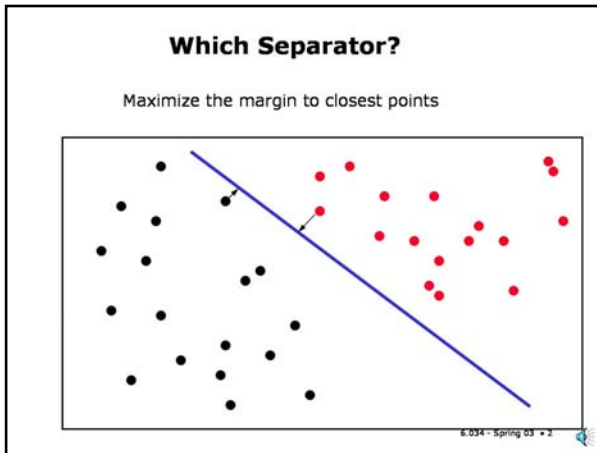
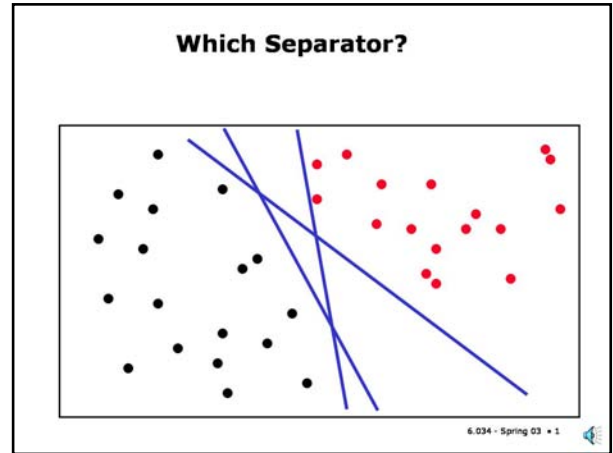


6.034 Notes: Section 8.1

Slide 8.1.1

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 8.1.2

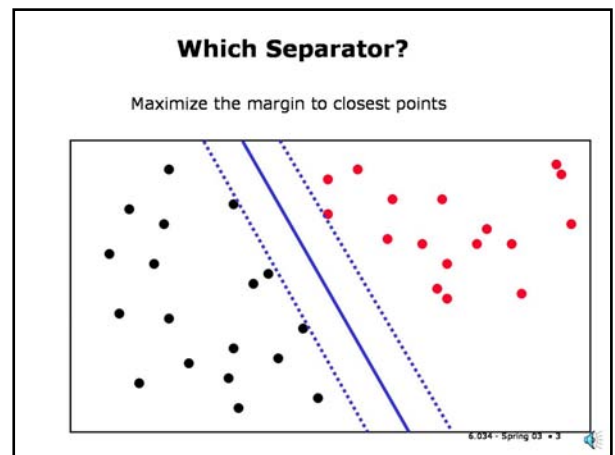
Yes. One natural choice is to pick the separator that has the maximal margin to its closest points on either side. This is the separator that seems most conservative. Any other separator will be "closer" to one class than to the other. The one shown in this figure, for example, seems like it's closer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

Slide 8.1.3

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.



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 8.1.4

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 \mathbf{w} , b notation instead of $\bar{\mathbf{w}}$, because we will end up giving b special treatment in the future.

Slide 8.1.5

Remember that any scaling of \mathbf{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 \mathbf{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 \mathbf{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 8.1.6

The next issue is that 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 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 \mathbf{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 \mathbf{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.

Slide 8.1.7

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 \mathbf{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 \mathbf{w} gives perpendicular distance between dashed lines ($2 \times$ geometric margin)

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

6.034 - Spring 03 • 7

Picking \mathbf{w} to Maximize Margin

- Pick \mathbf{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 8.1.8

So, we want to pick \mathbf{w} to maximize the geometric margin, that is, to maximize 2 over the magnitude of \mathbf{w} . To maximize this expression, we want to minimize the magnitude of \mathbf{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 $\mathbf{w} = 0$ which would be completely useless.

Slide 8.1.9

We'd like to find the \mathbf{w} that specifies the maximum margin separator. To be a separator, \mathbf{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 will be greater than or equal to 1. The fact that we are minimizing the magnitude of \mathbf{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 \mathbf{w} to Maximize Margin

- Pick \mathbf{w} to maximize geometric margin

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

- or, equivalently, minimize

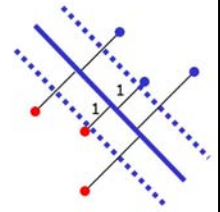
$$\frac{1}{2} \|\mathbf{w}\|^2 = \frac{1}{2} \mathbf{w} \cdot \mathbf{w} = \frac{1}{2} \sum_j w_j^2$$

- 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

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 8.1.10

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 8.1.8

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 α_i . Think of these terms as penalty terms that will penalize values of \mathbf{w} that do not satisfy the constraints.

Picking \mathbf{w} to Maximize Margin

- Pick \mathbf{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

Constrained Optimization

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

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

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

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 8.1.12**

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 \mathbf{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.

Slide 8.1.13

This method we have begun to outline here is called the **method of Lagrange multipliers** and the α_i are the individual Lagrange multipliers.

Constrained Optimization

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

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

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

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...

Lagrange multipliers

Method of Lagrange multipliers

6.034 - Spring 03 • 13

**6.034 Notes: Section 8.2****Slide 8.2.1**

The details of solving a Lagrange multiplier problem are a little bit complicated. 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.

Maximizing the Margin

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

6.034 - Spring 03 • 1



Maximizing the Margin

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

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

6.034 - Spring 03 • 2

Slide 8.2.2

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 \mathbf{w} and b would optimize L for those fixed alphas. We can do this by taking the partial derivatives of L with respect to \mathbf{w} and b and setting them to zero, getting two constraints. We find that the best value of \mathbf{w} , \mathbf{w}^* is a weighted sum of the input points (in the same form as the dual perceptron); 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 8.2.3

We can substitute this expression for the optimal \mathbf{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 \mathbf{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

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

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

Substituting \mathbf{w}^* into L yields dual Lagrangian:

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

Only dot products of the feature vectors appear

6.034 - Spring 03 • 3

Dual Lagrangian

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

6.034 - Spring 03 • 4

Slide 8.2.4

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 later 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 \geq inequalities. The constraint on the alphas comes from the setting to zero the derivative of the Lagrangian with respect to the offset b .

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.

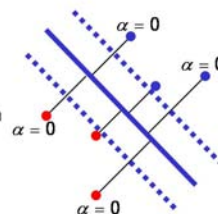
Slide 8.2.5

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.

Dual Lagrangian

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

In general, since $\alpha_i \geq 0$, either
 $\alpha_i = 0$: constraint is satisfied with no distortion at optimum \mathbf{w}
 or
 $\alpha_i > 0$: constraint is satisfied with equality (in this case \mathbf{x}' is known as a support vector)



6.034 - Spring 03 • 5

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 $\alpha_i \geq 0$, either
 $\alpha_i = 0$: constraint is satisfied with no distortion at optimum w
 or
 $\alpha_i > 0$: constraint is satisfied with equality (x^i is known as a support vector)

$w^* = \sum_i \alpha_i y^i x^i$

$b = 1/y^i - w^* \cdot x^i$

6.034 - Spring 03 • 6

Slide 8.2.6

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.

Slide 8.2.7

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 $\alpha_i \geq 0$, either
 $\alpha_i = 0$: constraint is satisfied with no distortion at optimum w
 or
 $\alpha_i > 0$: constraint is satisfied with equality (x^i is known as a support vector)

$w^* = \sum_i \alpha_i y^i x^i$

$b = 1/y^i - w^* \cdot x^i$

- 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:

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

- The sum is over k support vectors

6.034 - Spring 03 • 8

Slide 8.2.8

With the values of the optimal alpha'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 8.2.9

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

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 8.2.10

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 8.2.8

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.

SVM Classifier

- Given unknown vector \mathbf{u} , predict class (1 or -1) as follows:

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

- The sum is over k support vectors

6.034 - Spring 03 • 8

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 8.2.12

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.

Slide 8.2.13

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.

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.
- Max margin lowers hypothesis variance.

6.034 - Spring 03 • 13

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.
- Max margin lowers hypothesis variance.
- The optimal classifier is defined uniquely – there are no "local maxima" in the search space
- Polynomial in number of data points and dimensionality

6.034 - Spring 03 - 14



Slide 8.2.14

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 8.3

Slide 8.3.1

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.

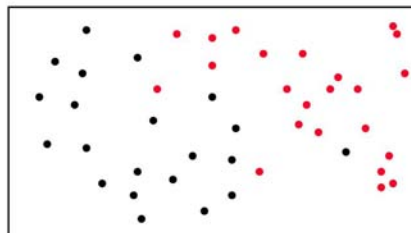
It is useful to think about the behavior of the dual perceptron on this type of problem. In that algorithm, the value of the α_i for a point is incremented proportionally to its distance to the separator. In fact, if the point is classified correctly, no change is made to the multiplier. We can see that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

The α 's in the dual perceptron are analogous to the values of the Lagrange multipliers in the SVM. In both cases, the separator is defined as a linear combination of the input points, with the α s being the weights.

So, one strategy for dealing with these noise points in an SVM is to limit the maximal value of any of the α_i 's (the Lagrange multipliers) 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 so as to achieve a better margin.

Not Linearly Separable?

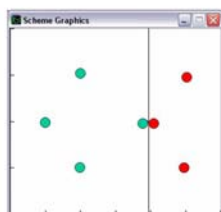
- Require $0 \leq \alpha_i \leq C$
- C specified by user; controls tradeoff between size of margin and classification errors
- $C = \infty$ for separable case



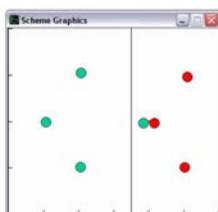
6.034 - Spring 03 - 1



C Change



C=10



C=1

6.034 - Spring 03 - 2



Slide 8.3.2

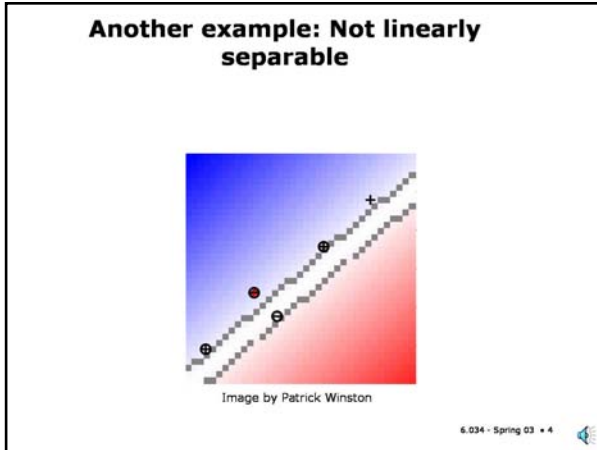
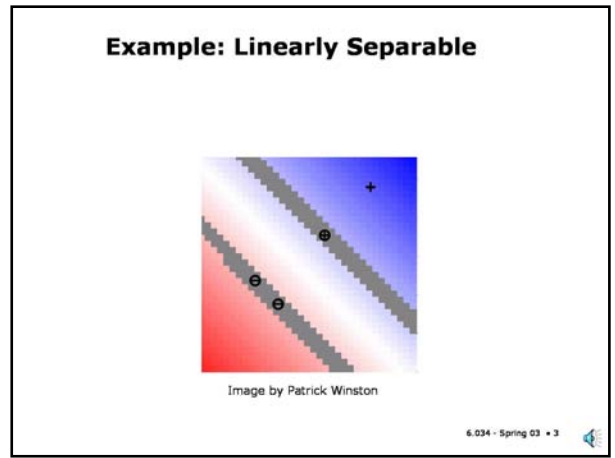
This simple example shows how changing C causes the geometric margin to change. High values of C penalize misclassifications more. Low values may permit misclassifications to achieve better margin.

Slide 8.3.3

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 8.3.4

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.

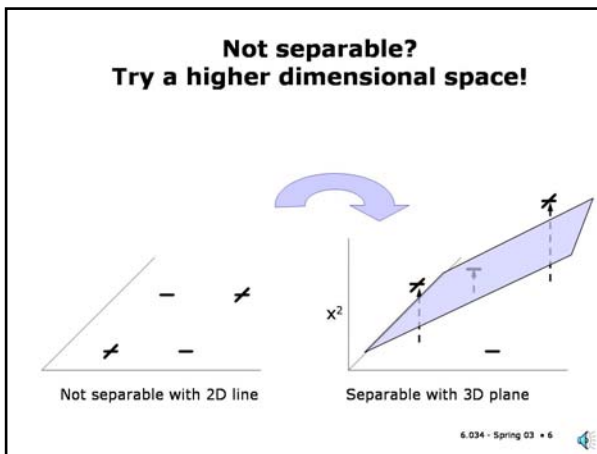
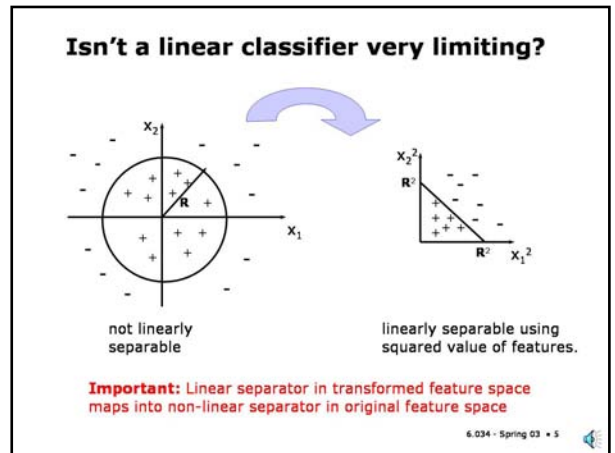
Second, the added point produced quite a different solution. The algorithm is looking for best possible dividing line; a tradeoff between margin and classification error defined by C . If we had kept a solution close to the one in the previous slide, the rogue plus point would have been misclassified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

Slide 8.3.5

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.



Slide 8.3.6

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 8.3.7

First, suppose there is a function, ϕ , 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.

What you need

- To get into the new feature space, you use $\Phi(\mathbf{x}^i)$
- The transformation can be to a higher-dimensional feature space and may be non-linear in the feature values.

6.034 - Spring 03 • 7

**What you need**

- To get into the new feature space, you use $\Phi(\mathbf{x}^i)$
- The transformation can be to a higher-dimensional feature space and may be non-linear in the feature values.
- Recall that SVM's only use dot products of the data, so
- To optimize classifier, you need $\Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{x}^k)$
- To run classifier, you need $\Phi(\mathbf{x}^i) \cdot \Phi(\mathbf{u})$
- So, all you need is a way to compute dot products in transformed space as a function of vectors in original space!

6.034 - Spring 03 • 8

**Slide 8.3.8**

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.

Slide 8.3.9

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 8.3.10**

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

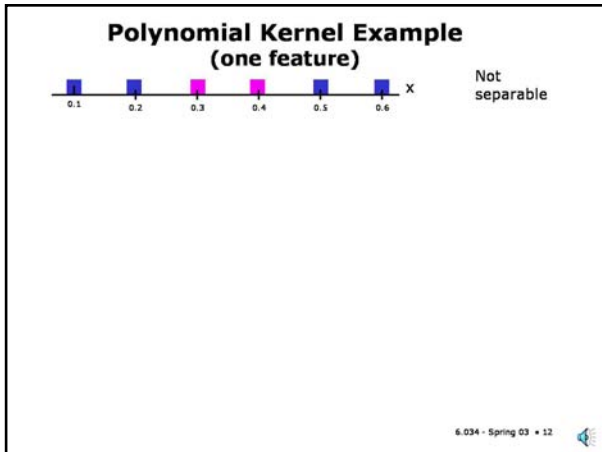
Slide 8.3.8

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.

What you need

- To get into the new feature space, you use $\Phi(\mathbf{x}')$
- The transformation can be to a higher-dimensional feature space and may be non-linear in the feature values.
- Recall that SVM's only use dot products of the data, so
- To optimize classifier, you need $\Phi(\mathbf{x}') \cdot \Phi(\mathbf{x}'')$
- To run classifier, you need $\Phi(\mathbf{x}') \cdot \Phi(\mathbf{u})$
- So, all you need is a way to compute dot products in transformed space as a function of vectors in original space!

6.034 - Spring 03 • 8



Slide 8.3.12

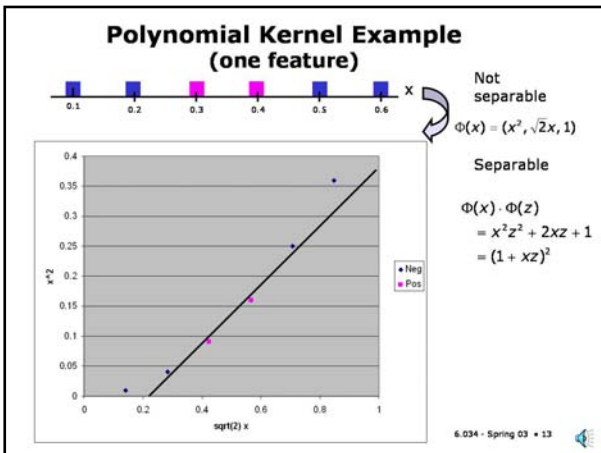
Let's look at a simple example of using a polynomial kernel. Consider the one dimensional problem shown here, which is clearly not separable. Let's map it into a higher dimensional feature space using the polynomial kernel of second degree ($n=2$).

Slide 8.3.13

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.



Polynomial Kernel

- Polynomial kernel for $n=2$ and features $\mathbf{x}=[x_1 \ x_2]$

$$K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x} \cdot \mathbf{z})^2$$

is equivalent to the following feature mapping:

$$\Phi(\mathbf{x}) = [x_1^2 \ x_2^2 \ \sqrt{2}x_1x_2 \ \sqrt{2}x_1 \ \sqrt{2}x_2 \ 1]$$
- We can verify that:

$$\begin{aligned} \Phi(\mathbf{x}) \cdot \Phi(\mathbf{z}) &= x_1^2 z_1^2 + x_2^2 z_2^2 + 2x_1 x_2 z_1 z_2 + 2x_1 z_1 + 2x_2 z_2 + 1 \\ &= (1 + x_1 z_1 + x_2 z_2)^2 \\ &= (1 + \mathbf{x} \cdot \mathbf{z})^2 \\ &= K(\mathbf{x}, \mathbf{z}) \end{aligned}$$

6.034 - Spring 03 • 14

Slide 8.3.14

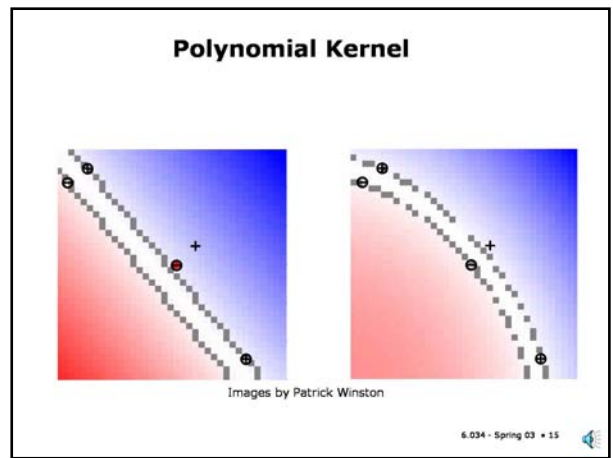
Here is a similar transformation for a two dimensional feature vector. Note that the dimension of the transformed feature vector is now 6. In general, the dimension of the transformed feature vector will grow very rapidly with the dimension of the input vector and the degree of the polynomial.

Slide 8.3.15

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.



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 8.3.16

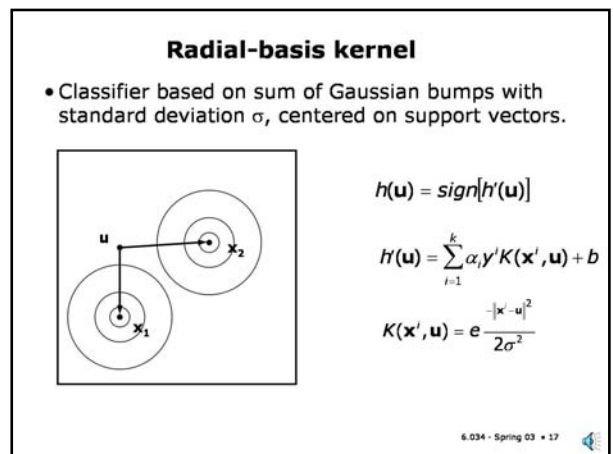
Another popular kernel function is an exponential of the square of the distance between vectors, divided by sigma squared. This is the formula for a Gaussian bump in the feature space, where sigma is the standard deviation of the Gaussian. Sigma is a parameter of the learning that determines the properties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

Slide 8.3.17

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

$$\sigma = 0.1$$



6.034 - Spring 03 • 18

Slide 8.3.18

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 positive and the pinkish purple ones are negative. Clearly this arrangement is not linearly separable.

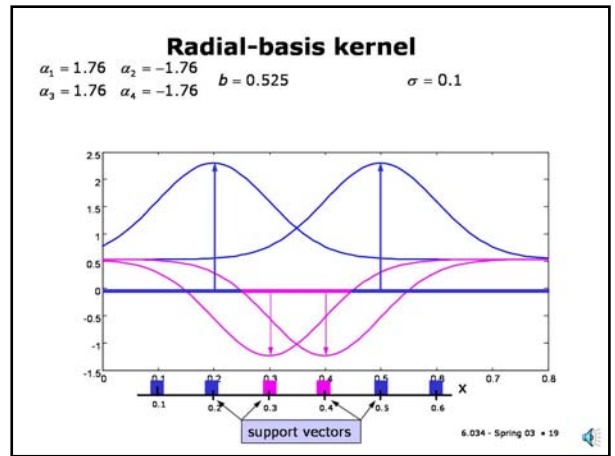
$K(x^i, u)$ can be seen as a "Gaussian bump"; that is, as a function with a maximum at $u = x^i$, that decreases monotonically with the distance between u and x^i , but is always positive and goes to 0 at infinite distance. The parameter sigma specifies how high the bump is and how fast it falls off (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 8.3.19

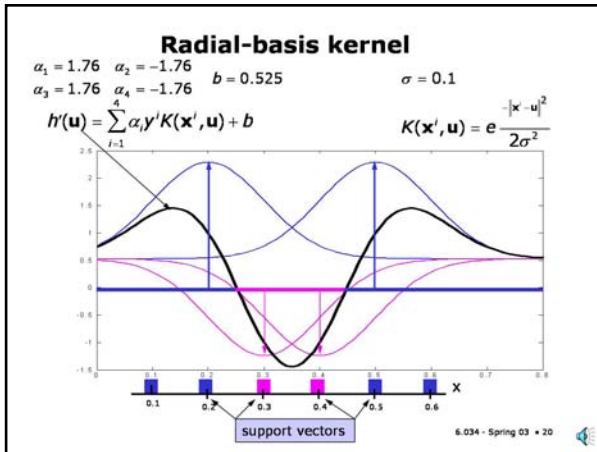
Here is the solution obtained from an SVM quadratic optimization algorithm. Note that four points are support vectors, as expected, the points near where the decision boundary has to be. The farther positive points receive $\alpha=0$. The value of the offset, b is also shown.

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



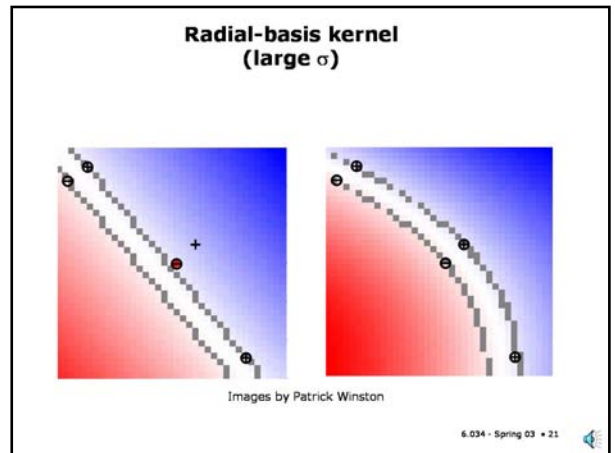
Slide 8.3.20

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.



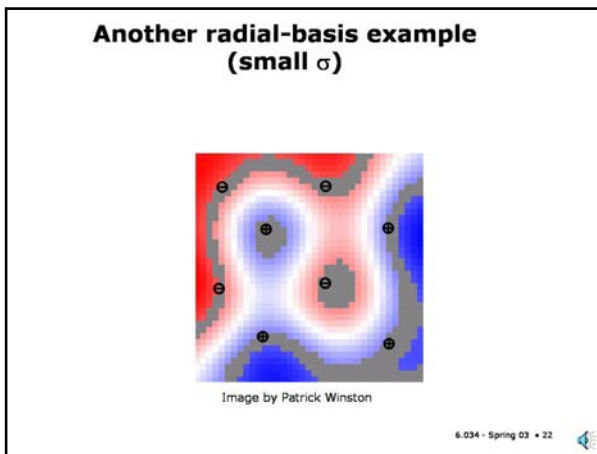
Slide 8.3.21

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 8.3.22

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.



Slide 8.3.23

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 8.3.24

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.

Slide 8.3.25

So, let's summarize the SVM story. One key point is that SVMs have a training method that guarantees a unique global optimum. This eliminates 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 8.3.26

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 8.3.27

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

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 8.3.28

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 8.3.29

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

6.034 Notes: Section 8.4

Slide 8.4.1

In many machine-learning applications, there are huge numbers of features. In text classification, you often have as many features as there are words in the dictionary. Gene expression arrays have five to fifty thousand elements. Images can have as many as 512 by 512 pixels.

Feature Selection

- In many machine learning applications, there are huge numbers of features
 - text classification (# words)
 - gene arrays (5,000 – 50,000)
 - images (512 x 512 pixels)

6.034 - Spring 03 • 1

Feature Selection

- In many machine learning applications, there are huge numbers of features
 - text classification (# words)
 - gene arrays (5,000 – 50,000)
 - images (512 x 512 pixels)
- Too many features
 - make algorithms run slowly
 - risk overfitting

6.034 - Spring 03 • 2

Slide 8.4.2

When there are lots of features in a domain, it can make some machine learning algorithms run much too slowly. Worse, it often causes overfitting problems: most classifiers have a complexity related to the number of features, and in many of these cases we can have many more features than training examples, which doesn't give us much confidence in our parameter estimates.

Feature Selection

- In many machine learning applications, there are huge numbers of features
 - text classification (# words)
 - gene arrays (5,000 – 50,000)
 - images (512 x 512 pixels)
- Too many features
 - make algorithms run slowly
 - risk overfitting
- Find a smaller feature space
 - subset of existing features
 - new features constructed from old ones

6.034 - Spring 03 • 3

Slide 8.4.3

There are two approaches to dealing with very large feature spaces: one is to select a subset of the given set of features to work with; the other is to make new features that are supposed to describe the input space more efficiently than the given set of features.

Feature Ranking

- For each feature, compute a measure of its relevance to the output
- Choose the k features with the highest rankings
- Correlation between feature j and output

$$R(j) = \frac{\sum_i (x_j^i - \bar{x}_j)(y^i - \bar{y})}{\sqrt{\sum_i (x_j^i - \bar{x}_j)^2 \sum_i (y^i - \bar{y})^2}}$$

$$\bar{x}_j = \frac{1}{n} \sum_i x_j^i \quad \bar{y} = \frac{1}{n} \sum_i y^i$$

- Correlation measures how much x tends to deviate from its mean on the same examples on which y deviates from its mean

6.034 - Spring 03 • 4

Slide 8.4.4

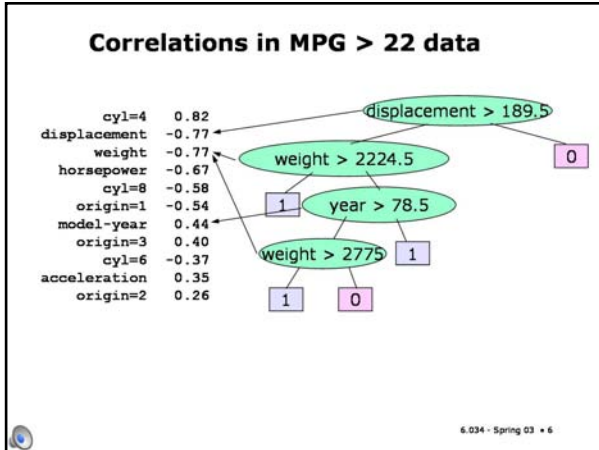
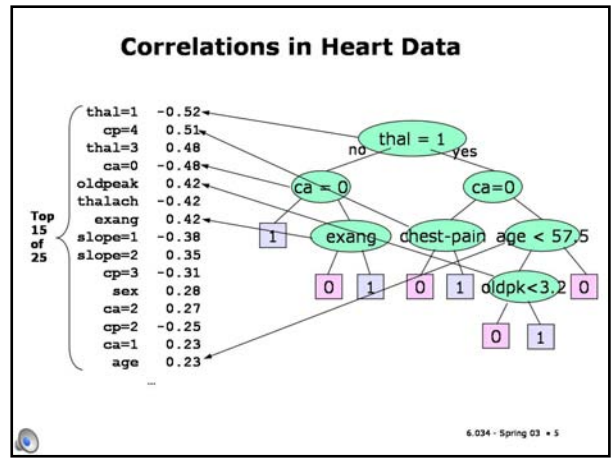
The simplest feature-selection strategy is to compute some score for each feature, and then select the k features with the highest rankings.

A popular feature score is the correlation between a feature and the output variable. It measures the degree to which a feature varies with the output, and is usable when the output is discrete or continuous.

Slide 8.4.5

We computed the correlations of each of the features in the heart disease data set with the output. They are shown here in sorted order, with reference to the decision tree we learned on this data.

We can see that most of the features used in the tree show up among the top features, ranked according to correlation. You can see the features with a positive correlation score indicate that heart disease is more likely, and those with a negative score indicate that it is less likely.



Slide 8.4.6

Here's a similar figure for the auto fuel efficiency data. It's interesting to see that the highest-correlation feature is binary choice about whether there are 4 cylinders. It looks like binary features have a tendency to be preferred (since the output is binary, as well, and so they often match up perfectly). But displacement is also very highly ranked, and probably contains more information than the number of cylinders.

Slide 8.4.7

As usual, XOR will cause us trouble if we do scoring of single features. In an XOR problem, each feature will, individually, have a correlation of 0 with the output.

To solve xor problems, we need to look at groups of features together.

XOR Bites Back

- As usual, functions with XOR in them will cause us trouble
- Each feature will, individually, have a correlation of 0 (it occurs positively as much as negatively for positive outputs)
- To solve XOR, we need to look at groups of features together

Subset Selection

- Consider subsets of variables
 - too hard to consider all possible subsets
 - wrapper methods: use training set or cross-validation error to measure the goodness of using different feature subsets with your classifier
 - greedily construct a good subset by adding or subtracting features one by one

Slide 8.4.8

Ideally, we'd like to try all possible subsets of the features and see which one works best. We can evaluate a subset of features by training a classifier using just that subset, and then measuring the performance using training set or cross-validation error.

Instead of trying all subsets, we'll consider greedy methods that add or subtract features one at a time.

Slide 8.4.9

In the forward selection method, we start with no features at all in our feature set. Then, for each feature, we consider adding it to the feature set: we add it, train a classifier, and see how well it performs (on a separate validation set or by using cross-validation). We then add the feature that generated the best classifier to our existing set and continue.

We'll terminate the algorithm when we have as many features as we can handle, or when the error has quit decreasing.

Forward Selection

Given a particular classifier you want to use

```

F = {}
For each  $f_j$ 
  Train classifier with inputs  $F + \{f_j\}$ 
  Add  $f_j$  that results in lowest-error classifier to F
Continue until F is the right size, or error has quit decreasing

```

6.034 - Spring 03 • 9

Slide 8.4.10

Decision trees work sort of like this: they add features one at a time, choosing the next feature in the context of the ones already chosen. However, they establish a whole tree of feature-selection contexts.

Forward Selection

Given a particular classifier you want to use

```

F = {}
For each  $f_j$ 
  Train classifier with inputs  $F + \{f_j\}$ 
  Add  $f_j$  that results in lowest-error classifier to F
Continue until F is the right size, or error has quit decreasing

```

- Decision trees, by themselves, do something similar to this

6.034 - Spring 03 • 10

Slide 8.4.11

Even if we do forward selection, XOR can cause us trouble. Because we only consider adding features one by one, neither of the features will look particularly attractive individually, and so we would be unlikely to add them until the very end.

Forward Selection

Given a particular classifier you want to use

```

F = {}
For each  $f_j$ 
  Train classifier with inputs  $F + \{f_j\}$ 
  Add  $f_j$  that results in lowest-error classifier to F
Continue until F is the right size, or error has quit decreasing

```

- Decision trees, by themselves, do something similar to this
- Trouble with XOR

6.034 - Spring 03 • 11

Slide 8.4.12

Backward elimination works in the other direction. It starts with all the features in the feature set and eliminates them one by one, removing the one that results in the best classifier at each step.

This strategy can cope effectively with XOR-like problems. But it might be impractical if the initial feature set is so large that it makes the algorithm to slow to run.

Backward Elimination

Given a particular classifier you want to use

```

F = all features
For each  $f_j$ 
  Train classifier with inputs  $F - \{f_j\}$ 
  Remove  $f_j$  that results in lowest-error classifier from F
Continue until F is the right size, or error increases too much

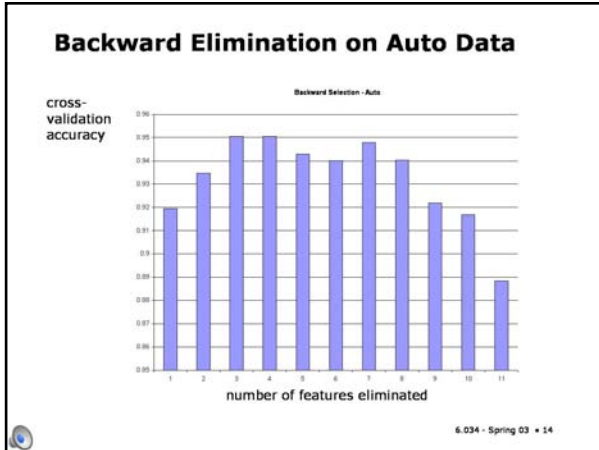
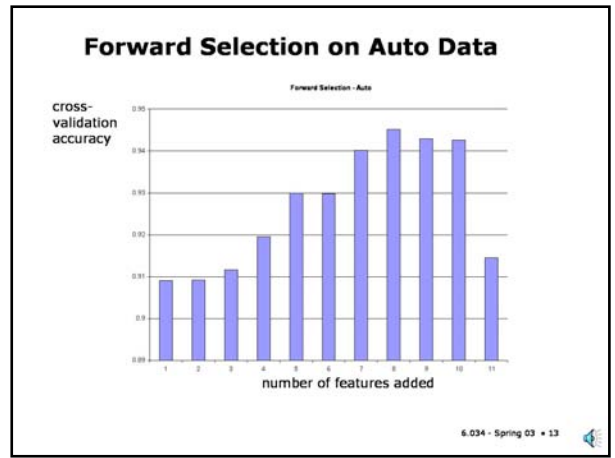
```

6.034 - Spring 03 • 12

Slide 8.4.13

Here's a plot of the cross-validation accuracy against number of features chosen by forward selection on the auto data. The classifier we used was nearest neighbor.

Here we can see that adding features improves cross-validation accuracy until the last couple of features. If there were a large number of relatively noisy features, adding them would make the performance go down even further, as they would give the classifier further opportunity for overfitting.

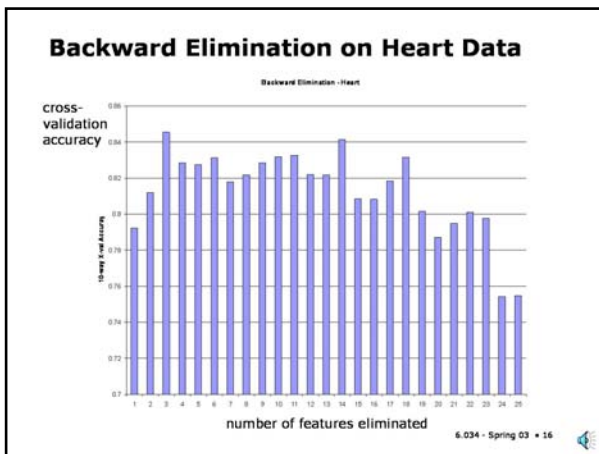
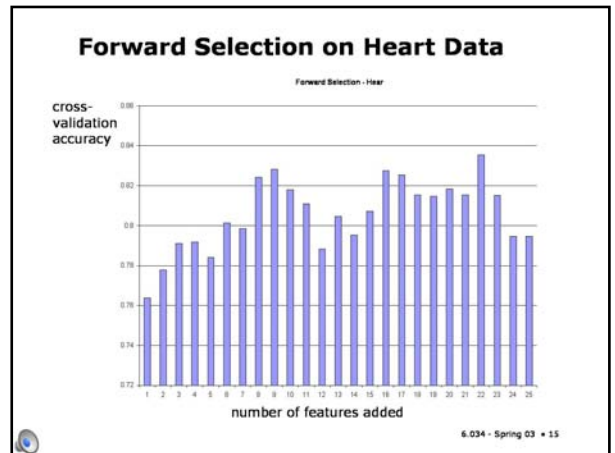


Slide 8.4.14

The picture for backward elimination is similar. But notice that it seems to work a bit better, even when we are eliminating a lot of features. This may be because it can decide which features to eliminate in the context of all the other features. Forward selection, especially in the early phases, picks features without much context.

Slide 8.4.15

On the heart data, we need about 8 features before we're getting reasonably good performance. The accuracies are pretty erratic after that; it's probably an indication of overall variance in the performance estimates.



Slide 8.4.16

We can see similar performance with backward elimination. It's possible to get rid of a lot of features before performance suffers dramatically. And, it really seems to be worthwhile to eliminate some of the features, from a performance perspective.

Slide 8.4.17

Backward elimination and forward selection can be computationally quite expensive, because they require you, on each iteration, to train approximately as many classifiers as you have features.

In some classifiers, such as linear support-vector machines and linear neural networks, it's possible to do backward elimination more efficiently. You train the classifier once, and then remove the feature that has the smallest input weight.

These methods can be extended to non-linear SVMs and neural networks, but it gets somewhat more complicated there.

Recursive Feature Elimination

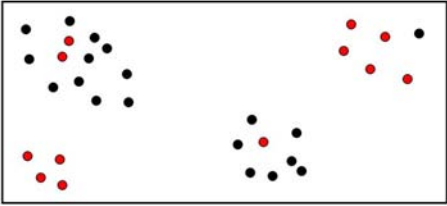
Train a linear SVM or neural network
Remove the feature with the smallest weight
Repeat

- More efficient than regular backward elimination
- Requires only one training phase per feature

6.034 - Spring 03 • 17

Clustering

- Form clusters of inputs
- Map the clusters into outputs
- Given a new example, find its cluster, and generate the associated output



6.034 - Spring 03 • 18

Slide 8.4.18

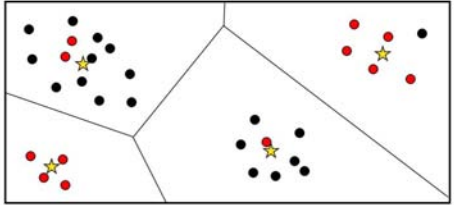
Another whole strategy for feature selection is to make new features. One very drastic method is to try to cluster all of the inputs in your data set into a relatively small number of groups, and then learn a mapping from each group into an output.

Slide 8.4.19

So, in this case, we might divide the input points into 4 clusters. The stars indicate the cluster centers.

Clustering

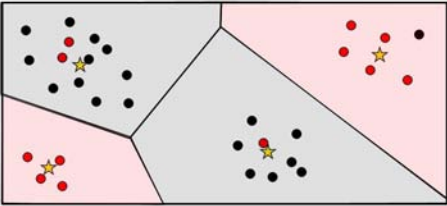
- Form clusters of inputs
- Map the clusters into outputs
- Given a new example, find its cluster, and generate the associated output



6.034 - Spring 03 • 19

Clustering

- Form clusters of inputs
- Map the clusters into outputs
- Given a new example, find its cluster, and generate the associated output



6.034 - Spring 03 • 20

Slide 8.4.20

Then, for each cluster, we would assign the majority class. Now, to predict the value of a new point, we would see which region it would land in, and predict the associated class.

This is different from nearest neighbors in that we actually discard all the data except the cluster centers. This has the advantage of increasing interpretability, since the cluster centers represent "typical" inputs.

Slide 8.4.21

So, what makes a good clustering? There are lots and lots of different technical choices. The basic idea is usually that you want to have clusters in which the distance between points in the same group is small and the distance between points in different groups is large.

Clustering, like nearest neighbor, requires a distance metric, and the results you get are as scale-sensitive as they are in nearest-neighbor.

Clustering Criteria

- small distances between points within a cluster
- large distances between clusters
- Need a distance measure, as in nearest neighbor

6.034 - Spring 03 • 21

K-Means Clustering

- Tries to minimize

$$\sum_{j=1}^k \sum_{i \in S_j} \|x^i - \mu_j\|^2$$

Diagram illustrating the components of the K-Means objective function:

- k : # of clusters
- $\sum_{i \in S_j}$: elements of cluster j
- $\|x^i - \mu_j\|^2$: squared dist from point to mean
- μ_j : mean of elts in cluster j

- Only gets, greedily, to a local optimum

6.034 - Spring 03 • 22

Slide 8.4.22

One of the simplest and most popular clustering methods is K-means clustering. It tries to minimize the sum, over all the clusters, of the variance of the points within the cluster (the distances of the points to the geometric center of the cluster).

Unfortunately, it only manages to get to a local optimum of this measure, but it's usually fairly reasonable.

Slide 8.4.23

Here is the code for the k-means clustering algorithm. You start by choosing k , your desired number of clusters. Then, you can randomly choose k of your data points to serve as the initial cluster centers.

K-means Algorithm

```
Choose k
Randomly choose k points  $C_j$  to be cluster centers
```

6.034 - Spring 03 • 23

K-means Algorithm

```
Choose k
Randomly choose k points  $C_j$  to be cluster centers
Loop
  Partition the data into k classes  $S_j$  according
  to which of the  $C_j$  they're closest to
  For each  $S_j$ , compute the mean of its elements
  and let that be the new cluster center
```

6.034 - Spring 03 • 24

Slide 8.4.24

Then, we enter a loop with two steps. The first step is to divide the data up into k classes, using the cluster centers to make a Voronoi partition of the data. That is, we assign each data point to the cluster center that it's closest to.

Now, for each new cluster, we compute a new cluster center by averaging the elements that were assigned to that cluster on the previous step.

Slide 8.4.25

We stop when the centers quit moving. This process is guaranteed to terminate.

K-means Algorithm

```
Choose k
Randomly choose k points  $C_j$  to be cluster centers
Loop
  Partition the data into k classes  $S_j$  according
  to which of the  $C_j$  they're closest to
  For each  $S_j$ , compute the mean of its elements
  and let that be the new cluster center
Stop when centers quit moving
```

- Guaranteed to terminate

6.034 - Spring 03 • 25

K-means Algorithm

```
Choose k
Randomly choose k points  $C_j$  to be cluster centers
Loop
  Partition the data into k classes  $S_j$  according
  to which of the  $C_j$  they're closest to
  For each  $S_j$ , compute the mean of its elements
  and let that be the new cluster center
Stop when centers quit moving
```

- Guaranteed to terminate
- If a cluster becomes empty, re-initialize the center

6.034 - Spring 03 • 26

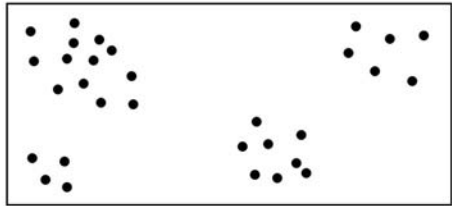
Slide 8.4.26

One possible problem is that cluster centers can become "orphaned". That is, they no longer have any points in them (or perhaps just a single point). A standard method for dealing with this problem is simply to randomly re-initialize that cluster center.

Slide 8.4.27

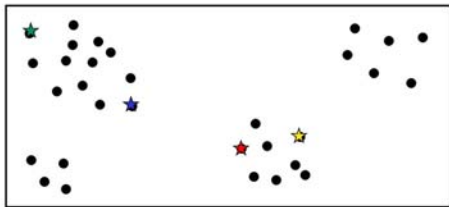
Here's a running example simulation of the k-means algorithm. We start with this set of input points.

K-Means Example



6.034 - Spring 03 • 27

K-Means Example



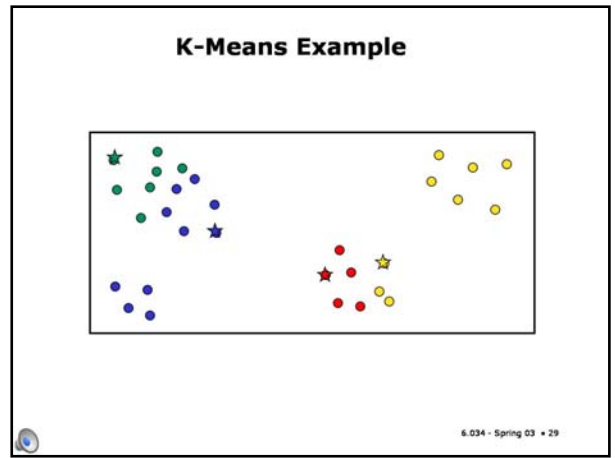
6.034 - Spring 03 • 28

Slide 8.4.28

And randomly pick 4 of them to be our cluster centers.

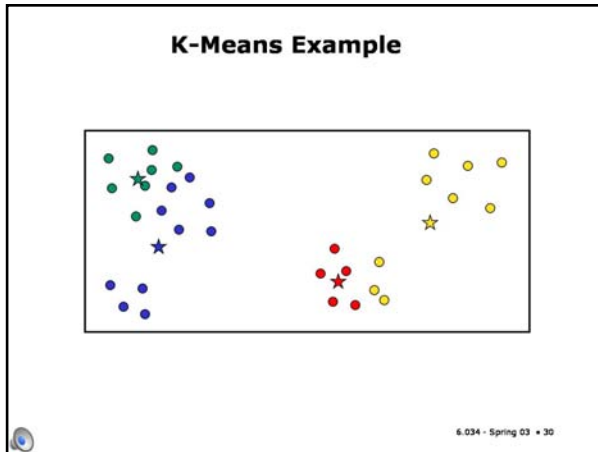
Slide 8.4.29

Now we partition the data, assigning each point to the center to which it is closest.



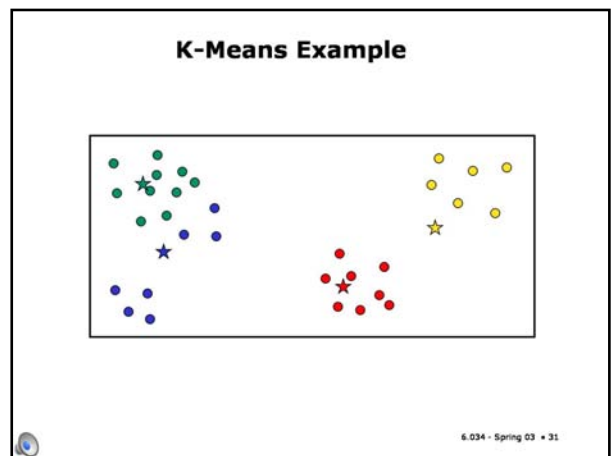
Slide 8.4.30

We move each center to the mean of the points that belong to it.



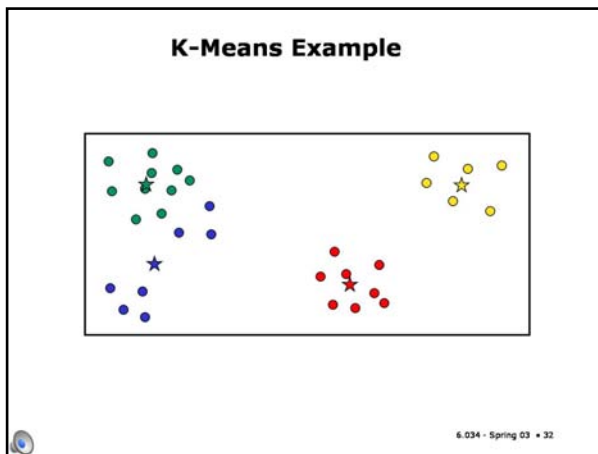
Slide 8.4.31

Having moved the means, we can now do a new reassignment of points.



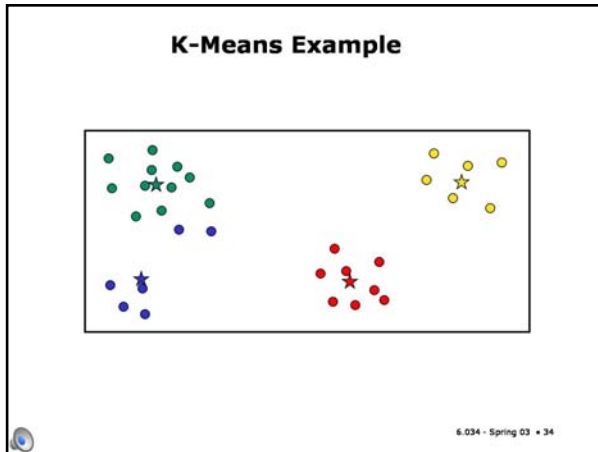
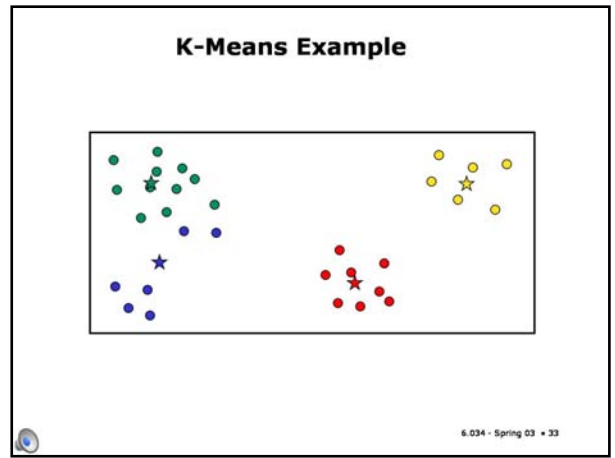
Slide 8.4.32

And recompute the centers.



Slide 8.4.33

Here we reassign one more point to the green cluster,

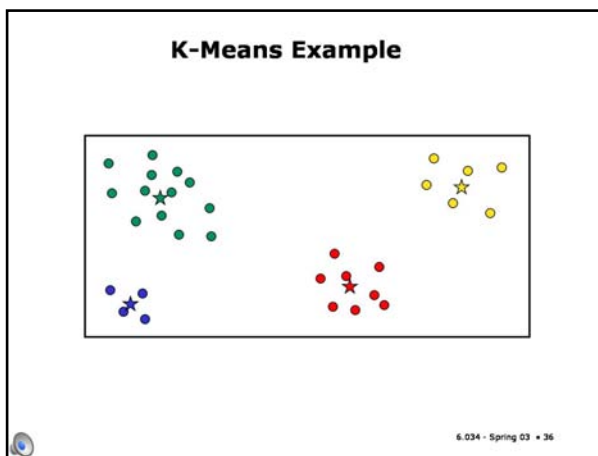
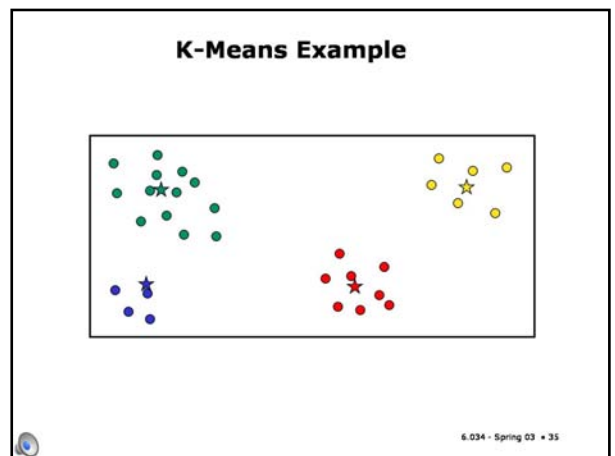


Slide 8.4.34

Which causes the green and blue centers to move a bit. At this point, the red and yellow clusters are stable.

Slide 8.4.35

Now two more points get reassigned to green,



Slide 8.4.36

And we recompute the centers, to get a clustering that is stable, and will not change under further iterations.

Slide 8.4.37

The k-means algorithm takes a real-valued input space and generates a one-dimensional discrete description of the inputs. In principal components analysis, we take a real-valued space, and represent the data in a new multi-dimensional real-valued space with lower dimensionality. The new coordinates are linear combinations of the originals.

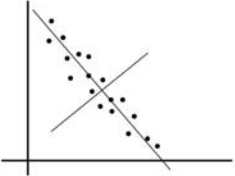
Principal Components Analysis

- Given an n-dimensional real-valued space, data are often nearly restricted to a lower-dimensional subspace
- PCA helps us find such a subspace whose coordinates are linear functions of the originals

6.034 - Spring 03 • 37

Principal Components Analysis

- Given an n-dimensional real-valued space, data are often nearly restricted to a lower-dimensional subspace
- PCA helps us find such a subspace whose coordinates are linear functions of the originals



6.034 - Spring 03 • 38

Slide 8.4.38

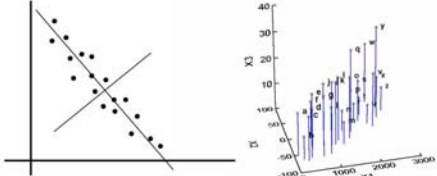
The idea is that even if your data are described using a large number of dimensions, they may lie in a lower-dimensional subspace of the original space. So, in this figure, the data are described with two dimensions, but a single dimension that runs diagonally through the data would describe it without losing too much information.

Slide 8.4.39

It's harder to see in three dimensions, but here's a data set that might be effectively described using only two dimensions.

Principal Components Analysis

- Given an n-dimensional real-valued space, data are often nearly restricted to a lower-dimensional subspace
- PCA helps us find such a subspace whose coordinates are linear functions of the originals



<http://www.okstate.edu/artsci/botany/ordinate/PCA.htm>

Cartoon of algorithm

- Normalize the data (subtract mean, divide by stdev)

6.034 - Spring 03 • 40

Slide 8.4.40

To really understand what's going on in this algorithm, you need to have had linear algebra. We'll just give you a "cartoon" idea of how it works.


We start out by normalizing the data (subtracting the mean and dividing by the standard deviation). The new set of coordinates we construct will have its origin at the centroid of the data.

Slide 8.4.41

Now, we find the single line along which the data have the most variance. It's the dimension that, were we to project the data onto it, would result in the most "spread" of the data. We'll let this be our first principal component.

Cartoon of algorithm


- Normalize the data (subtract mean, divide by stdev)
- Find the line along which the data has the most variability: that's the first principal component



6.034 - Spring 03 • 41

Cartoon of algorithm

- Normalize the data (subtract mean, divide by stdev)
- Find the line along which the data has the most variability: that's the first principal component
- Project the data into the n-1 dimensional space orthogonal to the line
- Repeat



6.034 - Spring 03 • 42

Slide 8.4.42

Now, we project the data down into the n-1 dimensional space that's orthogonal to the line we just chose, and repeat.


Slide 8.4.43

The result of this process is a new set of orthogonal axes. The first k of them give a lower-dimensional space that represents the variability of the data as well as possible.

Cartoon of algorithm

- Normalize the data (subtract mean, divide by stdev)
- Find the line along which the data has the most variability: that's the first principal component
- Project the data into the n-1 dimensional space orthogonal to the line
- Repeat

- Result is a new orthogonal set of axes
- First k give a lower-D space that represents the variability of the data as well as possible




6.034 - Spring 03 • 43

Cartoon of algorithm

- Normalize the data (subtract mean, divide by stdev)
- Find the line along which the data has the most variability: that's the first principal component
- Project the data into the n-1 dimensional space orthogonal to the line
- Repeat

- Result is a new orthogonal set of axes
- First k give a lower-D space that represents the variability of the data as well as possible
- Really: find the eigenvectors of the covariance matrix with the k largest eigenvalues



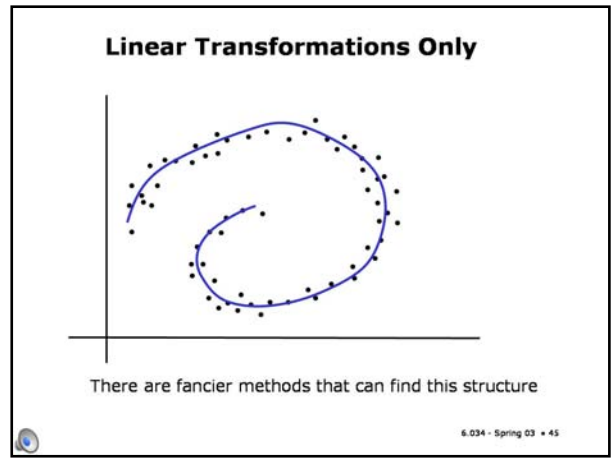
6.034 - Spring 03 • 44

Slide 8.4.44

If you have some experience with linear algebra, then I can tell you that what we really do is find the eigenvectors of the covariance matrix with the k largest eigenvalues.

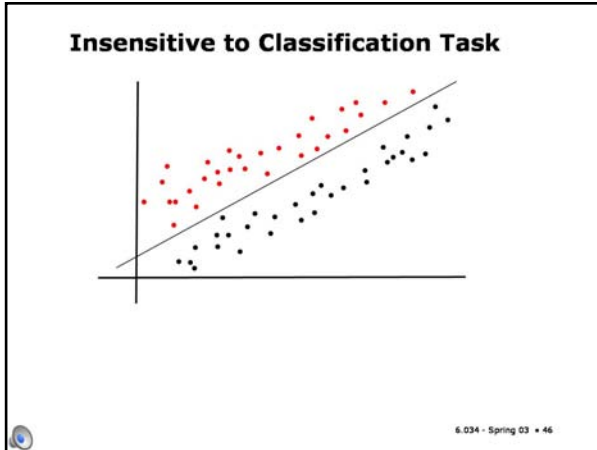
Slide 8.4.45

One problem with PCA (as it's called by its friends) is that it can only produce a set of coordinates that's a linear transformation of the originals. But here's a data set that seems to have a fundamentally one-dimensional structure. Unfortunately, we can't express its axis as a linear combination of the original ones. There are some other cool dimensionality reduction techniques that can actually find this structure!



Slide 8.4.46

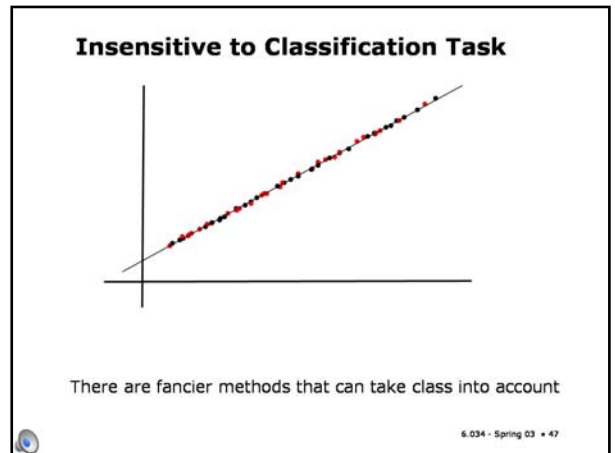
Another problem with PCA is that it (like k-Means clustering) ignores the classes of the points. So, in this example, the principal component is the line that goes between the two classes (it's a great separator, but that's not what we're looking for right now).



Slide 8.4.47

Now, if we project the data onto that line (which is what would happen if we wanted to reduce the dimensionality of our data set to 1), the positive and negative points are completely intermingled, and we can never get a separator.

There are dimensionality-reduction techniques, also, sadly beyond our scope, that try to optimize the discriminability of the data rather than its variability, which don't suffer from this problem.



Slide 8.4.48

We're just going to tack one additional topic onto the end of this section. It has to do with understanding how well a classifier works. So far, we've been thinking about optimizing training error or cross-validation error, where "error" is measured as the number of examples we get wrong. Let's examine this a little more carefully.

In a binary classification problem, on a single example, there are 4 possible outcomes, depending on the true output value for the input and the predicted output value. In this table, we'll assign values A through D to be the number of times each of these outcomes happens on a data set.

Validating a Classifier

		predicted y	
		0	1
true y	0	A	B
	1	C	D

6.034 - Spring 03 • 48

Slide 8.4.49

Case B, in which the answer was supposed to be 0 but the classifier predicted 1 is called a "false positive" or a type 1 error.

Validating a Classifier

		predicted y		
		0	1	
true y	0	A	B	false positive type 1 error
	1	C	D	

6.034 - Spring 03 • 49

Validating a Classifier

		predicted y		
		0	1	
true y	0	A	B	false positive type 1 error
	1	C	D	

false negative
type 2 error

6.034 - Spring 03 • 50

Slide 8.4.50

Case C, in which the answer was supposed to be 1 but the classifier predicted 0 is called a "false negative" or a type 2 error.

Slide 8.4.51

Given these 4 numbers, we can define different characterizations of the classifier's performance. The **sensitivity** is the probability of predicting a 1 when the actual output is 1. This is also called the **true positive rate**, or TP.

Validating a Classifier

		predicted y		
		0	1	
true y	0	A	B	false positive type 1 error
	1	C	D	

false negative
type 2 error

- sensitivity: $P(\text{predict } 1 \mid \text{actual } 1) = D/(C+D)$
- "true positive rate" (TP)

6.034 - Spring 03 • 51

Validating a Classifier

		predicted y		
		0	1	
true y	0	A	B	false positive type 1 error
	1	C	D	

false negative
type 2 error

- sensitivity: $P(\text{predict } 1 \mid \text{actual } 1) = D/(C+D)$
- "true positive rate" (TP)
- specificity: $P(\text{predict } 0 \mid \text{actual } 0) = A/(A+B)$

6.034 - Spring 03 • 52

Slide 8.4.52

The **specificity** is the probability of predicting a 0 when the actual output is 0.

Slide 8.4.53

The **false-alarm rate** is the probability of predicting a 1 when the actual output is 0. This is also called the **false positive rate**, or FP.

Classifiers are usually characterized using sensitivity and specificity, or using TP and FP.

Validating a Classifier

		predicted y		
		0	1	
true y	0	A	B	false positive type 1 error
	1	C	D	

false negative
type 2 error

- sensitivity: $P(\text{predict } 1 \mid \text{actual } 1) = D/(C+D)$
 - "true positive rate" (TP)
- specificity: $P(\text{predict } 0 \mid \text{actual } 0) = A/(A+B)$
- false-alarm rate: $P(\text{predict } 1 \mid \text{actual } 0) = B/(A+B)$
 - "false positive rate" (FP)

6.034 - Spring 03 • 53

Cost Sensitivity

- Predict whether a patient has pseuditis based on blood tests
 - Disease is often fatal if left untreated
 - Treatment is cheap and side-effect free

6.034 - Spring 03 • 54

Slide 8.4.54

Imagine that you're a physician and you need to predict whether a patient has pseuditis based on the results of some blood tests. The disease is often fatal if it's left untreated, and the treatment is cheap and relatively side-effect free.

Slide 8.4.55

You have two different classifiers that you could use to make the decision. The first has a true-positive rate of 0.9 and a false-positive rate of 0.4. That means that it will diagnose the disease in 90 percent of the people who actually have it; and also diagnose it in 40 percent of people who don't have it.

Cost Sensitivity

- Predict whether a patient has pseuditis based on blood tests
 - Disease is often fatal if left untreated
 - Treatment is cheap and side-effect free
- Which classifier to use?
 - Classifier 1: TP = 0.9, FP = 0.4

6.034 - Spring 03 • 55

Cost Sensitivity

- Predict whether a patient has pseuditis based on blood tests
 - Disease is often fatal if left untreated
 - Treatment is cheap and side-effect free
- Which classifier to use?
 - Classifier 1: TP = 0.9, FP = 0.4
 - Classifier 2: TP = 0.7, FP = 0.1

6.034 - Spring 03 • 56

Slide 8.4.56

The second classifier only has a true-positive rate of 0.7, but a more reasonable false positive rate of 0.1.

Given the set-up of the problem, we might choose classifier 1, since all those false positives aren't too costly (but if it causes too much hassle, per patient, we might not want to bring 40 percent of them back for treatment).

Slide 8.4.57

One way to address this problem is to start by figuring out the relative costs of the two types of errors. Then, for many classifiers, we can build these costs directly into the choice of classification.

In decision trees, we could use a different splitting criterion. For neural networks we could change the error function to be asymmetric. In SVM's, we could use two different values of C.

Build Costs into Classifier

- Assess costs of both types of error
 - use a different splitting criterion for decision trees
 - make error function for neural nets asymmetric; different costs for each kind of error
 - use different values of C for SVMs depending on kind of error

6.034 - Spring 03 • 57

Tunable Classifiers

- Classifiers that have a threshold (naïve Bayes, neural nets, SVMs) can be adjusted, post learning, by changing the threshold, to make different trade-offs between type 1 and type 2 errors

6.034 - Spring 03 • 58

Slide 8.4.58

Often it's useful to deliver a classifier that is tunable. That is, a classifier that has a parameter in it that can be used, at application time, to change the trade-offs made between type 1 and type 2 errors. Most classifiers that have a threshold (such as naive Bayes, neural nets, or SVMs), can be tuned by changing the threshold. At different values of the threshold the classifier will tend to make more errors of one type versus the other.

Slide 8.4.59

In a particular application, we can choose a threshold as follows.

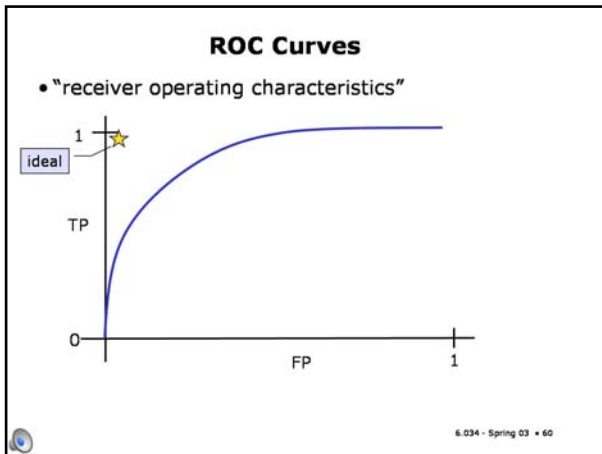
Let c_1 and c_2 be the costs of the two different types of errors; let p be the percentage of positive examples, let x be the threshold parameter that we are allowed to tune, and let $TP(x)$ and $FP(x)$ be the true-positive and false-positive rates, respectively, of the classifier when the threshold is set to have value x .

Then, we can characterize the average, or expected, cost based on this formula, as a function of x . We should choose the value of x that will minimize expected cost.

Tunable Classifiers

- Classifiers that have a threshold (naïve Bayes, neural nets, SVMs) can be adjusted, post learning, by changing the threshold, to make different trade-offs between type 1 and type 2 errors
 - C_1, C_2 : costs of errors
 - P : percentage of positive examples
 - x : tunable threshold
 - $TP(x)$: true positive rate at threshold x
 - $FP(x)$: false positive rate at threshold x
 - Expected Cost = $C_1P(1-TP(x)) + C_2(1-P)FP(x)$
 - choose x to minimize expected cost

6.034 - Spring 03 • 59



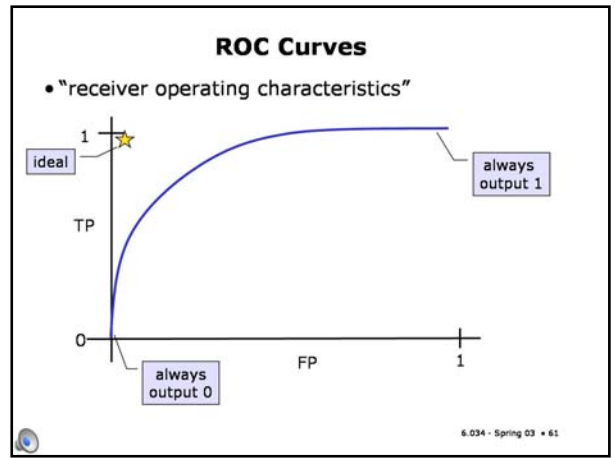
Slide 8.4.60

One way to see the overall performance of a tunable classifier is with a ROC curve. ROC stands for "receiver operating characteristics" from the days of the invention of radar.

An ROC curve is plotted on two axes; the x axis is the false positive rate and the y axis is the true positive rate. In an ideal world, we would have a false positive rate of 0 and a true positive rate of 1, which would put our performance up near the star on this graph.

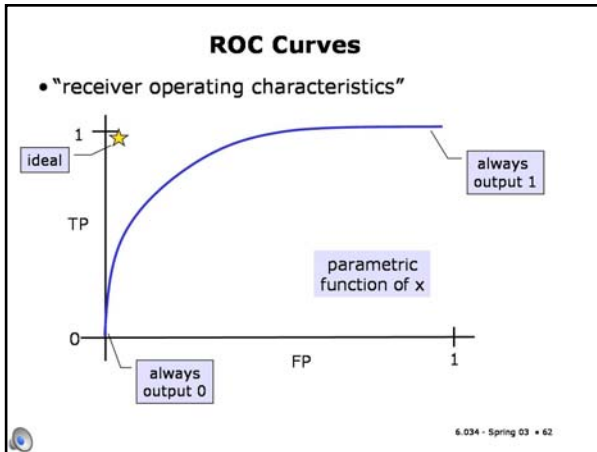
Slide 8.4.61

In reality, as we adjust the parameter in the classifier, we typically go from a situation in which the classifier always outputs 0, which generates no false positives and no true positives, to a situation in which the classifier always outputs 1, in which case we have both false positive and true positive rates of 1.



Slide 8.4.62

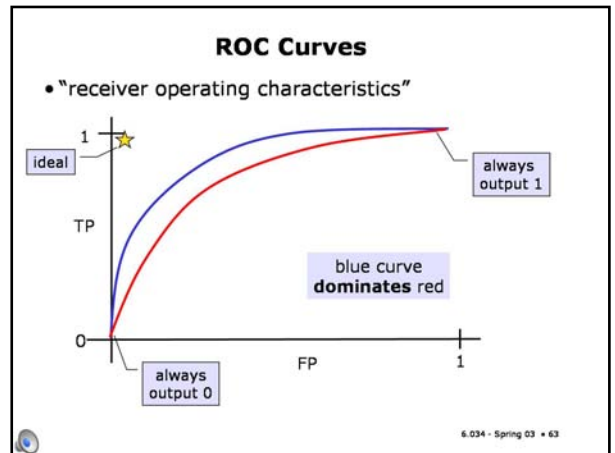
The ROC curve itself is a parametric curve; for each value of x , we plot the pair $FP(x), TP(x)$. The curve shows the range of possible behaviors of the classifier. It is typically shaped something like this blue curve; the higher the false positive rate we can stand, the higher the rate of detecting true positives we can achieve.



Slide 8.4.63

Often it is useful to compare two different classifiers by comparing their ROC curves. If we're lucky, then one curve is always higher than the other. In such a situation, we'd say that the blue curve **dominates** the red curve. That means that, no matter what costs apply in our domain, it will be better to use the blue classifier (because, for any fixed rate of false positives, the blue classifier can achieve more true positives; or for any fixed rate of true positives, the blue classifier can always achieve fewer false positives).

If the curves cross, then it will be better to use one classifier in some cost situations and the other classifier in other situations.



Slide 8.4.64

Machine learning is a huge field that we have just begun to cover. Even in the context of supervised learning, there are a variety of other issues, including how to handle missing data, what to do when you have very many negative examples and just a few positives (such as when you're trying to detect fraud), what to do when getting y values for your x 's is very expensive (you might actively choose which y 's you'd like to have labeled), and many others.

If you like this topic, take a probability course, and then take the graduate machine learning course.

Many more issues!

- Missing data
- Many examples in one class, few in other (fraud detection)
- Expensive data (active learning)
- ...

The slide number "6.034 - Spring 03 • 64" is in the bottom right.