CSC2515 Fall 2007
Introduction to Machine Learning

Lecture 1: What is Machine Learning?

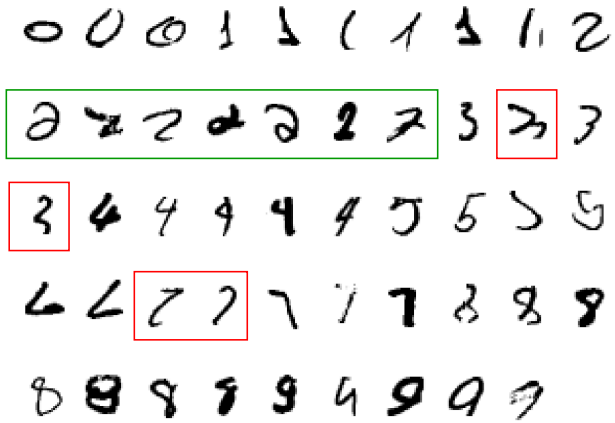
All lecture slides will be available as .ppt, .ps, & .htm at
www.cs.toronto.edu/~hinton

Many of the figures are provided by Chris Bishop
from his textbook: "Pattern Recognition and Machine Learning"

What is Machine Learning?

- It is very hard to write programs that solve problems like recognizing a face.
  - We don't know what program to write because we don't know how our brain does it.
  - Even if we had a good idea about how to do it, the program might be horrendously complicated.
- Instead of writing a program by hand, we collect lots of examples that specify the correct output for a given input.
- A machine learning algorithm then takes these examples and produces a program that does the job.
  - The program produced by the learning algorithm may look very different from a typical hand-written program. It may contain millions of numbers.
  - If we do it right, the program works for new cases as well as the ones we trained it on.

A classic example of a task that requires machine learning: It is very hard to say what makes a 2



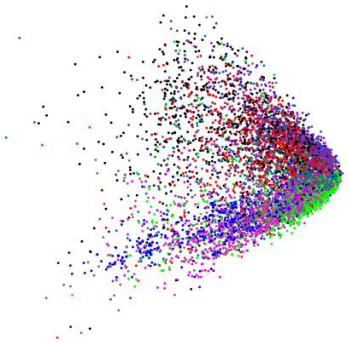
Some more examples of tasks that are best solved by using a learning algorithm

- Recognizing patterns:
  - Facial identities or facial expressions
  - Handwritten or spoken words
  - Medical images
- Generating patterns:
  - Generating images or motion sequences (demo)
- Recognizing anomalies:
  - Unusual sequences of credit card transactions
  - Unusual patterns of sensor readings in a nuclear power plant or unusual sound in your car engine.
- Prediction:
  - Future stock prices or currency exchange rates

Some web-based examples of machine learning

- The web contains a lot of data. Tasks with very big datasets often use machine learning
  - especially if the data is noisy or non-stationary.
- Spam filtering, fraud detection:
  - The enemy adapts so we must adapt too.
- Recommendation systems:
  - Lots of noisy data. Million dollar prize!
- Information retrieval:
  - Find documents or images with similar content.
- Data Visualization:
  - Display a huge database in a revealing way (demo)

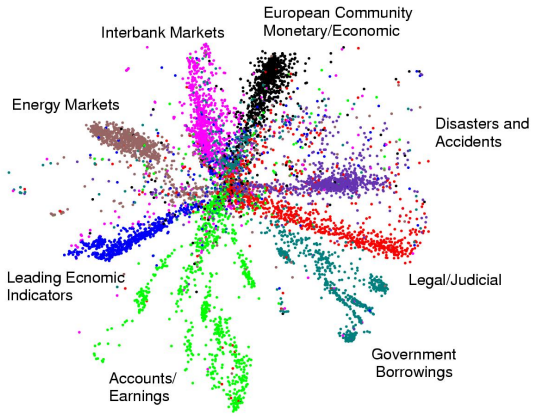
Displaying the structure of a set of documents using Latent Semantic Analysis (a form of PCA)



Each document is converted to a vector of word counts. This vector is then mapped to two coordinates and displayed as a colored dot. The colors represent the hand-labeled classes.

When the documents are laid out in 2-D, the classes are not used. So we can judge how good the algorithm is by seeing if the classes are separated.

Displaying the structure of a set of documents using a deep neural network



Machine Learning & Symbolic AI

- Knowledge Representation works with facts/assertions and develops rules of logical inference. The rules can handle quantifiers. Learning and uncertainty are usually ignored.
- Expert Systems used logical rules or conditional probabilities provided by "experts" for specific domains.
- Graphical Models treat uncertainty properly and allow learning (but they often ignore quantifiers and use a fixed set of variables)
  - Set of logical assertions → values of a subset of the variables and local models of the probabilistic interactions between variables.
  - Logical inference → probability distributions over subsets of the unobserved variables (or individual ones)
  - Learning = refining the local models of the interactions.

Machine Learning & Statistics

- A lot of machine learning is just a rediscovery of things that statisticians already knew. This is often disguised by differences in terminology:
  - Ridge regression = weight-decay
  - Fitting = learning
  - Held-out data = test data
- But the emphasis is very different:
  - A good piece of statistics: Clever proof that a relatively simple estimation procedure is asymptotically unbiased.
  - A good piece of machine learning: Demonstration that a complicated algorithm produces impressive results on a specific task.
- Data-mining: Using very simple machine learning techniques on very large databases because computers are too slow to do anything more interesting with ten billion examples.

A spectrum of machine learning tasks

Statistics----------Artificial Intelligence

- Low-dimensional data (e.g. less than 100 dimensions)
- Lots of noise in the data
- There is not much structure in the data, and what structure there is, can be represented by a fairly simple model.
- The main problem is distinguishing true structure from noise.

- High-dimensional data (e.g. more than 100 dimensions)
- The noise is not sufficient to obscure the structure in the data if we process it right.
- There is a huge amount of structure in the data, but the structure is too complicated to be represented by a simple model.
- The main problem is figuring out a way to represent the complicated structure that allows it to be learned.

Types of learning task

- Supervised learning
  - Learn to predict output when given an input vector
    - Who provides the correct answer?
- Reinforcement learning
  - Learn action to maximize payoff
    - Not much information in a payoff signal
    - Payoff is often delayed
  - Reinforcement learning is an important area that will not be covered in this course.
- Unsupervised learning
  - Create an internal representation of the input e.g. form clusters; extract features
    - How do we know if a representation is good?
  - This is the new frontier of machine learning because most big datasets do not come with labels.

Hypothesis Space

- One way to think about a supervised learning machine is as a device that explores a "hypothesis space".
  - Each setting of the parameters in the machine is a different hypothesis about the function that maps input vectors to output vectors.
  - If the data is noise-free, each training example rules out a region of hypothesis space.
  - If the data is noisy, each training example scales the posterior probability of each point in the hypothesis space in proportion to how likely the training example is given that hypothesis.
- The art of supervised machine learning is in:
  - Deciding how to represent the inputs and outputs
  - Selecting a hypothesis space that is powerful enough to represent the relationship between inputs and outputs but simple enough to be searched.

Searching a hypothesis space

- The obvious method is to first formulate a loss function and then adjust the parameters to minimize the loss function.
  - This allows the optimization to be separated from the objective function that is being optimized.
- Bayesians do not search for a single set of parameter values that do well on the loss function.
  - They start with a prior distribution over parameter values and use the training data to compute a posterior distribution over the whole hypothesis space.

Some Loss Functions

- Squared difference between actual and target real-valued outputs.
- Number of classification errors
  - Problematic for optimization because the derivative is not smooth.
- Negative log probability assigned to the correct answer.
  - This is usually the right function to use.
  - In some cases it is the same as squared error (regression with Gaussian output noise)
  - In other cases it is very different (classification with discrete classes needs cross-entropy error)

Generalization

- The real aim of supervised learning is to do well on test data that is not known during learning.
- Choosing the values for the parameters that minimize the loss function on the training data is not necessarily the best policy.
- We want the learning machine to model the true regularities in the data and to ignore the noise in the data.
  - But the learning machine does not know which regularities are real and which are accidental quirks of the particular set of training examples we happen to pick.
- So how can we be sure that the machine will generalize correctly to new data?

Trading off the goodness of fit against the complexity of the model

- It is intuitively obvious that you can only expect a model to generalize well if it explains the data surprisingly well given the complexity of the model.
- If the model has as many degrees of freedom as the data, it can fit the data perfectly but so what?
- There is a lot of theory about how to measure the model complexity and how to control it to optimize generalization.
  - Some of this “learning theory” will be covered later in the course, but it requires a whole course on learning theory to cover it properly (Toni Pitassi sometimes offers such a course).

A sampling assumption

- Assume that the training examples are drawn independently from the set of all possible examples.
- Assume that each time a training example is drawn, it comes from an identical distribution (i.i.d)
- Assume that the test examples are drawn in exactly the same way – i.i.d. and from the same distribution as the training data.
- These assumptions make it very unlikely that a strong regularity in the training data will be absent in the test data.
  - Can we say something more specific?

The probabilistic guarantee

$$E_{test} \le E_{train} + \left(\frac{h + h\log(2N/h) - \log(p/4)}{N} \right)^{\frac{1}{2}}$$

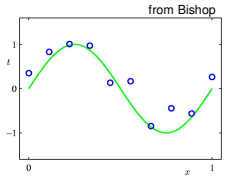
where N = size of training set
h = VC dimension of the model class = complexity
p = upper bound on probability that this bound fails

So if we train models with different complexity, we should pick the one that minimizes this bound

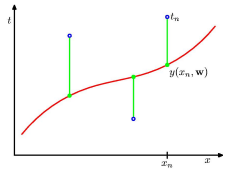
Actually, this is only sensible if we think the bound is fairly tight, which it usually isn't. The theory provides insight, but in practice we still need some witchcraft.

A simple example: Fitting a polynomial

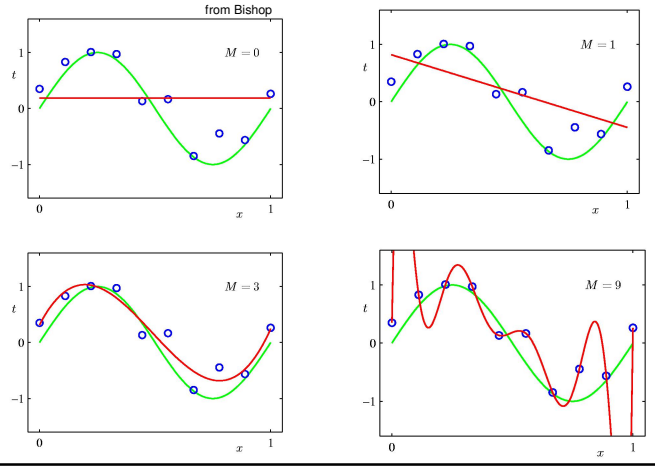
- The green curve is the true function (which is not a polynomial)
- The data points are uniform in x but have noise in y.



- We will use a loss function that measures the squared error in the prediction of y(x) from x. The loss for the red polynomial is the sum of the squared vertical errors.



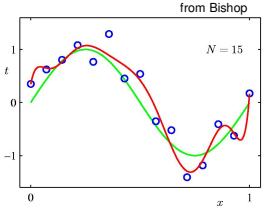
Some fits to the data: which is best?



A simple way to reduce model complexity

- If we penalize polynomials that have big values for their coefficients, we will get less wiggly solutions:

from Bishop



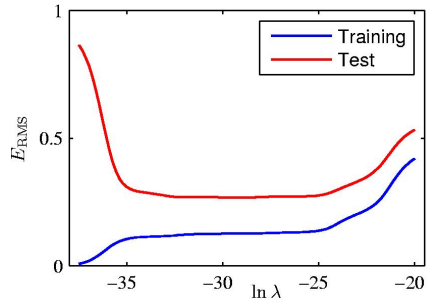
penalized loss function

regularization parameter

$$\tilde{E}(\mathbf{w}) = \frac{1}{2}\sum_{n=1}^{N}\{y(x_n, \mathbf{w}) - t_n\}^2 + \frac{\lambda}{2}\|\mathbf{w}\|^2$$

target value

Regularization: E_{RMS} vs. $\ln\lambda$



Polynomial Coefficients

	$\ln\lambda = -\infty$	$\ln\lambda = -18$	$\ln\lambda = 0$
$w_0^\star$	0.35	0.35	0.13
$w_1^\star$	232.37	4.74	-0.05
$w_2^\star$	-5321.83	-0.77	-0.06
$w_3^\star$	48568.31	-31.97	-0.05
$w_4^\star$	-231639.30	-3.89	-0.03
$w_5^\star$	640042.26	55.28	-0.02
$w_6^\star$	-1061800.52	41.32	-0.01
$w_7^\star$	1042400.18	-45.95	-0.00
$w_8^\star$	-557682.99	-91.53	0.00
$w_9^\star$	125201.43	72.68	0.01

Using a validation set

- Divide the total dataset into three subsets:
  - Training data is used for learning the parameters of the model.
  - Validation data is not used of learning but is used for deciding what type of model and what amount of regularization works best.
  - Test data is used to get a final, unbiased estimate of how well the network works. We expect this estimate to be worse than on the validation data.
- We could then re-divide the total dataset to get another unbiased estimate of the true error rate.

The Bayesian framework

- The Bayesian framework assumes that we always have a prior distribution for everything.
  - The prior may be very vague.
  - When we see some data, we combine our prior distribution with a likelihood term to get a posterior distribution.
  - The likelihood term takes into account how probable the observed data is given the parameters of the model.
    - It favors parameter settings that make the data likely.
    - It fights the prior
    - With enough data the likelihood terms always win.

A coin tossing example

- Suppose we know nothing about coins except that each tossing event produces a head with some unknown probability p and a tail with probability 1-p. Our model of a coin has one parameter, p.
- Suppose we observe 100 tosses and there are 53 heads. What is p?
- The frequentist answer: Pick the value of p that makes the observation of 53 heads and 47 tails most probable.

$$P(D) = p^{53}(1-p)^{47}$$ ← probability of a particular sequence

$$\frac{dP(D)}{dp} = 53p^{52}(1-p)^{47} - 47p^{53}(1-p)^{46}$$

$$= \left(\frac{53}{p} - \frac{47}{1-p}\right)\left[p^{53}(1-p)^{47}\right]$$

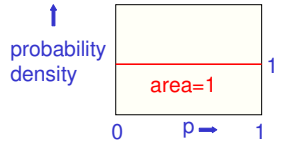
$$= 0 \;\; if \;\; p = .53$$

Some problems with picking the parameters that are most likely to generate the data

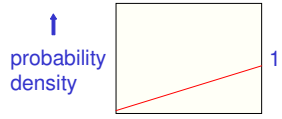
- What if we only tossed the coin once and we got 1 head?
  - Is p=1 a sensible answer?
    - Surely p=0.5 is a much better answer.
- Is it reasonable to give a single answer?
  - If we don't have much data, we are unsure about p.
  - Our computations of probabilities will work much better if we take this uncertainty into account.

Using a distribution over parameter values

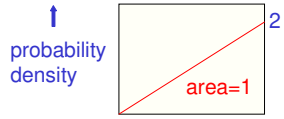
- Start with a prior distribution over p. In this case we used a uniform distribution.



- Multiply the prior probability of each parameter value by the probability of observing a head given that value.

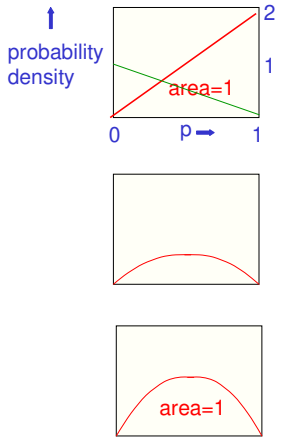


- Then scale up all of the probability densities so that their integral comes to 1. This gives the posterior distribution.



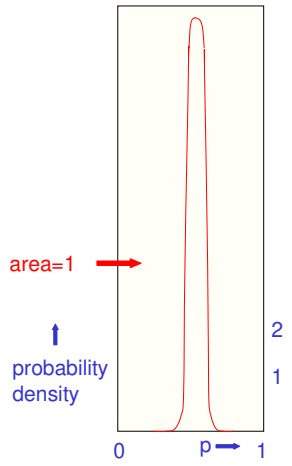
Lets do it again: Suppose we get a tail

- Start with a prior distribution over p.
- Multiply the prior probability of each parameter value by the probability of observing a tail given that value.
- Then renormalize to get the posterior distribution. Look how sensible it is!



Lets do it another 98 times

- After 53 heads and 47 tails we get a very sensible posterior distribution that has its peak at 0.53 (assuming a uniform prior).



Bayes Theorem

joint probability

conditional probability

$$p(D)\,p(W \mid D) = p(D,W) = p(W)\,p(D \mid W)$$

Prior probability of weight vector W

Probability of observed data given W

$$p(W \mid D) = \frac{p(W)\;p(D \mid W)}{p(D)}$$

Posterior probability of weight vector W given training data D

$$p(D) = \int_W p(W)\,p(D \mid W)$$

A cheap trick to avoid computing the posterior probabilities of all weight vectors

- Suppose we just try to find the most probable weight vector.
  - We can do this by starting with a random weight vector and then adjusting it in the direction that improves p(W | D).
- It is easier to work in the log domain. If we want to minimize a cost we use negative log probabilities:

$$p(W \mid D) = p(W)\;p(D \mid W) \;/\; p(D)$$

$$Cost = -\log p(W \mid D) = -\log p(W) - \log p(D \mid W) + \log p(D)$$

Why we maximize sums of log probs

- We want to maximize the product of the probabilities of the outputs on the training cases
  - Assume the output errors on different training cases, c, are independent.

$$p(D\,|\,W) = \prod_c p(d_c\,|\,W)$$

- Because the log function is monotonic, it does not change where the maxima are. So we can maximize sums of log probabilities

$$\log p(D\,|\,W) = \sum_c \log p(d_c\,|\,W)$$

A even cheaper trick

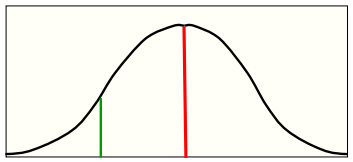
- Suppose we completely ignore the prior over weight vectors
  - This is equivalent to giving all possible weight vectors the same prior probability density.
- Then all we have to do is to maximize:

$$\log p(D\,|\,W) = \sum_c \log p(D_c\,|\,W)$$

- This is called maximum likelihood learning. It is very widely used for fitting models in statistics.

Supervised Maximum Likelihood Learning

- Minimizing the squared residuals is equivalent to maximizing the log probability of the correct answer under a Gaussian centered at the model's guess.



d = the correct answer

y = model's estimate of most probable value

$$y_c = f(input_c, W)$$

$$p(output = d_c\,|\,input_c, W) = p(d_c\,|\,y_c) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(d_c - y_c)^2}{2\sigma^2}}$$

$$-\log p(output = d_c\,|\,input_c, W) = k + \frac{(d_c - y_c)^2}{2\sigma^2}$$

Supervised Maximum Likelihood Learning

- Finding a set of weights, W, that minimizes the squared errors is exactly the same as finding a W that maximizes the log probability that the model would produce the desired outputs on all the training cases.
  - We implicitly assume that zero-mean Gaussian noise is added to the model's actual output.
  - We do not need to know the variance of the noise because we are assuming it's the same in all cases. So it just scales the squared error.