

6.034 Notes: Section 4.1

Slide 4.1.1

So far, we have talked a lot about building systems that have knowledge represented in them explicitly. One way to acquire that knowledge is to build it in by hand. But that can be time-consuming and error prone. And many times, humans just don't have the relevant information available to them.

For instance, you are all experts in visual object recognition. But it's unlikely that you could sit down and write a program that would do the job even remotely as well as you can.

But, in lots of cases, when we don't have direct access to a formal description of a problem, we can learn something about it from examples.

Learning

- It is often hard to articulate the knowledge we need to build AI systems
- Often, we don't even know it!
- Frequently, we can arrange to build systems that learn it themselves

6.034 - Spring 03 • 1

What is Learning?

- memorizing something
- learning facts through observation and exploration
- improving motor and/or cognitive skills through practice
- organizing new knowledge into general, effective representations

6.034 - Spring 03 • 2

Slide 4.1.2

The word "learning" has many different meanings. It is used, at least, to describe

- memorizing something
- learning facts through observation and exploration
- development of motor and/or cognitive skills through practice
- organization of new knowledge into general, effective representations

Slide 4.1.3

Herb Simon (an important historical figure in AI and in Economics) gave us this definition of learning:

Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the task or tasks drawn from the same population more efficiently and more effectively the next time.

This is not entirely precise, but it gives us the idea that learning systems have to acquire some information from examples of a problem, and perform better because of it (so we wouldn't call a system that simply logs all the images it has ever seen a learning system, because, although, in some sense, it knows a lot, it can't take advantage of its knowledge).

What is Learning?

- memorizing something
- learning facts through observation and exploration
- improving motor and/or cognitive skills through practice
- organizing new knowledge into general, effective representations

"Learning denotes changes in the system that are adaptive in the sense that they enable the system to do the task or tasks drawn from the same population more efficiently and more effectively the next time." -- Herb Simon

6.034 - Spring 03 • 3

Induction

Piece of bread 1 was nourishing when I ate it
 Piece of bread 2 was nourishing when I ate it.
 ...
 Piece of bread 100 was nourishing when I ate it.
Therefore, all pieces of bread will be nourishing if I eat them.

David Hume

6.034 - Spring 03 • 4

Slide 4.1.4

One of the most common kinds of learning is the acquisition of information with the goal of making predictions about the future. But what exactly gives us license to imagine we can predict the future? Lots of philosophers have thought about this problem. David Hume first framed the problem of induction as follows:

- Piece of bread number 1 was nourishing when I ate it.
- Piece of bread number 2 was nourishing when I ate it.
- Piece of bread number 3 was nourishing when I ate it.
- Piece of bread number 100 was nourishing when I ate it.
- *Therefore*, all pieces of bread will be nourishing if I eat them.

Slide 4.1.5

And here, for no good reason, is a photo of David Hume's Tomb, just because I saw it once in Edinburgh.

Induction

Piece of bread 1 was nourishing when I ate it
 Piece of bread 2 was nourishing when I ate it.
 ...
 Piece of bread 100 was nourishing when I ate it.
Therefore, all pieces of bread will be nourishing if I eat them.

David Hume

6.034 - Spring 03 • 5

Why is Induction Okay?

It has been argued that we have reason to know the future will resemble the past, because what was the future has constantly become the past, and has always been found to resemble the past, so that we really have experience of the future, namely of times which were formerly future, which we may call past futures. But such an argument really begs the very question at issue. We have experience of past futures, but not of future futures, and the question is: Will future futures resemble past futures?

Bertrand Russell

<http://www.ditext.com/russell/rus6.html>

6.034 - Spring 03 • 6

Slide 4.1.6

When and why is it okay to apply inductive reasoning??

My favorite quote on the subject is the following, excerpted from Bertrand Russell's "On Induction": (see <http://www.ditext.com/russell/rus6.html> for the whole thing)

If asked why we believe the sun will rise tomorrow, we shall naturally answer, 'Because it has always risen every day.' We have a firm belief that it will rise in the future, because it has risen in the past.

The real question is: Do any number of cases of a law being fulfilled in the past afford evidence that it will be fulfilled in the future?

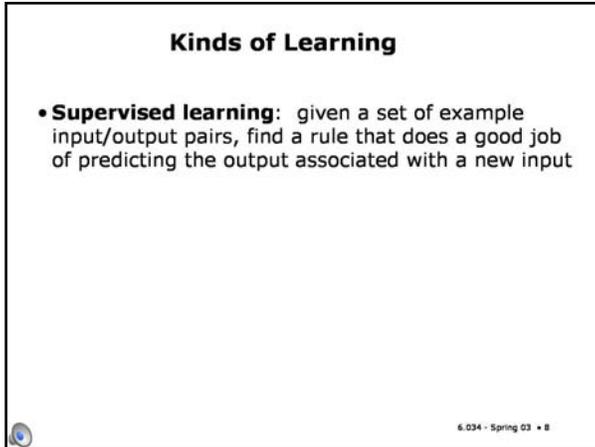
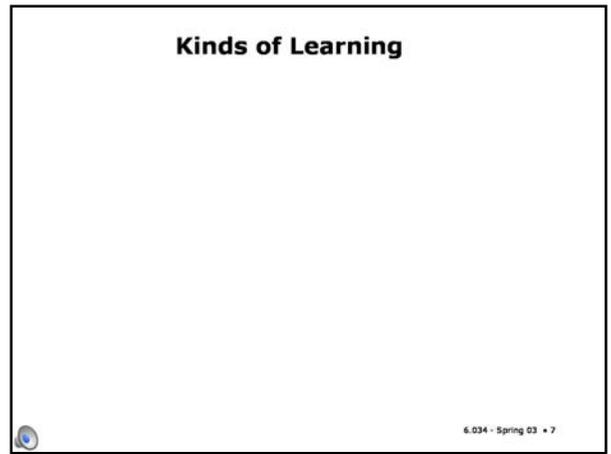
It has been argued that we have reason to know the future will resemble the past, because what was the future has constantly become the past, and has always been found to resemble the past, so that we really have experience of the future, namely of times which were formerly future, which we may call past futures. But such an argument really begs the very question at issue. We have experience of past futures, but not of future futures, and the question is: Will future futures resemble past futures?

no reason to get out of bed in the morning, let alone study machine learning!

We won't worry too much about this problem. If induction is not, somehow, justified, then we have

Slide 4.1.7

When viewed technically, there are lots of different kinds of machine learning problems. We'll just sketch them out here, so you get an idea of their range.

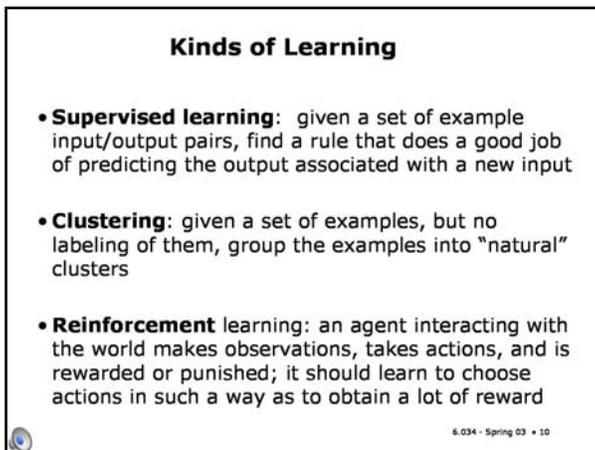
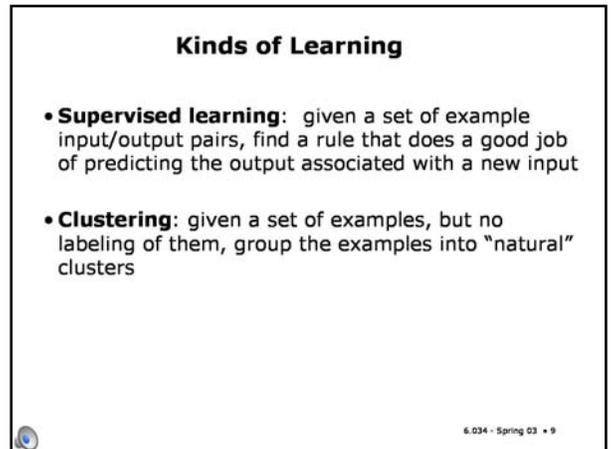


Slide 4.1.8

Supervised learning is the most common learning problem. Let's say you are given the weights and lengths of a bunch of individual salmon fish, and the weights and lengths of a bunch of individual tuna fish. The job of a supervised learning system would be to find a predictive rule that, given the weight and length of a fish, would predict whether it was a salmon or a tuna.

Slide 4.1.9

Another, somewhat less well-specified, learning problem is **clustering**. Now you're given the descriptions of a bunch of different individual animals (or stars, or documents) in terms of a set of features (weight, number of legs, presence of hair, etc), and your job is to divide them into groups, or possibly into a hierarchy of groups that "makes sense". What makes this different from supervised learning is that you're not told in advance what groups the animals should be put into; just that you should find a natural grouping.



Slide 4.1.10

Another learning problem, familiar to most of us, is learning motor skills, like riding a bike. We call this **reinforcement learning**. It's different from supervised learning because no-one explicitly tells you the right thing to do; you just have to try things and see what makes you fall over and what keeps you upright.

Supervised learning is the most straightforward, prevalent, and well-studied version of the learning problem, so we are going to start with it, and spend most of our time on it. Most of the fundamental insights into machine learning can be seen in the supervised case.

Slide 4.1.11

One way to think about learning is that you're trying to find the definition of a function, given a bunch of examples of its input and output. This might seem like a pretty narrow definition, but it actually covers a lot of cases.

Learning a Function

Given a set of examples of input/output pairs, find a function that does a good job of expressing the relationship



6.034 - Spring 03 • 11

Learning a Function

Given a set of examples of input/output pairs, find a function that does a good job of expressing the relationship

- Pronunciation: function from letters to sounds
- Throw a ball: function from target locations to joint torques
- Read handwritten characters: function from collections of image pixels to letters
- Diagnose diseases: function from lab test results to disease categories



6.034 - Spring 03 • 12

Slide 4.1.12

- Learning how to pronounce words can be thought of as finding a function from letters to sounds.
- Learning how to throw a ball can be thought of as finding a function from target locations to joint torques.
- Learning to recognize handwritten characters can be thought of as finding a function from collections of image pixels to letters.
- Learning to diagnose diseases can be thought of as finding a function from lab test results to disease categories.

Slide 4.1.13

The problem of learning a function from examples, is complicated. You can think of at least three different problems being involved: memory, averaging, and generalization. We'll look at each of these problems, by example.

Aspects of Function Learning

- memory
- averaging
- generalization



6.034 - Spring 03 • 13

Example problem

When to drive the car? Depends on:

- temperature
- expected precipitation
- day of the week
- whether she needs to shop on the way home
- what she's wearing



6.034 - Spring 03 • 14

Slide 4.1.14

We'll do this in the context of a silly example problem. Imagine that I'm trying predict whether my neighbor is going to drive into work tomorrow, so I can ask for a ride. Whether she drives into work seems to depend on the following attributes of the day: the temperature, what kind of precipitation is expected, the day of the week, whether she needs to shop on the way home, and what she's wearing.

Slide 4.1.15

Okay. Let's say we observe our neighbor on three days, which are described in the table, which specifies the properties of the days and whether or not the neighbor walked or drove.

Memory

temp	precip	day	shop	clothes	
80	none	sat	no	casual	walk
19	snow	mon	yes	casual	drive
65	none	tues	no	casual	walk

6.034 - Spring 03 • 15

Memory

temp	precip	day	shop	clothes	
80	none	sat	no	casual	walk
19	snow	mon	yes	casual	drive
65	none	tues	no	casual	walk
19	snow	mon	yes	casual	

6.034 - Spring 03 • 16

Slide 4.1.16

Now, we find ourselves on a snowy 19-degree Monday, when the neighbor is wearing casual clothes and going shopping. Do you think she's going to drive?

Slide 4.1.17

The standard answer in this case is "yes". This day is just like one of the ones we've seen before, and so it seems like a good bet to predict "yes." This is about the most rudimentary form of learning, which is just to memorize the things you've seen before. Still, it can help you do a better job in the future, if those same cases arise again.

Memory

temp	precip	day	shop	clothes	
80	none	sat	no	casual	walk
19	snow	mon	yes	casual	drive
65	none	tues	no	casual	walk
19	snow	mon	yes	casual	drive

6.034 - Spring 03 • 17

Averaging

Dealing with noise in the data

temp	precip	day	shop	clothes	
80	none	sat	no	casual	walk
80	none	sat	no	casual	walk
80	none	sat	no	casual	drive
80	none	sat	no	casual	drive
80	none	sat	no	casual	walk
80	none	sat	no	casual	walk
80	none	sat	no	casual	walk

6.034 - Spring 03 • 18

Slide 4.1.18

Things are not always as easy as they were in the previous case. What if you get this set of data?

Slide 4.1.23

Take a good look at this data set.

Generalization

Dealing with previously unseen cases

temp	precip	day	shop	clothes	
71	none	fri	yes	formal	drive
36	none	sun	yes	casual	walk
62	rain	weds	no	casual	walk
93	none	mon	no	casual	drive
55	none	sat	no	formal	drive
80	none	sat	no	casual	walk
19	snow	mon	yes	casual	drive
65	none	tues	no	casual	walk

6.034 - Spring 03 • 23

Generalization

Dealing with previously unseen cases

temp	precip	day	shop	clothes	
71	none	fri	yes	formal	drive
36	none	sun	yes	casual	walk
62	rain	weds	no	casual	walk
93	none	mon	no	casual	drive
55	none	sat	no	formal	drive
80	none	sat	no	casual	walk
19	snow	mon	yes	casual	drive
65	none	tues	no	casual	walk
58	rain	mon	no	casual	

6.034 - Spring 03 • 24

Slide 4.1.24

Now, consider this day. It's 58 degrees and raining on a Monday. The neighbor is wearing casual clothing and doesn't need to shop. Will she walk or drive?

The first thing to observe is that there's no obviously right answer. We have never seen this case before. We could just throw up our hands and decline to make a prediction. But ultimately, we have to decide whether or not to call the neighbor. So, we might fall back on some assumptions about the domain. We might assume, for instance, that there's a kind of *smoothness* property: similar situations will tend to have similar categories.

You might plausibly make any of the following arguments:

- She's going to walk because it's raining today and the only other time it rained, she walked.
- She's going to drive because she has always driven on Mondays.
- She's going to walk because she only drives if she is wearing formal clothes, or if the temperature is above 90 or below 20.

The question of which one to choose is hard. It's one of the deep underlying problems of machine learning. We'll look at some examples to try to get more intuition, then come back and revisit it a bit

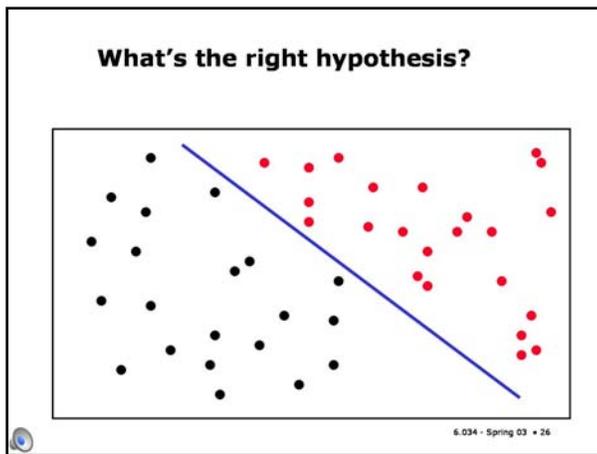
more formally in the next section.

Slide 4.1.25

Imagine that you were given all these points, and you needed to guess a function of their x, y coordinates that would have one output for the red ones and a different output for the black ones.

The Red and the Black

6.034 - Spring 03 • 25

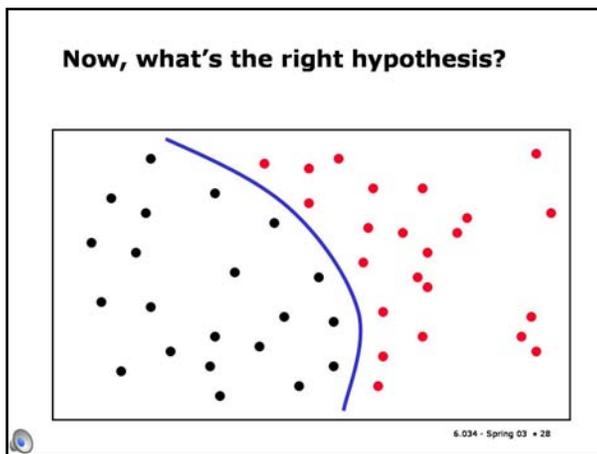
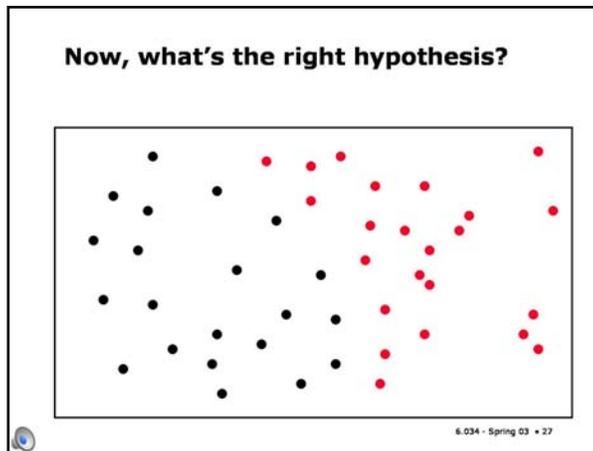


Slide 4.1.26

In this case, it seems like you could do pretty well by defining a line that separates the two classes.

Slide 4.1.27

Now, what if we have a slightly different configuration of points? We can't divide them conveniently with a line.

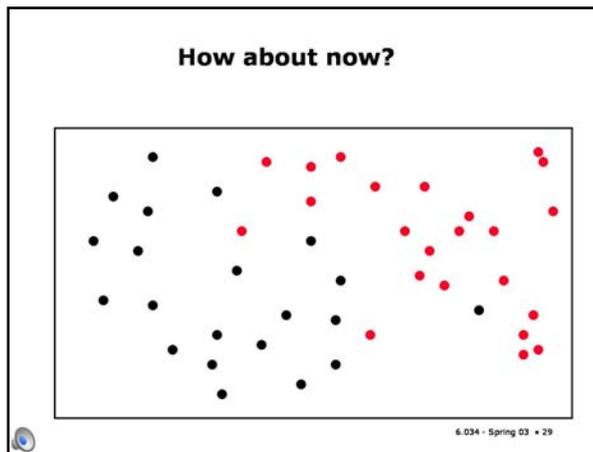


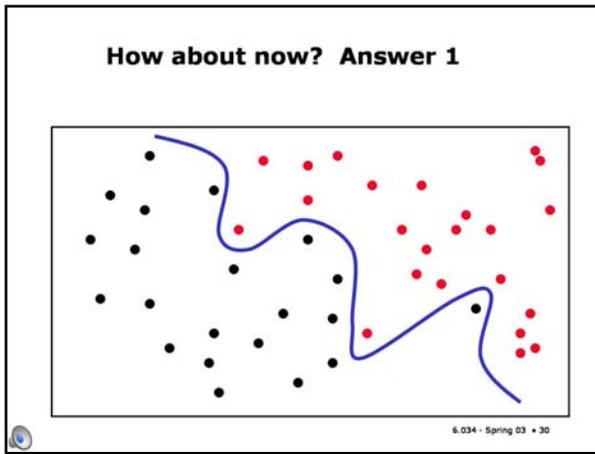
Slide 4.1.28

But this parabola-like curve seems like it might be a reasonable separator.

Slide 4.1.29

Now, what? This seems much trickier. There are at least a couple of reasonable kinds of answers.





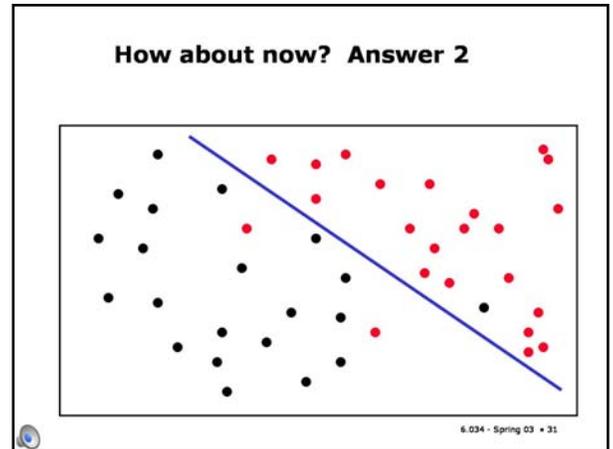
Slide 4.1.30

Here's an answer. It successfully separates the reds from the blacks. But there's something sort of unsatisfactory about it. Most people have an intuition that it's too complicated.

Slide 4.1.31

Here's another answer. It's not a perfect separator; it gets some of the points wrong. But it's prettier.

In general, we will always be faced with making trade-offs between hypotheses that account for the data perfectly and hypotheses that are, in some sense, simple. There are some mathematical ways to understand these trade-offs; we'll avoid the math, but try to show you, intuitively, how they come up and what to do about them.



Slide 4.1.32

There are lots and lots of different kinds of learning algorithms. I want to show you cartoon versions of a couple of them first, and then we'll get down to the business of understanding them at a detailed algorithmic level.

The learning methods differ both in terms of the kinds of hypotheses they work with and the algorithms they use to find a good hypothesis given the data.

Variety of Learning Methods

Learning methods differ in terms of:

- the form of the hypothesis
- the way the computer finds a hypothesis given the data

6.034 - Spring 03 • 32

Slide 4.1.33

Here's the most simple-minded of all learning algorithms (and my personal favorite (which might imply something about me!)). It's called nearest neighbor (or lazy learning). You simply remember all the examples you've ever seen.

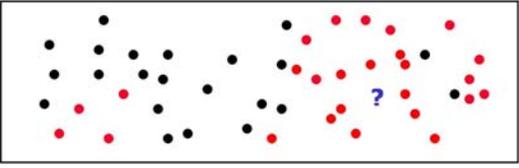
Nearest Neighbor

- Remember all your data

6.034 - Spring 03 • 33

Nearest Neighbor

- Remember all your data
- When someone asks a question,
 - find the nearest old data point
 - return the answer associated with it



6.034 - Spring 03 • 34

Slide 4.1.34

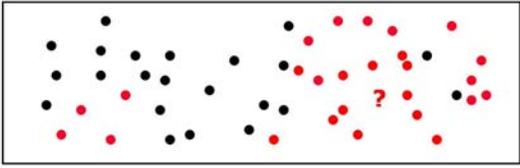
When someone asks you a question, you simply find the existing point that's nearest the one you were asked about, and return its class.

Slide 4.1.35

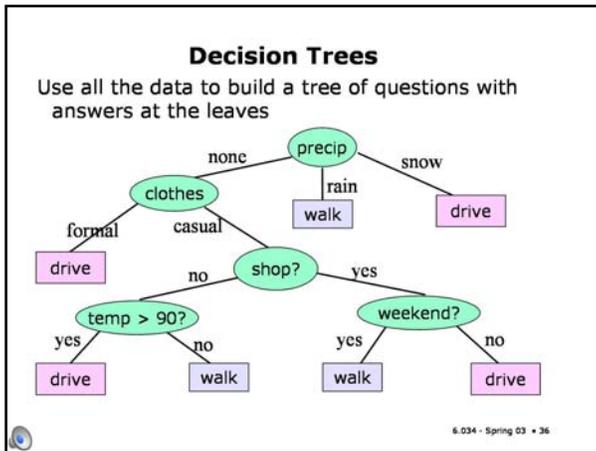
So, in this case, the nearest point to the query is red, so we'd return red.

Nearest Neighbor

- Remember all your data
- When someone asks a question,
 - find the nearest old data point
 - return the answer associated with it



6.034 - Spring 03 • 35

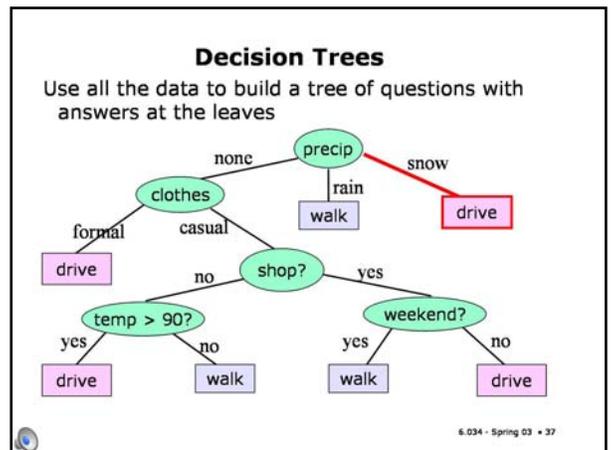


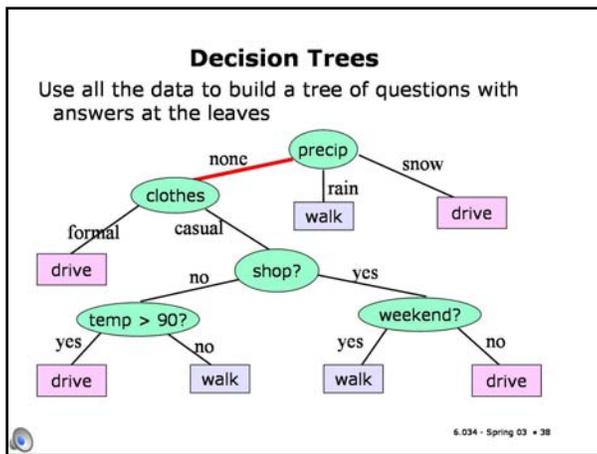
Slide 4.1.36

Another well-loved learning algorithm makes hypotheses in the form of decision trees. In a decision tree, each node represents a question, and the arcs represent possible answers. We can use this decision tree to find out what prediction we should make in the drive/walk problem.

Slide 4.1.37

We'd start by asking what the current precipitation is. If it's snow, we just stop asking questions and drive.





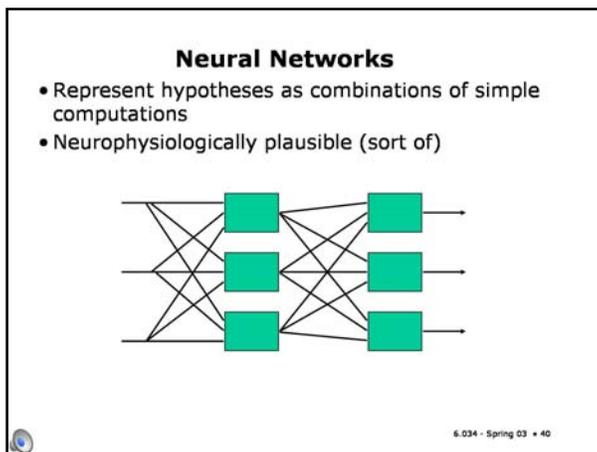
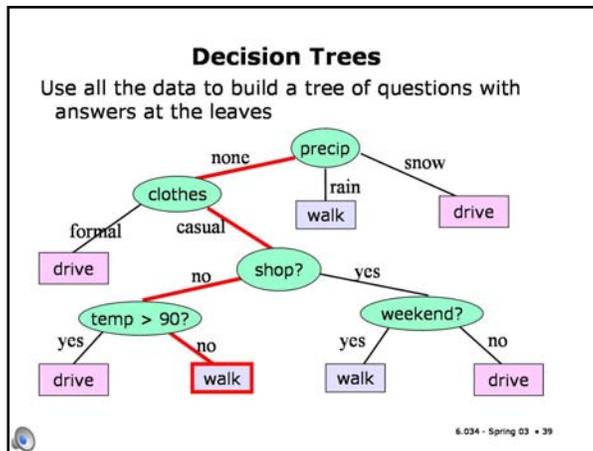
Slide 4.1.38

If there's no precipitation, then we have to ask what kind of clothes the neighbor is wearing. If they're formal, she'll drive. If not, we have to ask another question.

Slide 4.1.39

We can always continue asking and answering questions until we get to a leaf node of the tree; the leaf will always contain a prediction.

Hypotheses like this are nice for a variety of reasons. One important one is that they're relatively easily interpretable by humans. So, in some cases, we run a learning algorithm on some data and then show the results to experts in the area (astronomers, physicians), and they find that the learning algorithm has found some regularities in their data that are of real interest to them.

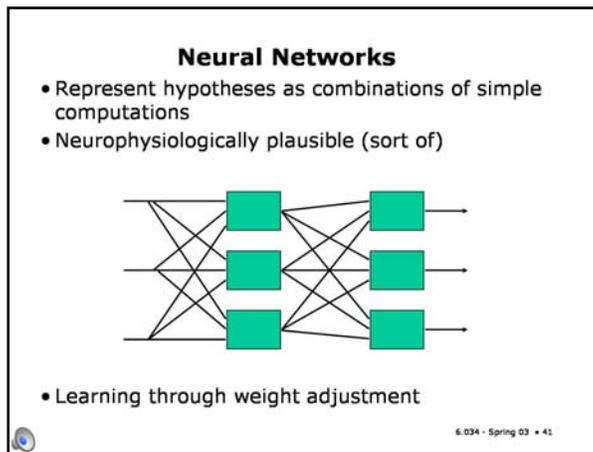


Slide 4.1.40

Most people have heard of neural networks. They can represent complicated hypotheses in high-dimensional continuous spaces. They are attractive as a computational model because they are composed of many small computing units. They were motivated by the structure of neural systems in parts of the brain. Now it is understood that they are not an exact model of neural function, but they have proved to be useful from a purely practical perspective.

Slide 4.1.41

In neural networks, the individual units do a simple, fixed computation that combines the values coming into them with "weights" on the arcs. The learning process consists of adjusting the weights on the arcs in reaction to errors made by the network.



Machine Learning Successes

- assessing loan credit risk
- detecting credit card fraud
- cataloging astronomical images
- detecting and diagnosing manufacturing faults
- helping NBA coaches analyze performance
- personalizing news and web searches
- steering an autonomous car across the US

Slide 4.1.42

Machine learning methods have been successfully fielded in a variety of applications, including

- assessing loan credit risk
- detecting credit card fraud
- cataloging astronomical images
- deciding which catalogs to send to your house
- helping NBA coaches analyze players' performance
- personalizing news and web searches
- steering an autonomous car across the US

Now, we'll roll up our sleeves, and see how all this is done.

6.034 Notes: Section 4.2

Slide 4.2.1

A supervised learning problem is made up of a number of ingredients. The first is the data (sometimes called the training set). The data, D , is a set of m input-output pairs. We will write the i th element of the data set as $\langle x^i, y^i \rangle$. In the context of our old running example, an x^i would be a vector of values, one for each of the input features, and a y^i would be an output value (walk or drive).

Supervised Learning

- Given data (training set)

$$D = \{ \langle x^1, y^1 \rangle, \langle x^2, y^2 \rangle, \dots, \langle x^m, y^m \rangle \}$$

Supervised Learning

- Given data (training set)

$$D = \{ \langle x^1, y^1 \rangle, \langle x^2, y^2 \rangle, \dots, \langle x^m, y^m \rangle \}$$

$$\langle x_1^1, x_2^1, \dots, x_n^1 \rangle$$

input

Slide 4.2.2

Each x^i is a vector of n values. We'll write x_j^i for the j th feature of the i th input-output pair. We'll consider different kinds of features. Sometimes we'll restrict ourselves to the case where the features are only 0s and 1s. Other times, we'll let them be chosen from a set of discrete elements (like "snow", "rain", "none"). And, still other times, we'll let them be real values, like temperature, or weight.

Slide 4.2.3

Similarly, the output, y^i , might be a boolean, a member of a discrete set, or a real value. For a particular problem, the y^i will all be of a particular type.

When y^i is a boolean, or a member of a discrete set, we will call the problem a **classification** problem. When y^i is real-valued, we call this a **regression** problem.

Supervised Learning

- Given data (training set)

$$D = \{ \langle x^1, y^1 \rangle, \langle x^2, y^2 \rangle, \dots, \langle x^m, y^m \rangle \}$$

Classification: discrete Y
Regression: continuous Y

6.034 - Spring 03 • 3

Supervised Learning

- Given data (training set)

$$D = \{ \langle x^1, y^1 \rangle, \langle x^2, y^2 \rangle, \dots, \langle x^m, y^m \rangle \}$$

Classification: discrete Y
Regression: continuous Y

- Goal: find a hypothesis h in hypothesis class H that does a good job of mapping x to y

6.034 - Spring 03 • 4

Slide 4.2.4

Now, the goal of learning will be to find a hypothesis, h , that does a good job of describing the relationship between the inputs and the outputs. So, a part of the problem specification is capital H , the hypothesis class. H is the set of possible hypotheses that we want our learning algorithm to choose from. It might be something like decision trees with 6 nodes, or lines in two-dimensional space, or neural networks with 3 components.

Slide 4.2.5

So, now we have a bunch of data, and a class of possible answers (hypotheses) and we're supposed to return the best one. In order to do that, we need to know exactly what is meant by "best".

Best Hypothesis

6.034 - Spring 03 • 5

Best Hypothesis

Hypothesis should

- do a good job of describing the data
- not be too complex

6.034 - Spring 03 • 6

Slide 4.2.6

There are typically two components to the notion of best. The hypothesis should do a good job of describing the data and it shouldn't be too complicated.

Slide 4.2.7

Ideally, we would like to find a hypothesis h such that, for all data points i , $h(x^i) = y^i$. We will not always be able to (or maybe even want to) achieve this, so perhaps it will only be true for most of the data points, or the equality will be weakened into "not too far from".

We can sometimes develop a measure of the "error" of a hypothesis to the data, written $E(h, D)$. It might be the number of points that are miscategorized, for example.

Best Hypothesis

Hypothesis should

- do a good job of describing the data
 - ideally: $h(x^i) = y^i$
 - number of errors: $E(h, D)$
- not be too complex

6.034 - Spring 03 • 7

Best Hypothesis

Hypothesis should

- do a good job of describing the data
 - ideally: $h(x^i) = y^i$
 - number of errors: $E(h, D)$
- not be too complex
 - measure: $C(h)$

6.034 - Spring 03 • 8

Slide 4.2.8

Another issue is the complexity of the hypothesis. We can measure the complexity of a decision tree by the number of nodes it has, or a line by how bendy it is. In general, we'll define a measure of the complexity of hypotheses in H , $C(h)$.

Slide 4.2.9

Why do we care about hypothesis complexity? We have an intuitive sense that, all things being equal, simpler hypotheses are better. Why is that? There are lots of statistical and philosophical and information-theoretic arguments in favor of simplicity.

So, for now, let's just make recourse to William of Ockham, a 14th century Franciscan theologian, logician, and heretic. He is famous for "Ockham's Razor", or the principle of parsimony: "Non sunt multiplicanda entia praeter necessitatem." That is, "entities are not to be multiplied beyond necessity". People interpret this to mean that we should always adopt the simplest satisfactory hypothesis.

Best Hypothesis

Hypothesis should

- do a good job of describing the data
 - ideally: $h(x^i) = y^i$
 - number of errors: $E(h, D)$
- not be too complex
 - measure: $C(h)$



William of Ockham

Non sunt multiplicanda entia praeter necessitatem

6.034 - Spring 03 • 9

Best Hypothesis

Hypothesis should

- do a good job of describing the data
 - ideally: $h(x^i) = y^i$
 - number of errors: $E(h, D)$
- not be too complex
 - measure: $C(h)$

Minimize $E(h, D) + \alpha C(h)$

trade-off



William of Ockham

Non sunt multiplicanda entia praeter necessitatem

6.034 - Spring 03 • 10

Slide 4.2.10

So, given a data set, D , a hypothesis class H , a goodness-of-fit function E , and a complexity measure C , our job will be to find the h in H that minimizes $E(h, D) + \alpha * C(h)$, where α is a number that we can vary to change how much we value fitting the data well versus having a complex hypothesis.

We won't, in general, be able to do this efficiently, so, as usual, we'll have to make a lot of approximations and short-cuts. In fact, most learning algorithms aren't described exactly this way. But it's the underlying principle behind most of what we do.

Slide 4.2.11

How can we go about finding the hypothesis with the least value of our criterion? We can certainly think of this as a search problem, although many learning algorithms don't seem to have the character of doing a search.

For some carefully-chosen combinations of hypothesis classes and error criteria, we can calculate the optimal hypothesis directly.

In other cases, the hypothesis space can be described with a vector of real-valued parameters, but for which we can't calculate the optimal values. A common practice for these problems is to do a kind of local hill-climbing search called gradient ascent (or descent).

Finally, sometimes it is possible to construct a hypothesis iteratively, starting with a simple hypothesis and adding to it in order to make it fit the data better. These methods are often called "greedy", because they start by trying to pick the best simple hypothesis, and then make the best addition to it, etc. The result isn't optimal, but it is often very good. We will start by looking at some methods of this kind.

Learning as Search

- How can we find the hypothesis with the lowest value of $E(h, D) + \alpha C(h)$? Search!
- For some hypothesis classes we can calculate the optimal h directly! (linear separators)
- For others, do local search (gradient descent in neural networks)
- For some structured hypothesis spaces, construct one greedily

6.034 - Spring 03 • 11

Propositional Logic

- Next sections use the language of propositional logic to specify hypotheses in the space of vectors of binary variables
 - "A" is a variable that can take on values "true" and "false" or 1 and 0
 - $A \wedge B$ means that variable A **and** variable B both have to be true (1)
 - $A \vee B$ means that either variable A **or** variable B (or both) have to be true (1)
 - $\neg A$ means that variable A has to be false (0)

6.034 - Spring 03 • 12

Slide 4.2.12

In the following sections, we speak about a set of problems in which the examples are described using vectors of binary variables. In such a space, it's convenient to characterize the hypotheses using propositional logic. We will talk about logic a lot in the last part of the course, but we just need to introduce some notation here.

We'll use single letters to stand for variables that can be true or false (often we'll use 1 for "true" and 0 for "false").

A "caret" B says that A and B both have to be true.

A "vee" B says that either A or B (or both) have to be true.

This thing that's sort of like a bent minus sign before a variable means that it has to be false.

Slide 4.2.13

Let's start with a very simple problem, in which all of the input features are Boolean (we'll represent them with 0's and 1's) and the desired output is also Boolean.

Learning Conjunctions

- Boolean features and output

6.034 - Spring 03 • 13

Learning Conjunctions

- Boolean features and output
- H = conjunctions of features

6.034 - Spring 03 • 14

Slide 4.2.14

Our hypothesis class will be conjunctions of the input features.

Slide 4.2.15

Here's an example data set. It is described using 4 features, which we'll call $f_1, f_2, f_3,$ and f_4 .

Learning Conjunctions

- Boolean features and output
- $H =$ conjunctions of features

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

6.034 - Spring 03 • 15

Learning Conjunctions

- Boolean features and output
- $H =$ conjunctions of features

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

$h = f_1 \wedge f_3$

6.034 - Spring 03 • 16

Slide 4.2.16

So, to understand the hypothesis space let's consider the hypothesis $f_1 \wedge f_3$. It would be true on examples 2 and 3, and false on all the rest.

Slide 4.2.17

We will measure the error of our hypothesis as the number of examples it gets wrong. It marks one negative example as positive, and two positives as negative. So, the error of the hypothesis $f_1 \wedge f_3$ on this data set would be 3.

Learning Conjunctions

- Boolean features and output
- $H =$ conjunctions of features

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

$h = f_1 \wedge f_3$

$E(h, D) = 3$

6.034 - Spring 03 • 17

Learning Conjunctions

- Boolean features and output
- $H =$ conjunctions of features

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

$h = f_1 \wedge f_3$

$E(h, D) = 3$

$C(h) = 2$

- Set alpha so we're looking for smallest h with 0 error

6.034 - Spring 03 • 18

Slide 4.2.18

Finally, we'll measure the complexity of our hypothesis by the number of features mentioned in the conjunction. So the hypothesis $f_1 \wedge f_3$ has complexity 2.

Furthermore, let's assume that alpha is set to be very small, so that our primary goal is to minimize error, but, errors being equal, we'd like to have the smallest conjunction.

Slide 4.2.19

There's now an obvious way to proceed. We could do a general-purpose search in the space of hypotheses, looking for the one that minimizes the cost function. In this case, that might work out okay, since the problem is very small. But in general, we'll want to work in domains with many more features and much more complex hypothesis spaces, making general-purpose search infeasible.

Algorithm

- Could search in hypothesis space using tools we've already studied



6.034 - Spring 03 • 19

Algorithm

- Could search in hypothesis space using tools we've already studied
- Instead, be greedy!



6.034 - Spring 03 • 20

Slide 4.2.20

Instead, we'll be greedy! (When I talk about this and other algorithms being greedy, I'll mean something slightly different than what we meant when we talked about "greedy search". That was a particular instance of a more general idea of greedy algorithm. In greedy algorithms, in general, we build up a solution to a complex problem by adding the piece to our solution that looks like it will help the most, based on a partial solution we already have. This will not, typically, result in an optimal solution, but it usually works out reasonably well, and is the only plausible option in many cases (because trying out all possible solutions would be much too expensive)).

Slide 4.2.21

We'll start out with our hypothesis set to True (that's the empty conjunction). Usually, it will make some errors. Our goal will be to add as few elements to the conjunction as necessary to make no errors.

Algorithm

- Could search in hypothesis space using tools we've already studied
- Instead, be greedy!
- Start with $h = \text{True}$



6.034 - Spring 03 • 21

Algorithm

- Could search in hypothesis space using tools we've already studied
- Instead, be greedy!
- Start with $h = \text{True}$
- All errors are on negative examples
- On each step, add conjunct that rules out most new negatives (without excluding positives)



6.034 - Spring 03 • 22

Slide 4.2.22

Notice that, because we've started with the hypothesis equal to True, all of our errors are on negative examples. So, one greedy strategy would be to add the feature into our conjunction that rules out as many negative examples as possible without ruling out any positive examples.

Slide 4.2.23

Here is pseudo-code for our algorithm. We start with N equal to all the negative examples and the hypothesis equal to true. Then we loop, adding conjuncts that rule out negative examples, until N is empty.

Pseudo-Code

```

N = negative examples in D
h = True
Loop until N is empty

```

6.034 - Spring 03 • 23

Pseudo-Code

```

N = negative examples in D
h = True
Loop until N is empty
  For every feature j that does not have value 0 on
  any positive examples
    nj := number of examples in N
    for which fj = 0
    j* := j for which nj is maximized
    h := h ^ fj*
    N := N - examples in N for which fj* = 0
  If no such feature found, fail

```

6.034 - Spring 03 • 24

Slide 4.2.24

Inside the loop, we consider every feature that would not rule out any positive examples. We pick the one that rules out the most negative examples (here's where our greed kicks in), conjoin it to h , and remove the examples that it excludes from N .

Slide 4.2.25

Let's simulate the algorithm on our data. We start with N equal to x^1 , x^3 , and x^5 , which are the negative examples. And h starts as True. We'll cover all the examples that the hypothesis makes true pink.

Simulation

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- $N = \{x^1, x^3, x^5\}$, $h = \text{True}$

6.034 - Spring 03 • 25

Simulation

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- $N = \{x^1, x^3, x^5\}$, $h = \text{True}$
- $n_3 = 1$, $n_4 = 2$

6.034 - Spring 03 • 26

Slide 4.2.26

Now, we consider all the the features that would not exclude any positive examples. Those are features f_3 and f_4 . f_3 would exclude 1 negative example; f_4 would exclude 2. So we pick f_4 .

Slide 4.2.27

Now we remove the examples from N that are ruled out by f_4 and add f_4 to h .

Simulation

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- $N = \{x^1, x^3, x^5\}$, $h = \text{True}$
- $n_3 = 1, n_4 = 2$
- $N = \{x^5\}$, $h = f_4$

6.034 - Spring 03 • 27

Simulation

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- $N = \{x^1, x^3, x^5\}$, $h = \text{True}$
- $n_3 = 1, n_4 = 2$
- $N = \{x^5\}$, $h = f_4$
- $n_3 = 1, n_4 = 0$

6.034 - Spring 03 • 28

Slide 4.2.28

Now, based on the new N , $n_3 = 1$ and $n_4 = 0$. So we pick f_3 .

Slide 4.2.29

Because f_3 rules out the last remaining negative example, we're done!

Simulation

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	0
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- $N = \{x^1, x^3, x^5\}$, $h = \text{True}$
- $n_3 = 1, n_4 = 2$
- $N = \{x^5\}$, $h = f_4$
- $n_3 = 1, n_4 = 0$
- $N = \{\}$, $h = f_4 \wedge f_3$

6.034 - Spring 03 • 29

A Harder Problem

- We made one negative into a positive

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

6.034 - Spring 03 • 30

Slide 4.2.30

What if we have this data set? In which we just made one negative example into a positive?

Slide 4.2.31

If we simulate the algorithm, we will arrive at the situation in which we still have elements in N , but there are no features left that do not exclude positive examples. So we're stuck.

A Harder Problem

- We made one negative into a positive
- Only usable feature is f_3
- Can't add any more features to h
- We're stuck

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

←

6.034 - Spring 03 • 31

A Harder Problem

- We made one negative into a positive
- Only usable feature is f_3
- Can't add any more features to h
- We're stuck

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

←

- Best we can do when H is conjunctions
- Live with error or change H

6.034 - Spring 03 • 32

Slide 4.2.32

What's going on? The problem is that this hypothesis class simply can't represent the data we have with no errors. So now we have a choice: we can accept the hypothesis we've got, or we can increase the size of the hypothesis class. In practice, which one you should do often depends on knowledge of the domain. But the fact is that pure conjunctions is a very limiting hypothesis class. So let's try something a little fancier.

Slide 4.2.33

You can kind of think of a conjunction as narrowing down on a small part of the space. And if all the positive examples can be corralled into one group this way, everything is fine. But for some concepts, it might be necessary to have multiple groups.

This corresponds to a form of Boolean hypothesis that's fairly easy to think about: disjunctive normal form. Disjunctive normal form is kind of like CNF, only this time it has the form $(A \text{ and } B \text{ and } C) \text{ or } (D \text{ and } E \text{ and } F)$

As a hypothesis class, it's like finding multiple groups of positive examples within the negative space.

Disjunctive Normal Form

- Like the opposite of conjunctive normal form (but, for now, without negations of the atoms)
 $(A \wedge B \wedge C) \vee (D \wedge E) \vee F$
- Think of each disjunct as narrowing in on a subset of the positive examples

6.034 - Spring 03 • 33

Disjunctive Normal Form

- Like the opposite of conjunctive normal form (but, for now, without negations of the atoms)
 $(A \wedge B \wedge C) \vee (D \wedge E) \vee F$
- Think of each disjunct as narrowing in on a subset of the positive examples

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

6.034 - Spring 03 • 34

Slide 4.2.34

So, let's look at our harder data set and see if we can find a good hypothesis in DNF.

Slide 4.2.35

It's pretty to see that one way to describe it is $f_3 \wedge f_4 \vee f_1 \wedge f_2$. Now let's look at an algorithm for finding it.

Disjunctive Normal Form

- Like the opposite of conjunctive normal form (but, for now, without negations of the atoms)
 $(A \wedge B \wedge C) \vee (D \wedge A) \vee E$
- Think of each disjunct as narrowing in on a subset of the positive examples

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

$$(f_3 \wedge f_4) \vee (f_1 \wedge f_2)$$

6.034 - Spring 03 • 35

Learning DNF

- Let H be DNF expressions
- $C(h)$: number of mentions of features

$$C((f_3 \wedge f_4) \vee (f_1 \wedge f_2)) = 4$$

6.034 - Spring 03 • 36

Slide 4.2.36

We'll let our hypothesis class be expressions in disjunctive normal form. What should our measure of complexity be? One reasonable choice is the total number of mentions of features (so $f_3 \wedge f_4 \vee f_1 \wedge f_2$ would have complexity 4).

Learning DNF

- Let H be DNF expressions
- $C(h)$: number of mentions of features

$$C((f_3 \wedge f_4) \vee (f_1 \wedge f_2)) = 4$$

- Really hard to search this space, so be greedy again!

6.034 - Spring 03 • 37

Slide 4.2.37

Now, searching in this space would be much harder than searching in the space of conjunctions. So we'll use another greedy algorithm.

Learning DNF

- Let H be DNF expressions
- $C(h)$: number of mentions of features

$$C((f_3 \wedge f_4) \vee (f_1 \wedge f_2)) = 4$$

- Really hard to search this space, so be greedy again!
- A conjunction **covers** an example if all of the features mentioned in the conjunction are true in the example

6.034 - Spring 03 • 38

Slide 4.2.38

We'll say a conjunction **covers** an example if all of the features mentioned in the conjunction are true in the example.

Slide 4.2.39

Here's pseudocode for the algorithm. It has two main loops. The inner loop constructs a conjunction (much like our previous algorithm). The outer loop constructs multiple conjunctions and disjoins them.

The idea is that each disjunct will *cover* or account for some subset of the positive examples. So in the outer loop, we make a conjunction that includes some positive examples and no negative examples, and add it to our hypothesis. We keep doing that until no more positive examples remain to be covered.

Algorithm

```

P = set of all positive examples
h = False
Loop until P is empty
  r = True
  N = set of all negative examples
  Loop until N is empty
    If all features are in r, fail
    Else, select a feature  $f_j$  to add to r
       $r = r \wedge f_j$ 
       $N = N - \text{examples in } n \text{ for which } f_j = 0$ 
  h := h v r
  Covered := examples in P covered by r
  If Covered is empty, fail
  Else P := P - Covered
end

```

6.034 - Spring 03 • 39

Choosing a Feature

Heuristic: $v_j = \frac{n_j^+}{\max(n_j^-, 0.001)}$

6.034 - Spring 03 • 40

Slide 4.2.40

We also haven't said how to pick which feature to add to r. It's trickier here, because we can't restrict our attention to features that allow us to cover all possible examples. Here's one heuristic for selecting which feature to add next.

Slide 4.2.41

Let v_j be n_j^+ / n_j^- , where n_j^+ is the total number of so-far uncovered positive examples that are covered by rule $r \wedge f_j$ and n_j^- is the total number of thus-far not-ruled-out negative examples that are covered by rule $r \wedge f_j$. The intuition here is that we'd like to add features that cover a lot of positive examples and exclude a lot of negative examples, because that's our overall goal.

There's one additional problem about what to do when n^- is 0. In that case, this is a really good feature, because it covers positives and doesn't admit any negatives. We'll prefer features with zero in the denominator over all others; if we have multiple such features, we'll prefer ones with bigger numerator. To make this fit easily into our framework, if the denominator is zero, we just return as a score 1 million times the numerator.

Choosing a Feature

Heuristic: $v_j = \frac{n_j^+}{\max(n_j^-, 0.001)}$

$n_j^+ = \# \text{ not yet covered positive examples covered by } r \wedge f_j$

$n_j^- = \# \text{ not yet ruled out negative examples covered by } r \wedge f_j$

6.034 - Spring 03 • 41

Choosing a Feature

Heuristic: $v_j = \frac{n_j^+}{\max(n_j^-, 0.001)}$

$n_j^+ = \# \text{ not yet covered positive examples covered by } r \wedge f_j$

$n_j^- = \# \text{ not yet ruled out negative examples covered by } r \wedge f_j$

Choose feature with largest value of V_j

6.034 - Spring 03 • 42

Slide 4.2.42

Then we can replace this line in the code with one that says: Select the feature f_j with the highest value of v_j to add to r. And now we have a whole algorithm.

Slide 4.2.43

Let's simulate it on our data set. Here's a trace of what happens. It's important to remember that when we compute v_j , we consider only positive and negative examples not covered so far.

We start with $h = \text{false}$, and so our current hypothesis doesn't make any examples true.

Simulation

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- $h = \text{False}$, $P = \{x^2, x^3, x^4, x^6\}$

6.034 - Spring 03 • 43

Simulation

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- $h = \text{False}$, $P = \{x^2, x^3, x^4, x^6\}$
- $r = \text{True}$, $N = \{x^1, x^5\}$
- $v_1=2/1$, $v_2=2/1$, $v_3=4/1$, $v_4=3/1$
- $r = f_3$, $N = \{x^1\}$
- $v_1=2/0$, $v_2=2/1$, $v_4=3/0$
- $r = f_3 \wedge f_4$, $N = \{\}$
- $h = f_3 \wedge f_4$, $P = \{x^3\}$

6.034 - Spring 03 • 44

Slide 4.2.44

After the first iteration of the inner loop, our hypothesis covers the examples shown in pink. There's still another positive example to get.

Slide 4.2.45

After another iteration, we add a new rule, which covers the example shown in blue. And we're done.

Simulation

f_1	f_2	f_3	f_4	y
0	1	1	0	0
1	0	1	1	1
1	1	1	0	1
0	0	1	1	1
1	0	0	1	0
0	1	1	1	1

- $h = \text{False}$, $P = \{x^2, x^3, x^4, x^6\}$
- $r = \text{True}$, $N = \{x^1, x^5\}$
- $v_1=2/1$, $v_2=2/1$, $v_3=4/1$, $v_4=3/1$
- $r = f_3$, $N = \{x^1\}$
- $v_1=2/0$, $v_2=2/1$, $v_4=3/0$
- $r = f_3 \wedge f_4$, $N = \{\}$
- $h = f_3 \wedge f_4$, $P = \{x^3\}$
- $r = \text{True}$, $N = \{x^1, x^5\}$
- $v_1=1/1$, $v_2=1/1$, $v_3=1/1$, $v_4=0/1$
- $r = f_1$, $N = \{x^1\}$
- $v_2=1/0$, $v_3=1/0$, $v_4=0/1$
- $r = f_1 \wedge f_2$, $N = \{\}$
- $h = (f_3 \wedge f_4) \vee (f_1 \wedge f_2)$, $P = \{\}$

6.034 - Spring 03 • 45

Slide 4.3.1

Once our learning algorithm has generated a hypothesis, we'd like to be able to estimate how well it is going to work out in the "real world". What is the real world? Well, we can't expect the hypothesis to perform well in every possible world; we *can* expect it to perform well in the world that generated the data that was used to train it.

How Well Does it Work?

- We'd like to know how well our h will perform on new data (drawn from the same distribution as the training data)

6.034 - Spring 03 • 1

How Well Does it Work?

- We'd like to know how well our h will perform on new data (drawn from the same distribution as the training data)
- Performance of hypothesis on the training set is not indicative of its performance on new data

6.034 - Spring 03 • 2

Slide 4.3.2

Just by looking at the training data, we can't tell how well the hypothesis will work on unseen data. We have constructed the hypothesis so that it has zero error on the training data. But unless we're wildly optimistic, we can't expect to do so well on new data.

How Well Does it Work?

- We'd like to know how well our h will perform on new data (drawn from the same distribution as the training data)
- Performance of hypothesis on the training set is not indicative of its performance on new data
- Save some data as a **test set**; performance on h is a reasonable estimate of performance on new data

6.034 - Spring 03 • 3

Slide 4.3.3

The only way to know how well the hypothesis will do on new data is to save some of the original data in a set called the "validation set" or the "test set". The data in the validation set are not used during training. Instead, they are "held out" until the hypothesis has been derived. Then, we can test the hypothesis on the validation set and see what percentage of the cases it gets wrong. This is called "test set error".

Cross Validation

To evaluate performance of an algorithm as a whole (rather than a particular hypothesis):

Slide 4.3.4

Sometimes you'd like to know how well a particular learning algorithm is working, rather than evaluating a single hypothesis it may have produced. Running the algorithm just once and evaluating its hypothesis only gives you one sample of its behavior, and it might be hard to tell how well it really works from just one sample.

If you have **lots** of data, you can divide it up into batches, and use half of your individual batches to train new hypotheses and the other half of them to evaluate the hypotheses. But you rarely have enough data to be able to do this.

6.034 - Spring 03 • 4

Slide 4.3.5

One solution is called cross-validation. You divide your data up into some number (let's say 10 for now) of chunks. You start by feeding chunks 1 through 9 to your learner and evaluating the resulting hypothesis using chunk 10.

Now, you feed all but chunk 9 into the learner and evaluate the resulting hypotheses on chunk 9. And so on.

Cross Validation

To evaluate performance of an algorithm as a whole (rather than a particular hypothesis):

- Divide data into k subsets
- k different times
 - train on k-1 of the subsets
 - test on the held-out subset
- return average test score over all k tests

6.034 - Spring 03 • 5

Cross Validation

To evaluate performance of an algorithm as a whole (rather than a particular hypothesis):

- Divide data into k subsets
- k different times
 - train on k-1 of the subsets
 - test on the held-out subset
- return average test score over all k tests

Useful for deciding which class of algorithms to use on a particular data set.

6.034 - Spring 03 • 6

Slide 4.3.6

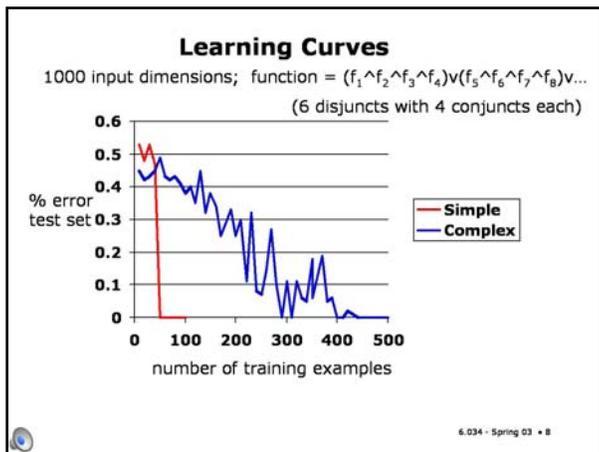
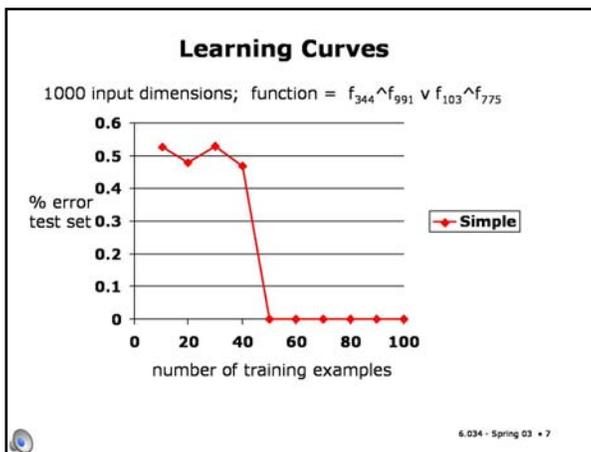
At the end of this process, you will have run the learning algorithm 10 times, getting 10 different hypotheses, and tested each of the hypotheses on a different test set. You can now average the error results from each of these trials to get an aggregate picture of how well your learning algorithm performs. (For those of you who might be statistically sophisticated, you have to be careful with these statistics because the samples are not independent.) Remember that this is a method for comparing algorithms on a data set, not for evaluating the quality of a particular hypothesis.

Slide 4.3.7

One interesting thing to see is how well the learning process works as a function of how much data is available to it. I made a learning curve by secretly picking a target function to generate the data, drawing input points uniformly at random and generating their associated y using my secret function. First I generated 10 points and trained a hypothesis on them, then evaluated it. Then I did it for 20 points, and so on.

This is the learning curve for a fairly simply function: $f_{344} \vee f_{991} \vee f_{103} \wedge f_{775}$, in a domain with 1000 features. The x axis is the amount of training data. The y axis is the number of errors made by the resulting hypothesis, on a test set of data that is drawn uniformly at random, as a percentage of the total size of the test set.

It takes suprisingly few samples before we find the correct hypothesis. With 1000 features, there are 2^{1000} possible examples. We are getting the correct hypothesis after seeing a tiny fraction of the whole space.



Slide 4.3.8

Now, let's look at a much more complex target function. It's $(f_1 \wedge f_2 \wedge f_3 \wedge f_4) \vee (f_5 \wedge f_6 \wedge f_7 \wedge f_8) \vee (f_9 \wedge f_{10} \wedge f_{11} \wedge f_{12}) \vee (f_{13} \wedge f_{14} \wedge f_{15} \wedge f_{16}) \vee (f_{17} \wedge f_{18} \wedge f_{19} \wedge f_{20}) \vee (f_{21} \wedge f_{22} \wedge f_{23} \wedge f_{24})$. It takes a lot more data to learn. Why is that?

Slide 4.3.9

First, it's important to remember that, at every point in the graph, the learning algorithm has found a hypothesis that gets all of the training data correct.

Simple Gifts

- At every point in the graph, we've found an h that gets the whole training set correct (call it apparently correct)



6.034 - Spring 03 • 9

Simple Gifts

- At every point in the graph, we've found an h that gets the whole training set correct (call it apparently correct)
- When input dimensionality is high and size of training set is small, there are lots of apparently correct hypotheses
- Our algorithm tries to return (but doesn't always) the simplest apparently correct hypothesis



6.034 - Spring 03 • 10

Slide 4.3.10

When we have just a little bit of data, there are many possible functions that agree with it all. How does our algorithm choose which answer to give? It tries to give a simple one. So, we would say that the algorithm has a "bias" in favor of simple hypotheses.

Slide 4.3.11

So, when the target hypothesis is simple, we discover it quickly (the other simple hypotheses are ruled out with just a little bit of data, because there are so few of them).

Simple Gifts

- At every point in the graph, we've found an h that gets the whole training set correct (call it apparently correct)
- When input dimensionality is high and size of training set is small, there are lots of apparently correct hypotheses
- Our algorithm tries to return (but doesn't always) the simplest apparently correct hypothesis
- So, when the target hypothesis is simple, we discover it quickly (the other simple hypotheses are ruled out quickly because there are so few)



6.034 - Spring 03 • 11

Simple Gifts

- At every point in the graph, we've found an h that gets the whole training set correct (call it apparently correct)
- When input dimensionality is high and size of training set is small, there are lots of apparently correct hypotheses
- Our algorithm tries to return (but doesn't always) the simplest apparently correct hypothesis
- So, when the target hypothesis is simple, we discover it quickly (the other simple hypotheses are ruled out quickly because there are so few)
- When the target hypothesis is complex, it's hard to rule out all of the (many) other competitors, so it takes more data to learn



6.034 - Spring 03 • 12

Slide 4.3.12

If we thought our problems were going to have complex answers, then we might think about biasing our algorithm to prefer more complex answers. But then we start to see the wisdom of Ockham. There are **lots** more complex hypotheses than there are simple ones. So there will be huge numbers of complex hypotheses that agree with the data and no basis to choose among them, unless we know something about the domain that generated our data (maybe some features are thought, in advance, to be more likely to have an influence on the outcome, for example).

Slide 4.3.13

There is, in fact, a theorem in machine learning called the "No Free Lunch" theorem. It basically states that, unless you know something about the process that generated your data, any hypothesis that agrees with all the data you've seen so far is as good a guess as any other one.

The picture will get a bit more complicated when we look at data that are noisy.

No Free Lunch

- Unless you know something about the distribution of problems your learning algorithm will encounter, ***any hypothesis that agrees with all your data is as good as any other.***
- You can't learn anything unless you already know something.

6.034 - Spring 03 • 13

Noisy Data

- Have to accept non-zero error on training data

6.034 - Spring 03 • 14

Slide 4.3.14

It is actually very rare to encounter a machine learning problem in which there is a deterministic function that accounts for y 's dependence on x in all cases. It is much more typical to imagine that the process that generates the x 's and y 's is probabilistic, possibly corrupting measurements of the features, or simply sometimes making different decisions in the same circumstances.

Slide 4.3.15

We can easily handle noise in the framework we have introduced. We left the door open for it, so to speak, in our formulation of the problem we were trying to solve. We want to find a hypothesis h that minimizes error plus complexity. When there's noise, we'll have to accept that our hypothesis will probably have non-zero error on the training set; and we'll have to be very careful about how we weight these two criteria, as we will see.

Noisy Data

- Have to accept non-zero error on training data
- Weaken DNF learning algorithm
 - Don't require the hypothesis to cover all positive examples
 - Don't require each rule to exclude all negative examples

6.034 - Spring 03 • 15

Pseudo Code: Noisy DNF Learning

```

P := the set of positive examples
h := False
np := epsilon * number of examples in P
nn := epsilon * number of examples in N
Loop until P has fewer than np elements
  r = True
  N = the set of negative examples
  Repeat until N has fewer than nn elements
    Select a feature  $f_j$  to add to r
     $r := r \wedge f_j$ 
     $N := N - \text{examples in } N \text{ for which } f_j = 0$ 
  h := h  $\vee$  r
P := P - elements in P covered by r

```

allow epsilon
percentage error

6.034 - Spring 03 • 16

Slide 4.3.16

We can use our DNF learning algorithm almost unchanged. But, instead of requiring each rule to exclude **all** negative examples, and requiring the whole hypothesis to cover **all** positive examples, we will only require them to cover some percentage of them. Here is the pseudocode, with the changes highlighted in blue.

Slide 4.3.17

We also have to add further conditions for stopping both loops. If there is no feature that can be added that reduces the size of N , then the inner loop must terminate. If the inner loop cannot generate a rule that reduces the size of P , then the outer loop must terminate and fail.

Pseudo Code: Noisy DNF Learning

```

P := the set of positive examples
h := False
np := epsilon * number of examples in P
nn := epsilon * number of examples in N
Loop until P has fewer than np elements or not progressing
  r = True
  N = the set of negative examples
  Repeat until N has fewer than nn elements or not progress
    Select a feature  $f_j$  to add to r
    r := r  $\wedge$   $f_j$ 
    N := N - examples in N for which  $f_j = 0$ 
  h := h  $\vee$  r
  P := P - elements in P covered by r

```

allow epsilon
percentage error

Handle failure to progress

6.034 - Spring 03 • 17

Epsilon is our Delta

- Parameter epsilon is the percentage error allowed

6.034 - Spring 03 • 18

Slide 4.3.18

Now we have a new parameter, epsilon, which is the percentage of examples we're allowed to get wrong. (Well, we'll actually get more wrong, because each disjunct is allowed to get epsilon percent of the negatives wrong).

Slide 4.3.19

Although it is not exactly equivalent to manipulating the alpha parameter in our ideal criterion, it is a way to trade off accuracy for complexity. The higher the epsilon, the simpler and more error-prone our hypothesis.

Epsilon is our Delta

- Parameter epsilon is the percentage error allowed
- The higher epsilon, the simpler and more error-prone the hypothesis

6.034 - Spring 03 • 19

Epsilon is our Delta

- Parameter epsilon is the percentage error allowed
- The higher epsilon, the simpler and more error-prone the hypothesis
- If epsilon is small and the data is noisy, the algorithm may fail to find an acceptable hypothesis

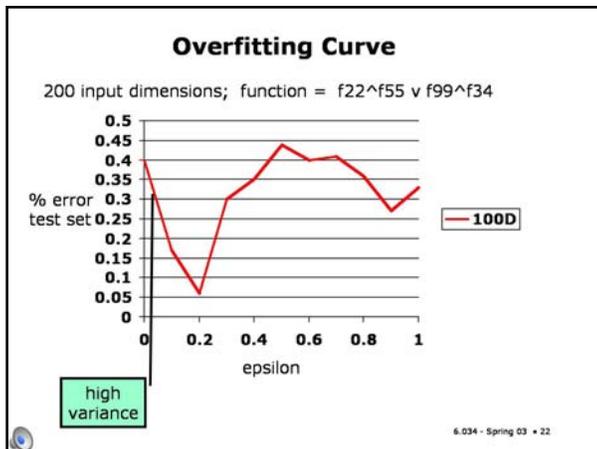
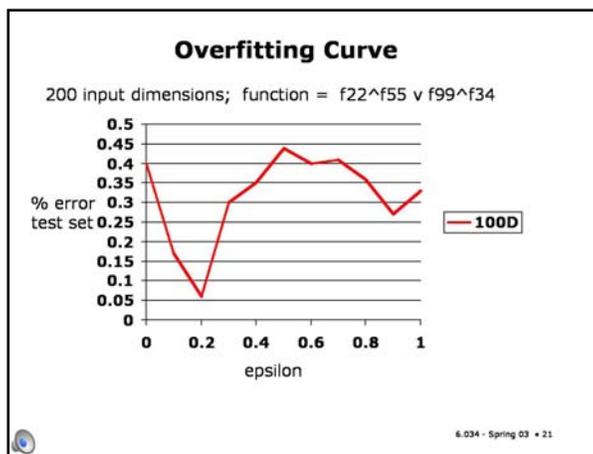
6.034 - Spring 03 • 20

Slide 4.3.20

If epsilon is small and the data is noisy, the algorithm may fail to find an acceptable hypothesis.

Slide 4.3.21

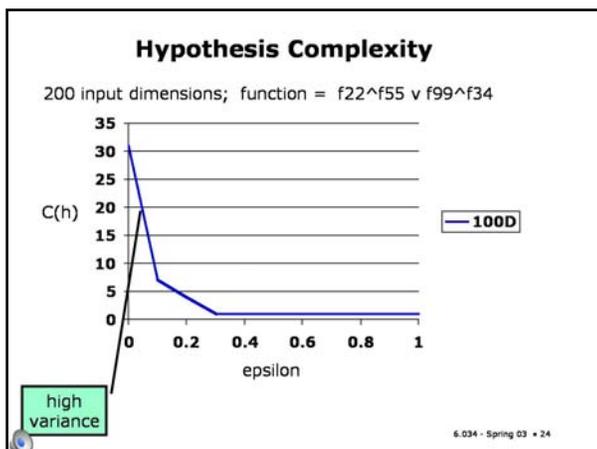
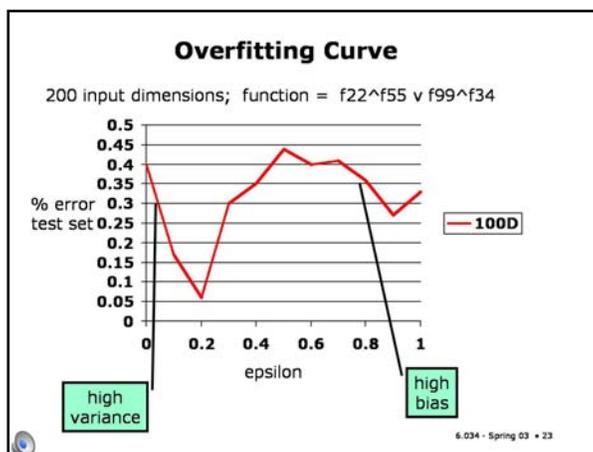
Let's see what happens in a moderately noisy domain. This curve shows the problem of learning our easy DNF function in a domain with 100 features, on a training set of size 100. The data has 10 percent noise, which means that, in 10 percent of the cases, we expect the output set to be the opposite of the one specified by the target function. The x axis is epsilon, ranging from 0 to 1. At setting 0, it is the same as our original algorithm. At setting 1, it's willing to make an arbitrary number of errors, and will always return the hypothesis False. The y axis is percentage error on a freshly drawn set of testing data. Note that, because there is 10 percent error in the data (training as well as testing), our very best hope is to generate a hypothesis that has 10 percent error on average on testing data.

**Slide 4.3.22**

We get the best performance for a value of epsilon that's between 0 and 1. What's the matter with setting it to some value near 0? We run into the problem of **overfitting**. In overfitting, we work very hard to reduce the error of our hypothesis on the training set. But we may have spent a lot of effort modeling the noise in the data, and perform poorly on the test set. That's what is happening here. It is said, in this case, that our algorithm has high **variance**. Another way to think about the problem is to see that if we get another noisy data set generated by the same target function, we are likely to get a wildly different answer. By turning epsilon up a bit, we generate simpler hypotheses that are not so susceptible to variations in the data set, and so can get better generalization performance.

Slide 4.3.23

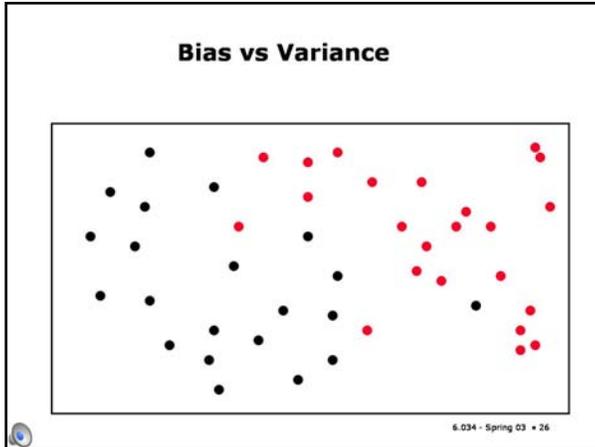
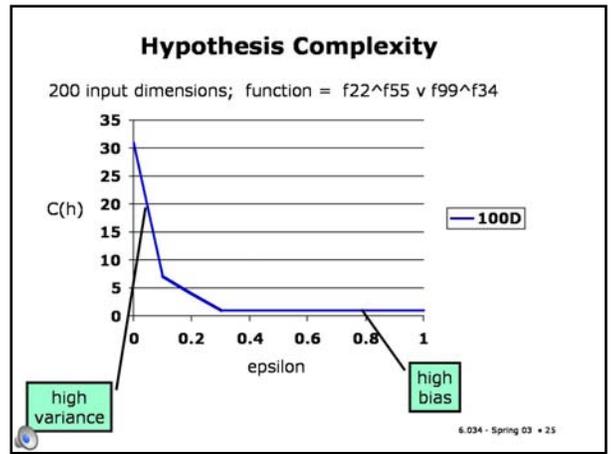
Of course, turning epsilon up too high will keep us from building a hypothesis of sufficient complexity. Then we'd do poorly because we are unable to even represent the right answer. It is said, in this case, that our algorithm has high **bias**.

**Slide 4.3.24**

It's also instructive to see how the complexity of the resulting hypothesis varies as a function of epsilon. When epsilon is 0, we are asking for a hypothesis with zero error on the training set. We are able to find one, but it's very complex (31 literals). This is clearly a high variance situation; that hypothesis is completely influenced by the particular training set and would change radically on a newly drawn training set (and therefore, have high error on a testing set, as we saw on the previous slide).

Slide 4.3.25

As epsilon increases, we rapidly go down to a complexity of 1, which is incapable of representing the target hypothesis. The low-point in the error curve was at epsilon = 0.2, which has a complexity of 4 here. And it's no coincidence that the target concept also has a complexity of 4.



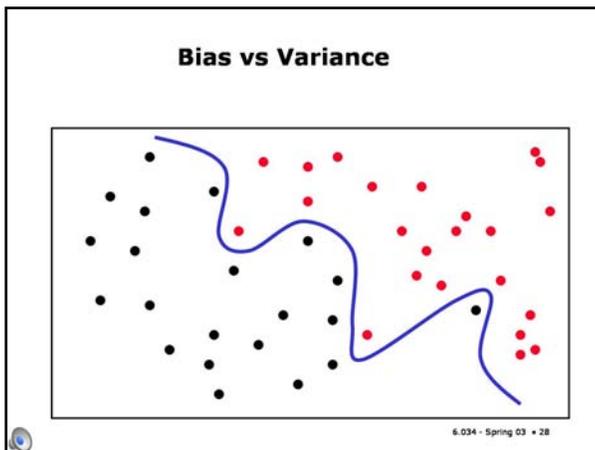
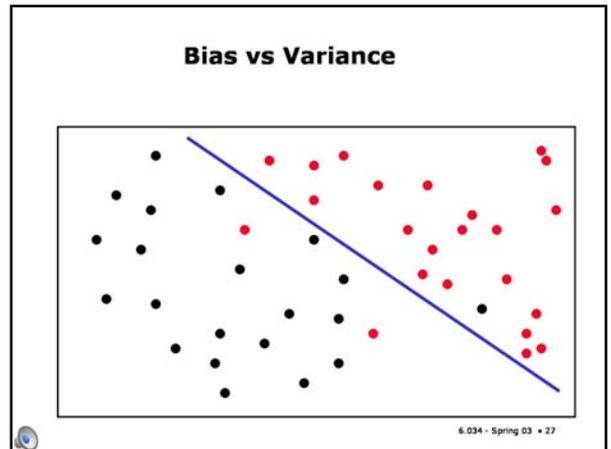
Slide 4.3.26

This problem is called the bias/variance trade-off. Because this is such an important idea in machine learning, let's look at it one more time in the context of our old pictures of dots on the plane.

Slide 4.3.27

Here's some noisy data.

If we separate it with a line, we have low complexity, but lots of error.

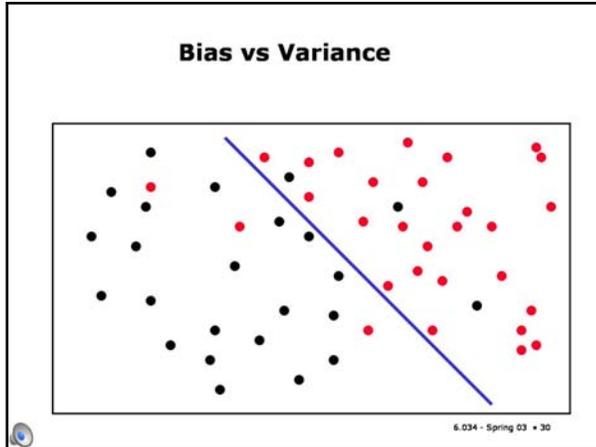
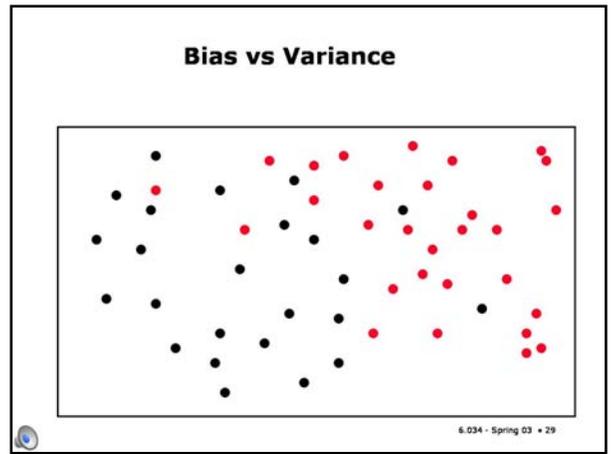


Slide 4.3.28

If we separate it with a squiggly snake, we can reduce the error but add a lot of complexity.

Slide 4.3.29

So, what, exactly, is so bad about complexity? Well, let's draw a bunch more points from the same underlying distribution and add them to our data set.



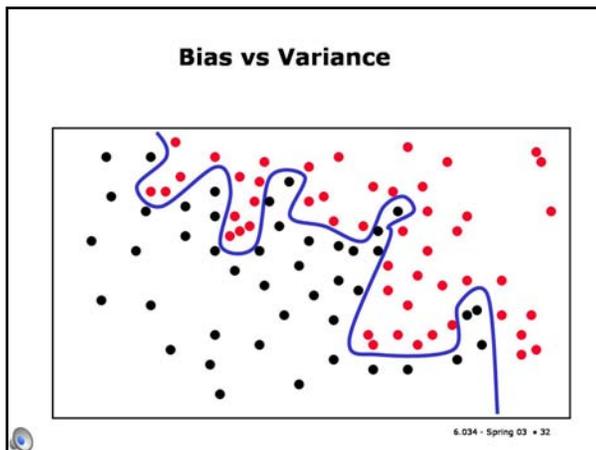
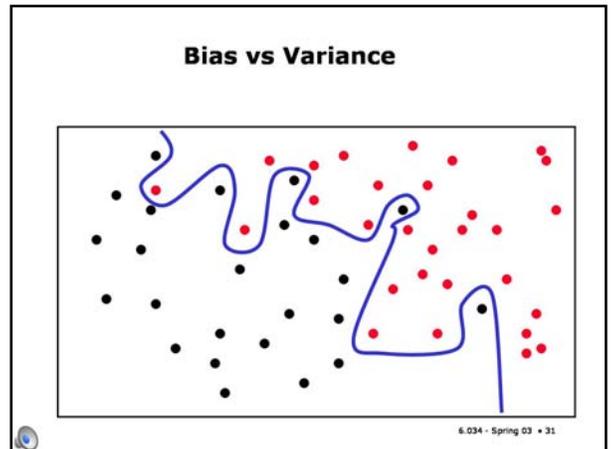
Slide 4.3.30

We wouldn't need to change our line hypothesis very much to do our best to accommodate them.

Slide 4.3.31

But we'd have to change the snake hypothesis radically. This is what it means to have high variance.

The line, although it was making mistakes on the original data set, performs better on unseen data.



Slide 4.3.32

In practice, it is common to allow hypotheses of increased complexity as the amount of available training data increases. So, in this situation, we might feel justified in choosing a moderately complex hypothesis.

Slide 4.3.33

So, given a data set, how should we choose epsilon? This is where cross-validation can come in handy. You select a number of different values of epsilon, and use cross-validation to estimate the error when using that epsilon. Then, you pick the epsilon value with the least estimated error, and train your learning algorithm with all the available data using that value of epsilon. The resulting hypothesis is probably the best you can do (given what you know).

Picking Epsilon

- Pick epsilon using cross validation
- Try multiple different values of epsilon
- See which gives the lowest cross-validation error
- Use that value of epsilon to learn a hypothesis from the whole training set
- Return that hypothesis as your best answer

Domains

Slide 4.3.34

So, is this algorithm actually good for anything? Yes. There are lots of domains that can be encoded as vectors of binary features with a binary classification. Here are some examples.

Slide 4.3.35

Congressional voting: given a congressperson's voting record, where the features are the individual's votes on various bills, can you predict whether they are a republican or democrat?

Domains

- Congressional voting: given a congressperson's voting record (list of 1s and 0s), predict party

Domains

- Congressional voting: given a congressperson's voting record (list of 1s and 0s), predict party
- Gene splice: predict the beginning of a coding section of the genome; input is vector of elements chosen from the set {ACGT}; encode each element with two bits (or possibly with 4)

Slide 4.3.36

Gene splicing: given a sequence of bases (A, C, G, or T), predict whether the mid-point is a splice junction between a coding segment of DNA and a non-coding segment. In order to apply our algorithms to this problem, we would have to change the representation somewhat. It would be easy to take each base and represent it as two binary values; A is 00, C is 01, G is 10 and T is 11. Even though it seems inefficient, for learning purposes, it is often better to represent discrete elements using a unary code; so A would be 0001, C would be 0010, G would be 0100 and T would be 1000.

Slide 4.3.37

Spam filtering: is this email message spam? Represent a document using a really big feature space, with one feature for each possible word in the dictionary (leaving out some very common ones, like "a" and "the"). The feature for a word has value 1 if that word exists in the document and 0 otherwise.

Domains

- Congressional voting: given a congressperson's voting record (list of 1s and 0s), predict party
- Gene splice: predict the beginning of a coding section of the genome; input is vector of elements chosen from the set {ACGT}; encode each element with one bit (or possibly with 4)
- Spam filtering: encode every message as a vector of features, one per word; a feature is on if that word occurs in the message; predict whether or not the message is spam

6.034 - Spring 03 • 37

Domains

- Congressional voting: given a congressperson's voting record (list of 1s and 0s), predict party
- Gene splice: predict the beginning of a coding section of the genome; input is vector of elements chosen from the set {ACGT}; encode each element with one bit (or possibly with 4)
- Spam filtering: encode every message as a vector of features, one per word; a feature is on if that word occurs in the message; predict whether or not the message is spam
- Marketing: predict whether a person will buy beer based on previous purchases; encode buying habits with a feature for all products, set to 1 if previously purchased

6.034 - Spring 03 • 38

Slide 4.3.38

Marketing: is this grocery-store customer likely to buy beer? Represent a person's buying history using a big feature space, with one feature for each product in the supermarket. The feature for a product has value 1 if the person has ever bought that product and 0 otherwise.

Slide 4.3.39

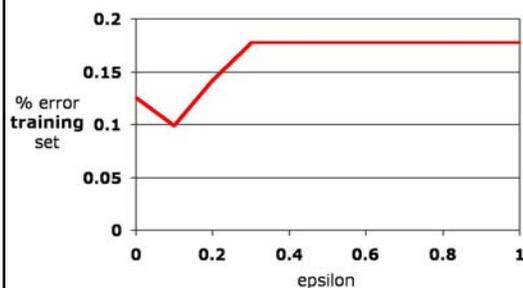
Just for fun, we ran the DNF learning algorithm on the congressional voting data. A lot of the records had missing data (those congress folks are always out in the hallways talking instead of in there voting); we just deleted those records (though in a machine learning class we would study better ways of handling missing data items). There were 232 training examples, each of which had 15 features. The features were the individuals' votes on these bills (unfortunately, we don't know any details about the bills, but the names are evocative).

Congressional Voting

0. handicapped-infants
1. water-project-cost-sharing
2. adoption-of-the-budget-resolution
3. physician-fee-freeze
4. el-salvador-aid
5. religious-groups-in-schools
6. anti-satellite-test-ban
7. aid-to-nicaraguan-contras
8. mx-missile
9. immigration
10. synfuels-corporation-cutback
11. education-spending
12. superfund-right-to-sue
13. crime
14. duty-free-exports
15. export-administration-act-south-africa

232 data points

6.034 - Spring 03 • 39

Congressional Voting: Republican?

6.034 - Spring 03 • 40

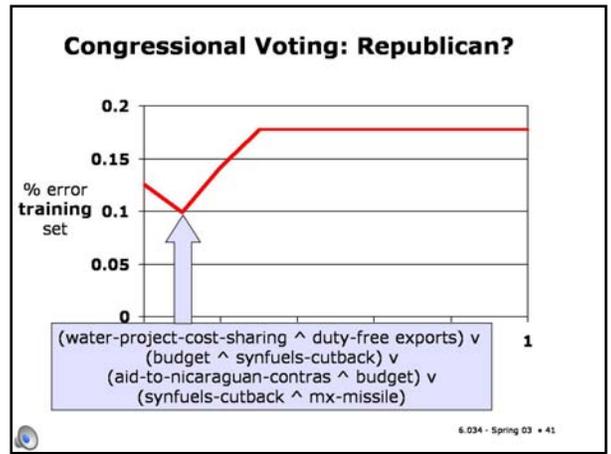
Slide 4.3.40

We are trying to predict whether a person is a Republican, based on their voting history during part of 1984. Here is a plot of the **training set error** as a function of epsilon. What I should really do is cross-validation, but instead I just ran the algorithm on the whole data set and I'm plotting the error on the training set.

Something sort of weird is going on. Training set error should get smaller as epsilon decreases. Why do we have it going up for epsilon equal zero? It's because of the greediness of our algorithm. We're not actually getting the best hypothesis in that case.

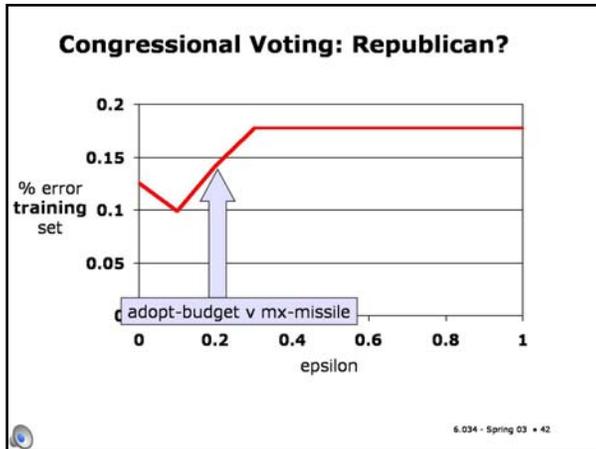
Slide 4.3.41

Here's the hypothesis we get at this point. It's kind of complicated (but mentions a lot of Republican things, like aid to the Nicaraguan contras and the mx-missile).



Slide 4.3.42

Here's a simpler hypothesis (with slightly worse error).



Slide 4.3.43

And here's the best hypothesis of complexity 1 for predicting whether someone is a Republican: did they vote "yes" for the MX missile?

And, just because you're all too young to remember this, a little history. The MX missile, also called the "Peacekeeper" was our last big nuclear missile project. It's 71 feet long, with 10 warheads, and launched from trains. It was politically very contentious. Congress killed it at least twice before it was finally approved.

