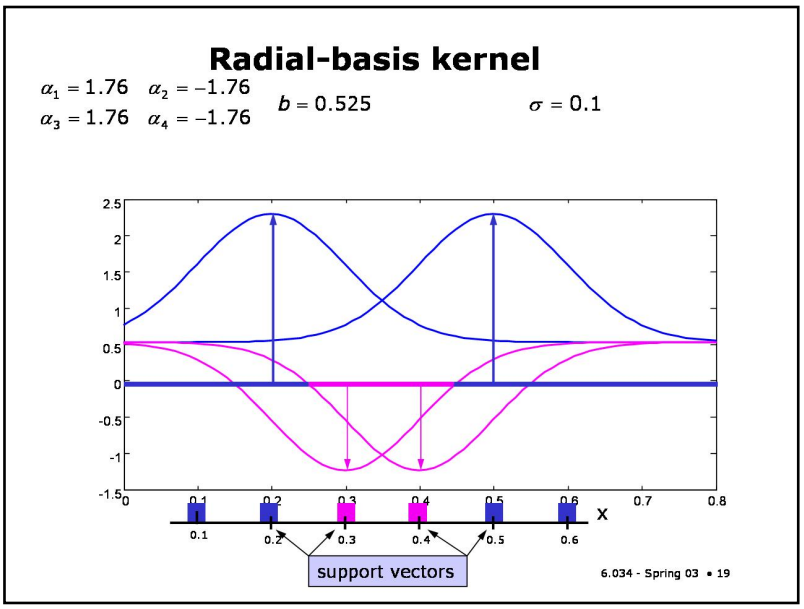
Slide 11.5.18
Along the bottom you see that we're dealing with the simple one-dimensional example that we looked at earlier using a polynomial kernel. The blue points are possitive and the pinkish purple ones are negative. Clearly this arrangement is not linearly separable.

$K(x^i,u)$ can be seen as a "Gaussian bump"; that is, as a function with a maximum at $u = x^i$, that decreases monotonically with the distance between u and x^i, but is always positive and goes to 0 at infinite distance. The parameter sigma specifies how high the bump is and how fast it falls off (the area under the curve of each bump is 1, no matter what the value of sigma is). The smaller the sigma, the more sharply peaked the bump.

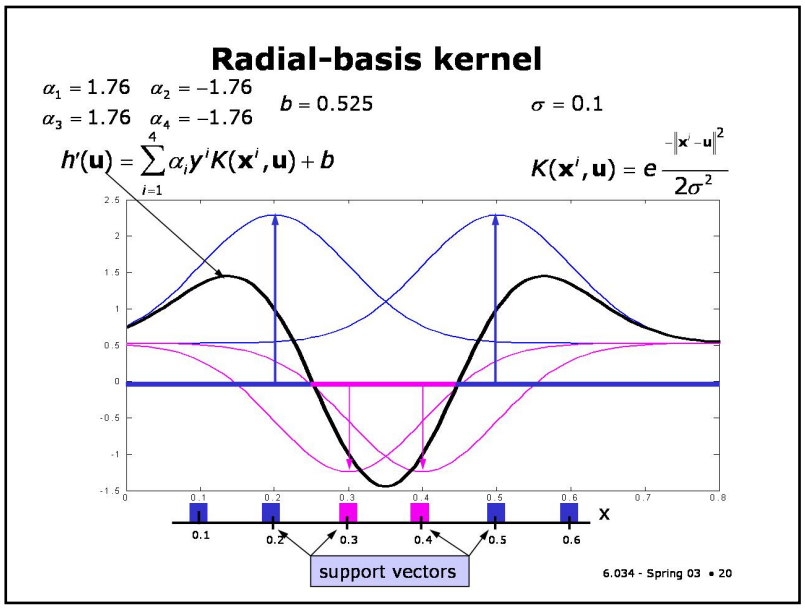
With a radial-basis kernel, we will be looking for a set of multipliers for Gaussian bumps with the specified sigma (here it is 0.1) so that the sum of these bumps (plus an offset) will give us a classification function that's positive where the positive points are and negative where the negative points are.

Slide 11.5.19
Here is the solution obtained from an SVM quadratic optimization algorithm. Note that four points are support vectors, as expected, the points near where the decision boundary has to be. The farther positive points receive alpha=0. The value of the offset, *b is also shown.*

The blue and pink Gaussian bumps correspond to copies of a Gaussian with standard deviation of 0.1 scaled by the corresponding alpha values.



6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

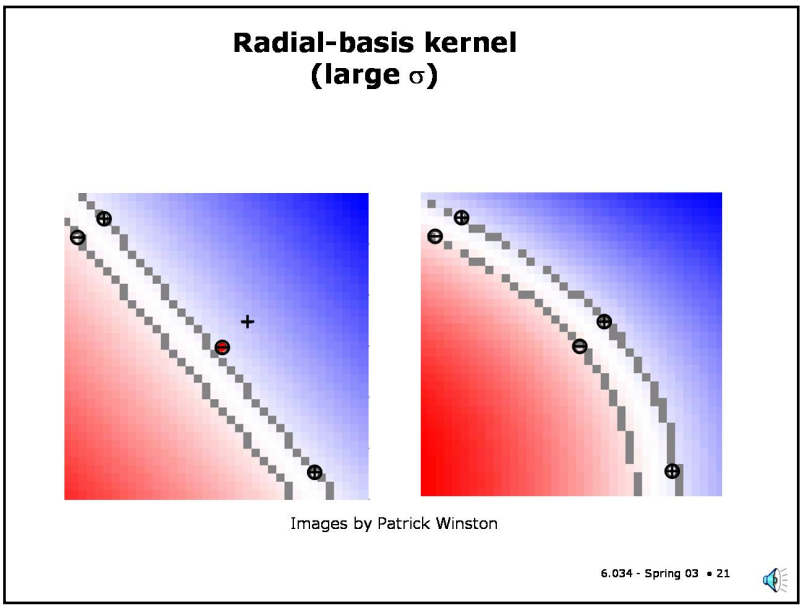


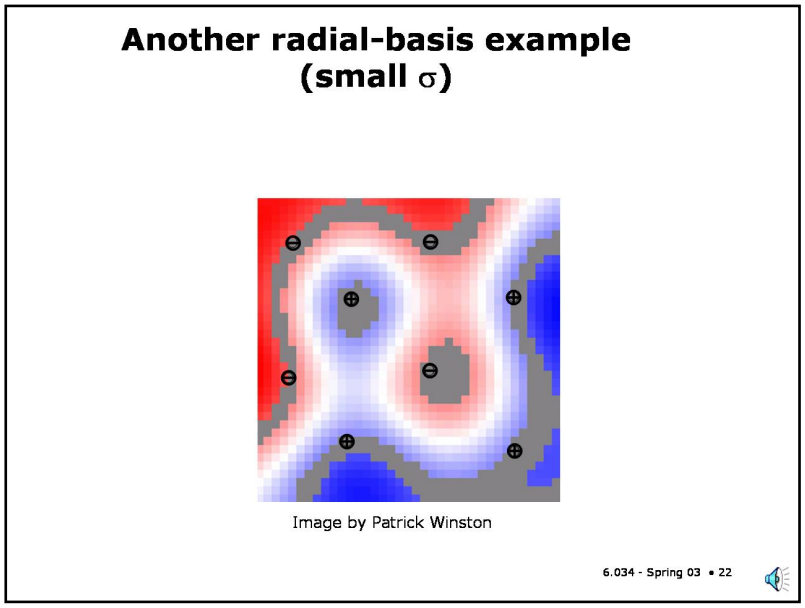
Slide 11.5.20

The black line corresponds to the sum of the four bumps (and the offset). The important point is to notice where this line crosses zero since that's the decision surface (in one dimension). Notice that, as required, it succeeds in separating the positive from the negative points.

Slide 11.5.21

Here we see a separator for our simple five point example computed using radial basis kernels. The solution on the left, for reference, is the original dot product. The solution on the right is for a radial basis function with a sigma of one. Note that all the points are now support vectors.





Slide 11.5.22

If a space is truly convoluted, you can always cover it with a radial basis solution with small-enough sigma. In extreme cases, like this one, each of the four plus and four minus samples has become a support vector, each specialized to the small part of the total space in its vicinity. This is basically similar to 1-nearest neighbor and is just as powerful and subject to overfitting.

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Slide 11.5.23

At this point alarm bells may be ringing. By creating these very high dimensional feature vectors, are we just setting ourselves up for severe overfitting? Intuitively, the more parameters we have the better we can fit the input, but that may not lead to better performance on new data.

It turns out that the fact that the SVM decision surface depends only on the support vectors and not directly on the dimensionality of the space comes to our rescue.

Cross-Validation Error

- Does mapping to a very high-dimensional space lead to over-fitting?
- Generally, no, thanks to the fact that only the support vectors determine the decision surface.

6.034 - Spring 03 • 23

Cross-Validation Error

- Does mapping to a very high-dimensional space lead to over-fitting?
- Generally, no, thanks to the fact that only the support vectors determine the decision surface.
- The expected leave-one-out cross-validation error depends on number of support vectors, not dimensionality of feature space.

$$\text{Expected CV error} \leq \frac{\text{Expected \# support vectors}}{\text{\# training samples}}$$

- If most data points are support vectors, a sign of possible overfitting, independent of the dimensionality of feature space.

6.034 - Spring 03 • 24

Slide 11.5.24

We can estimate the error on new data by computing the cross-validation error on the training data. If we look at the linearly separable case, it is easy to see that the expected value of leave-one-out cross-validation error is bounded by the proportion of support vectors.

If we take a data point that is not a support vector from the training set, the computation of the separator will not be affected and so it will be classified correctly. If we take a support vector out, then the classifier will in general change and there may be an error. So, the expected generalization error depends on the number of support vectors and not on the dimension.

Note that using a radial basis kernel with very small sigma gives you a high expected number of support vectors and therefore a high expected cross-validation error, as expected. Yet, a radial basis kernel with large sigma, although of similar dimensionality, has fewer expected support vectors and is likely to generalize better.

We shouldn't take this bound too seriously; it is not actually very predictive of generalization performance in practice but it does point out an important property of SVMs - that generalization performance is more related to expected number of support vectors than to dimensionality of the transformed feature space.

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Slide 11.5.25

So, let's summarize the SVM story. One key point is that SVMs have a training method that guarantees a unique global optimum. This eliminate many headaches in other approaches to machine learning.

Summary

- A single global optimum
  - Quadratic programming or gradient descent

6.034 - Spring 03 • 25

Summary

- A single global maximum
  - Quadratic programming or gradient descent
- Fewer parameters
  - C and kernel parameters (n for polynomial, σ for radial basis kernel)

6.034 - Spring 03 • 26

Slide 11.5.26

The other advantage of SVMs is that there are relatively few parameters to be chosen: C, the constant used to trade off classification error and width of the margin; and the kernel parameter, such as sigma in the radial basis kernel.

These can both be continuous parameters and so there still remains a search requiring some form of validation, but these are few parameters compared to some of the other methods.

Slide 11.5.27

And, last but not least, is the kernel trick. That is, that the whole process depends only on the dot products of the feature vectors, which is the key to the generalization to non-linear classifiers.

Summary

- A single global maximum
  - Quadratic programming or gradient descent
- Fewer parameters
  - C and kernel parameters (n for polynomial, σ for radial basis kernel)
- Kernel
  - Quadratic minimization depends only on dot products of sample vectors
  - Recognition depends only on dot products of unknown vector with sample vectors
  - Reliance on only dot products enables efficient feature mapping to higher-dimensional spaces where linear separation is more effective.

6.034 - Spring 03 • 27

6.034 Artificial Intelligence. Copyright © 2004 by Massachusetts Institute of Technology. All rights reserved

Real Data

- Wisconsin Breast Cancer Data
  - 9 features
  - C=1
  - 37 support vectors are used from 512 training data points
  - 12 prediction errors on training set (98% accuracy)
  - 96% accuracy on 171 held out points
  - Essentially same performance as nearest neighbors and decision trees
- Don't expect such good performance on every data set.

6.034 - Spring 03 • 28

Slide 11.5.28
The linear separator is very simple hypothesis class but it can perform very well on appropriate data sets. On the Wisconsin breast cancer data, the maximal margin classifier, with a linear kernel, does as well or better as any of the other classifiers we have seen on held-out data. Note that only 37 of the 512 training points are support vectors.

Slide 11.5.29
SVMs have proved useful in a wide variety of applications, particularly those with large numbers of features, such as image and text recognition problems. They are the method of choice in text classification problems, such as categorization of news articles by topic, or spam detection, because they can work in a huge feature space (typically with a linear kernel) without too much fear of overfitting.

Success Stories

- Gene microarray data
  - outperformed all other classifiers
  - specially designed kernel
- Text categorization
  - linear kernel in >10,000 D input space
  - best prediction performance
  - 35 times faster to train than next best classifier (decision trees)
- Many others:
  http://www.clopinet.com/isabelle/Projects/SVM/applist.html

6.034 - Spring 03 • 29