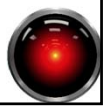
Learning

Professor Marie Roch

Chapter 18, Russell & Norvig

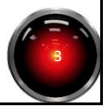


Learning

- Agents can learn to improve:
  - inference from percepts
  - information about world evolution
    - as the result of a changing world
    - as the result of actions
  - utility estimators
  - action choices
    - either update condition-action maps
    - goal modification to maximize utility

What we want to learn

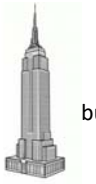
- Mapping function
  - Inputs are factored representations e.g. a vector of values
  - Outputs are
    - discrete (e.g. categorical)
    - continuous



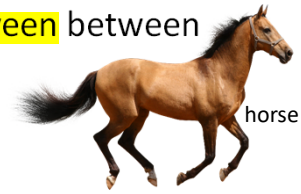
Types of learning

- Inductive – Learn map between between input/output pairs

ice cream



building



horse

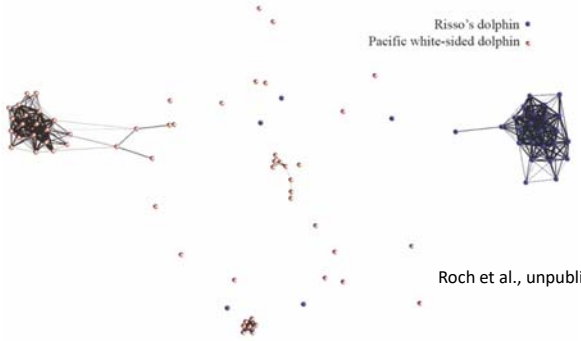
- Deductive – Creating rules that are logically entailed, such as if I am in a dark cave and I don't smell a breeze, I'm not going to step into a pit.

Inductive - characterized by the inference of general laws from particular instances.

Deductive - characterized by or based on the inference of particular instances from a general law.

Learners vary based on their feedback

- Unsupervised learning
  - No explicit feedback
  - Goal is to cluster "similar" things



Roch et al., unpublished

Learners

- Reinforcement learning
  - Learner is given rewards/punishments for actions
  - Example: Positive reinforcement animal training
- Supervised learning
  - Each input is paired with a label or value and the agent attempts to learn to predict the labels/values for novel data.
- Hybrids are possible, such as semi-supervised learning where a small set of labeled data accompanies a large set of unlabeled data.

Caveat about labeled data sets

- We refer to labels as "ground truth."
- One should be cautious with ground truth… Why?

Supervised learning

- Suppose there exists an unknown f: x → y such that

$$(y_1 = f(x_1)) \wedge (y_2 = f(x_2)) \wedge (y_3 = f(x_3)) \wedge \ldots$$

and we are given only a *training set*

$$(x_1, y_1), (x_2, y_2), (x_3, y_3), \ldots$$

- Supervised learning estimates a function h: x→y that approximates f.

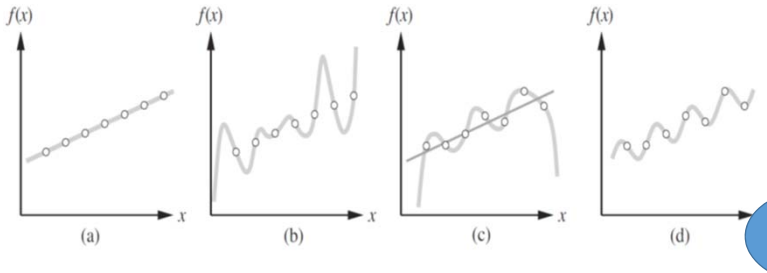
Supervised learning

- Function h is the *hypothesis* and our estimation is a search in *hypothesis function space* for a good hypothesis
- Learning is a search for a good hypothesis.
- How do we measure goodness?
  - Evaluate the function on a labeled *test* set.
  - The test set must be distinct from the training set:
    $$\text{training} \cap \text{test} = \varnothing$$
  - We say h *generalizes* well if it performs well on the test set.

Why do we need to test on novel data?

How to choose amongst functions?

Ockham's razor – Use simplest hypothesis consistent with the data



All of these functions fit the training data, but which one is most likely to correctly predict new data?

good note taking app allowing notes on pdf can be pushed to Kindles

Hypothesis spaces

of features used from dataset. incl labels

- The more complex a hypothesis space, the more difficult it is to find a good hypothesis.

? Complex?

- Fits well with Ockham's Razer.

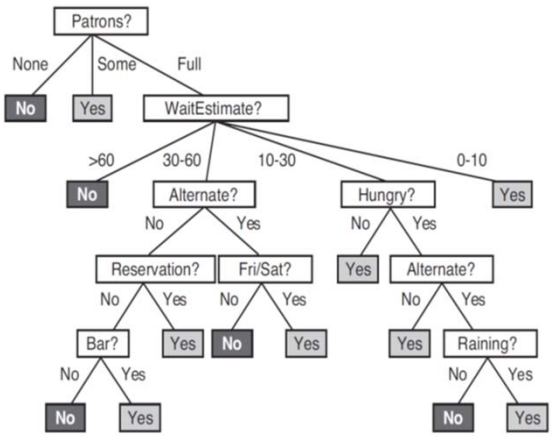
Supervised learning

- Function f could be stochastic
  - If so, f is not simply a function of x
  - In these cases, we learn a conditional probability distribution P(Y|x)
- What are we learning:
  - y is categorical → *classification*
    - example: y∈{happy, sad, angry, serious}
    - binary classifier – special case with exactly two classes
  - y is numeric → *regression*
    - example: y = change in sea level (m) since 1990

Decision tree learner

- Answers a series of questions to arrive at a solution
- For now, we restrict our discussion to
  - questions that have categorical (discrete) answers
  - binary classification decisions

Dr. Stuart Russell is hungry…



Professor Russell's decision tree for where to eat…
9 questions from 10 attributes (price is not used)

Learning a tree from examples

Example	Input Attributes										Goal
	Alt	*Bar*	*Fri*	*Hun*	*Pat*	*Price*	*Rain*	*Res*	*Type*	*Est*	*WillWait*
$\mathbf{x}_1$	*Yes*	*No*	*No*	*Yes*	*Some*	*$$$*	*No*	*Yes*	*French*	*0–10*	$y_1 = Yes$
$\mathbf{x}_2$	*Yes*	*No*	*No*	*Yes*	*Full*	*$*	*No*	*No*	*Thai*	*30–60*	$y_2 = No$
$\mathbf{x}_3$	*No*	*Yes*	*No*	*No*	*Some*	*$*	*No*	*No*	*Burger*	*0–10*	$y_3 = Yes$
$\mathbf{x}_4$	*Yes*	*No*	*Yes*	*Yes*	*Full*	*$*	*Yes*	*No*	*Thai*	*10–30*	$y_4 = Yes$
$\mathbf{x}_5$	*Yes*	*No*	*Yes*	*No*	*Full*	*$$$*	*No*	*Yes*	*French*	*>60*	$y_5 = No$
$\mathbf{x}_6$	*No*	*Yes*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Italian*	*0–10*	$y_6 = Yes$
$\mathbf{x}_7$	*No*	*Yes*	*No*	*No*	*None*	*$*	*Yes*	*No*	*Burger*	*0–10*	$y_7 = No$
$\mathbf{x}_8$	*No*	*No*	*No*	*Yes*	*Some*	*$$*	*Yes*	*Yes*	*Thai*	*0–10*	$y_8 = Yes$
$\mathbf{x}_9$	*No*	*Yes*	*Yes*	*No*	*Full*	*$*	*Yes*	*No*	*Burger*	*>60*	$y_9 = No$
$\mathbf{x}_{10}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$$$*	*No*	*Yes*	*Italian*	*10–30*	$y_{10} = No$
$\mathbf{x}_{11}$	*No*	*No*	*No*	*No*	*None*	*$*	*No*	*No*	*Thai*	*0–10*	$y_{11} = No$
$\mathbf{x}_{12}$	*Yes*	*Yes*	*Yes*	*Yes*	*Full*	*$*	*No*	*No*	*Burger*	*30–60*	$y_{12} = Yes$

Figure 18.3 Examples for the restaurant domain.

R&N p. 700

Constructing a tree from examples

- Which question to ask first?
- What do you look for when you play 20 questions?

  Chances are, you intuitively use information theory…

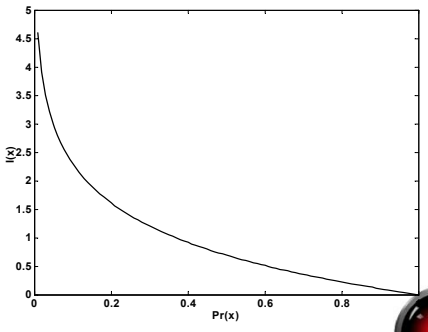
X = Class

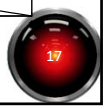
Quantity of information

- Amount of surprise that one sees when observing an event.
- If an event is rare, we can derive a large quantity of information (measured in bits) from it.

$$I(x_i) = \log_2 \frac{1}{\mathrm{P}(x_i)}$$

1/3





Note: We use log base 2 and will start omitting the base later on.

Expectation (review)

- An expected value is the value that we expect to see most often.
- We sum the product of each possible value and the probability that it occurs

$$E[X] = \sum_{x_i \in S} x_j \, \mathrm{P}(x_i) \text{ where S is the set of all possible values of X}$$

Entropy

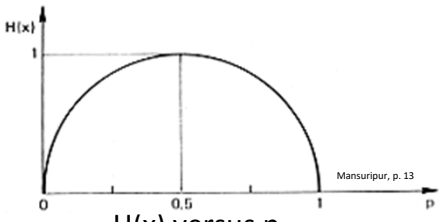
- Entropy is defined as the expected amount of information (average amount of surprise) and is usually denoted by the symbol H.

$$
\begin{aligned}
H(X) &= E[I(X)] \\
&= \sum_{x_i \in S} \mathrm{P}(x_i) I(x_i) \qquad S \text{ is all possible symbols} \\
&= \sum_{x_i \in S} \mathrm{P}(x_i) \log_2 \frac{1}{\mathrm{P}(x_i)} \quad \text{definition } I(x_i) \\
&= E[-\log_2 \mathrm{P}(X)]
\end{aligned}
$$

classtimes added only positives

Example

- Assume
  - X = {0, 1}
  - $\mathrm{P}(X) = \begin{cases} p & X = 0 \\ 1-p & X = 1 \end{cases}$
- Then



H(x) versus p

$$
\begin{aligned}
H(X) &= E[I(X)] \\
&= -p \log p - (1-p) \log(1-p)
\end{aligned}
$$

Price response:
Not Binary
classes = 3

total number of Samples = 12
of_samples_class_$ = 7 => 7/12
of_samples_class_$$ = 2 => 2/12
of_samples_class_$$$ = 3 => 3/12

Positive
negative
5/12
10/12
9/12

Restaurant example

- WillWait response:
  - 6 positive
  - 6 negative

binary

Example	Input Attributes										Goal
	Alt	Bar	Fri	Hun	Pat	Price	Rain	Res	Type	Est	WillWait
x_1	Yes	No	No	Yes	Some	$$$	No	Yes	French	0–10	y_1 = Yes
x_2	Yes	No	No	Yes	Full	$	No	No	Thai	30–60	y_2 = No
x_3	No	Yes	No	No	Some	$	No	No	Burger	0–10	y_3 = Yes
x_4	Yes	No	Yes	Yes	Full	$	Yes	No	Thai	10–30	y_4 = Yes
x_5	Yes	No	Yes	No	Full	$$$	No	Yes	French	>60	y_5 = No
x_6	No	Yes	No	Yes	Some	$$	Yes	Yes	Italian	0–10	y_6 = Yes
x_7	No	Yes	No	No	None	$	Yes	No	Burger	0–10	y_7 = No
x_8	No	No	No	Yes	Some	$$	Yes	Yes	Thai	0–10	y_8 = Yes
x_9	No	Yes	Yes	No	Full	$	Yes	No	Burger	>60	y_9 = No
x_{10}	Yes	Yes	Yes	Yes	Full	$$$	No	Yes	Italian	10–30	y_{10} = No
x_{11}	No	No	No	No	None	$	No	No	Thai	0–10	y_{11} = No
x_{12}	Yes	Yes	Yes	Yes	Full	$	No	No	Burger	30–60	y_{12} = Yes

Figure 18.3 Examples for the restaurant domain.

- Entropy

$$H(x) = -\frac{p}{p+n}\log_2\frac{p}{p+n} + -\frac{n}{p+n}\log_2\frac{n}{p+n}$$
$$= -\frac{6}{6+6}\log_2\frac{6}{6+6} + -\frac{6}{6+6}\log_2\frac{6}{6+6}$$
$$= \log_2 2 = 1$$

$$H(x) = -\left(\frac{7}{12}\right)\log_2\left(\frac{12}{7}\right) + \left(\frac{5}{12}\right)\log_2\left(\frac{12}{5}\right) + \left(\frac{2}{12}\right)\log_2\left(\frac{12}{2}\right) + \left(\frac{10}{12}\right)\log_2\left(\frac{12}{10}\right) + \left(\frac{3}{12}\right)\log\left(\frac{12}{3}\right) + \left(\frac{9}{12}\right)\log_2\left(\frac{12}{9}\right)$$

get rid of (-) and sign

Class 1 Binary entropy value $B\left(\frac{7}{12}\right)$

Class 2 Binary entropy value $B\left(\frac{2}{12}\right)$

Class 3 $B\left(\frac{3}{12}\right)$

only for binary → $B\left(\frac{3}{12}\right)$

Average

Entropy and tree questions

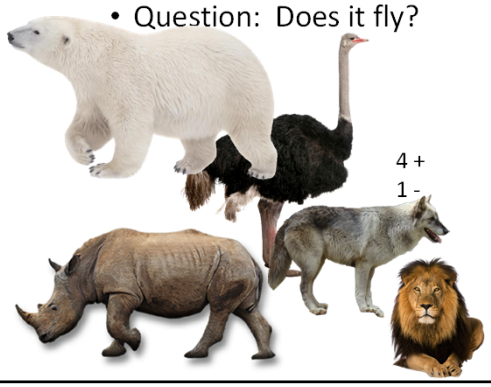
- Fig. 18.3 has an equal number of positive and negative examples (6 each: p=n=6)
- Training data has entropy of 1 bit:

$$H(x) = -\frac{p}{p+n}\log_2\frac{p}{p+n} + -\frac{n}{p+n}\log_2\frac{n}{p+n}$$
$$= -\frac{1}{2}\log_2\frac{1}{2} + -\frac{1}{2}\log_2\frac{1}{2}$$
$$= \log_2 2 = 1$$

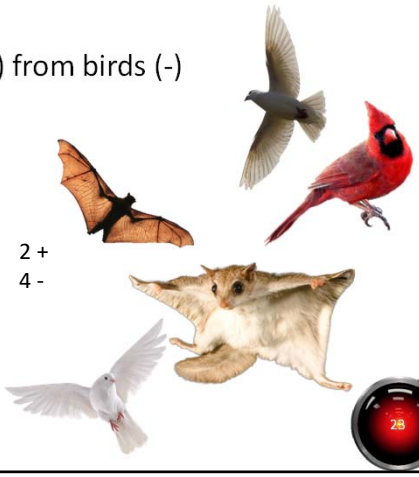
Tree questions

Animal set:
6 mammals (+), 5 birds (-)

- Tree questions have a binary response
- Suppose
  - Goal: separate mammals (+) from birds (-)
  - Question: Does it fly?



4 +
1 -



2 +
4 -

info is low

Tree question entropy

on average

- Remember, entropy is: $E\left[I\left(P(X)\right)\right] = E\left[-\log_2 \mathrm{P}(X)\right]$
- For binary categories, we define a short hand:
  - $q = {}^{p}/_{p+n}$, , the positive rate
  - $1 - q = {}^{n}/_{p+n}$, the negative rate
  - $B(q) = \mathrm{E}[\mathrm{I}(\mathrm{P}(\mathrm{X}))] = -q\log_2 q - (1-q)\log_2(1-q)$

Not Binary categories

Tree question entropy

Bird/mammal example

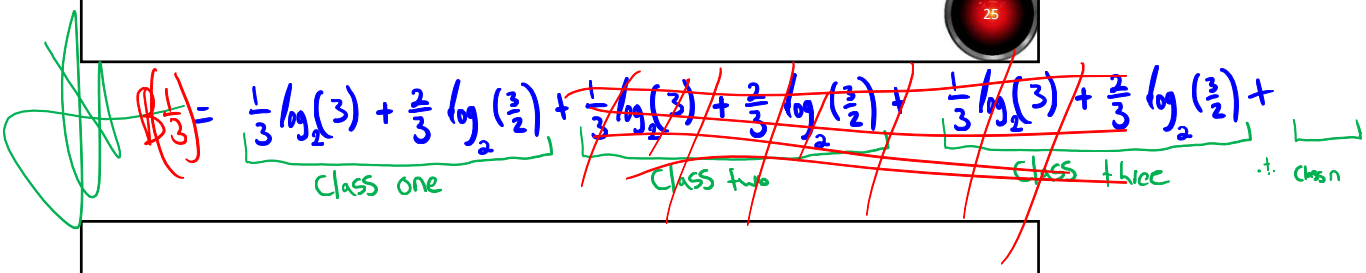
	p	n	q (+ rate)	B(q)
before question	6	5	6/11	0.99
¬flies	4	1	4/5	0.72
flies	2	4	1/3	0.92

Sample computation flies:

Shorthand for binary

$$q = \frac{2}{2+4} = \frac{1}{3}$$

$$B\left(\frac{1}{3}\right) = -\frac{1}{3}\log_2\left(\frac{1}{3}\right) - \left(1-\frac{1}{3}\right)\log_2\left(1-\frac{1}{3}\right) = \frac{1}{3}\log_2\left(\frac{1}{3}\right) - \left(\frac{2}{3}\right)\log_2\left(\frac{2}{3}\right) \approx .92$$



Entropy and tree questions

- Patrons – Categories (None, Some, Full)
  - None: 2 examples: B(0/2) = 0
  - Some: 4 examples: B(4/4) = 0
  - Many: 6 examples: B(2/6) = .918
- Restaurant type (French, Italian, Thai, Burger)
  - French: B(1/2) = 1
  - Italian: B(1/2) = 1
  - Thai: B(2/4) = 1
  - Burger: B(2/4) = 1

Information gain

- Goal: reduce the amount of information needed to represent the problem
- We can represent the remaining entropy after dividing data into d groups with question A as follows:

$$\text{Remainder}(A) = \sum_{k=1}^{d} \frac{p_k + n_k}{p+n} B\left(\frac{p_k}{p_k + n_k}\right)$$

and the information gain as:

$$Gain(A) = B\left(\frac{p}{p+n}\right) - \text{Remainder}(A)$$

only for binary?

Information gain examples

- Mammal/bird flight question
  - Split 11 animals into two groups of size 5 (¬flies) and 6 (flies).
  - $\text{Remainder(Does it fly?)} = \underbrace{\frac{2+4}{6+5} B\left(\frac{1}{3}\right)}_{flies} + \underbrace{\frac{4+1}{6+5} B\left(\frac{4}{5}\right)}_{\neg flies} = \frac{6}{11} \cdot .92 + \frac{5}{11} \cdot .72 \approx .83$
  - $\text{Gain(Does it fly?)} = B\left(\frac{6}{6+5}\right) - \text{Remainder(Does it fly?)} = 0.99 - 0.83 = 0.16$

Information gain examples

- Patrons – Categories (None, Some, Full)
  None: 2 examples: B(0/2) = 0
  Some: 4 examples: B(4/4) = 0
  Many: 6 examples: B(2/6) = .918

$$Gain(Patrons) = B\left(\frac{6}{6+6}\right) - \left(\frac{2}{12}\cdot 0 + \frac{4}{12}\cdot 0 + \frac{6}{12}\cdot .918\right) \approx .541 \text{ bits}$$

- Restaurant type (French, Italian, Thai, Burger)
  French: B(1/2) = 1
  Italian: B(1/2) = 1
  Thai: B(2/4) = 1
  Burger: B(2/4) = 1

$$Gain(Type) = B\left(\frac{6}{6+6}\right) - \left(\frac{2}{12}\cdot 1 + \frac{2}{12}\cdot 1 + \frac{4}{12}\cdot 1 + \frac{4}{12}\cdot 1\right) = 0 \text{ bits}$$

Decision tree learner

```
def decision-tree-learner(examples, attributes, parent_examples):
    if empty(examples):
        return plurality-value(parent_examples)  # pick whatever parent had most of
    else if all examples of same class:
        return the class
    else if empty(attributes):  # no more questions to ask
        return plurality-value(examples)
    else:
        a = arg max_{a∈attributes} importance(a)     # information gain or other measure
        t = new tree(a)      # Create a new tree rooted on most important question
        for each value v associated with a:
            vexamples = {e : e ∈examples such that e has value v for attribute a}
            subtree = decision-tree-learner(vexamples, attributes – a, examples)
            t.add_branch(v, subtree)  # Add in new subtree with current value as branch label
        return t
```

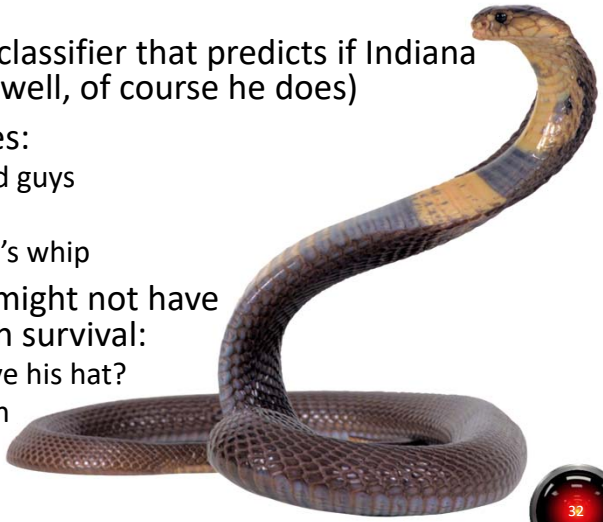
Will Indie survive?

image credit: Indian Jones © Lucasfilm Ltd.

31

Will Indie survive?

- We can build a classifier that predicts if Indiana Jones survives (well, of course he does)
- Possible features:
  - Number of bad guys
  - any snakes?
  - length of Indie's whip
- Some features might not have much to do with survival:
  - Does Indie have his hat?
  - Did Indie brush his teeth?



AbsurdWordPreferred - DeviantArt

32

Features and overlearning

- Useless features are not good for prediction, but…

  a learner may pick up on random patterns in the training data and incorporate these into the rules

- Example:
  - Random six sided fair die, learn whether or not we roll 5.
  - Height from which we roll should have any bearing on P(X=5)=.2
  - Decision tree may again pick up random patterns, but the lowest classification error rule is to simply say: we will not roll a 5.

Generalization and overfitting

- Learning random patterns that don't affect function f is called overfitting.

- Overfit models do a great job predicting *training* data, but *do not predict novel data well.*

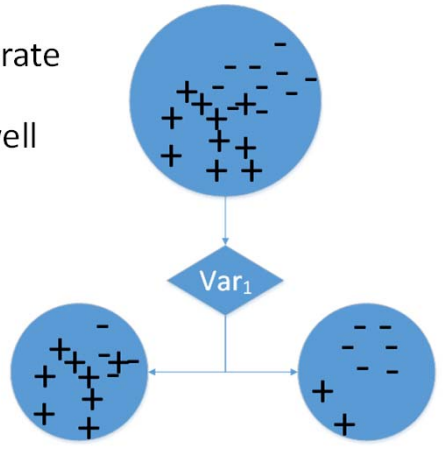
- Decision trees have a tendency to overfit.

Pruning Decision trees

- Overfitting of decision trees is addressed by pruning.
- For each leaf node, we ask ourselves if we had good information gain.
  If the node was informative, we keep it.
  If we didn't learn anything, we discard.
- NOTE: This is done *after* the tree is trained.

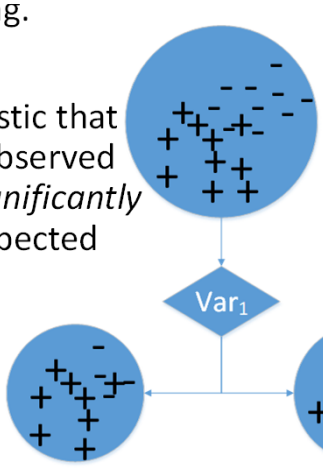
Pruning decision trees

- How do we know if our decisions were any good?
- Our goal was to separate into the positive and negative classes as well as possible.



Pruning decision trees

- Here we didn't do a very good job of separating.
- Can we devise a statistic that let's us know if our observed split is statistically *significantly different* from the expected ratio?



χ^2 Test

- Suppose decision tree splits a node into *v* categories.
- If the node does not add any new information, then we expect each child to have about the same distribution of class labels (e.g. similar % of +/- examples in a 2 class problem).

χ^2 Test

- Let's restrict an example to our two class (v=2; positive and negative) example.

$$P(p) = \frac{p}{p+n}, P(n) = \frac{n}{p+n} \text{ for the whole data set}$$

- The question will split the examples into two subsets k=1,2 with p_k positive examples and n_k negative examples.
- How many positive and negatives would we expect if there was no change in distribution from the training data?

$$\hat{p}_k = (p_k + n_k)\frac{p}{p+n}, \hat{n}_k = (p_k + n_k)\frac{n}{p+n}$$

χ^2 Test

- We can look at how much our categories differ from what would be expected if the proportion of categories did not change

χ^2 test statistic

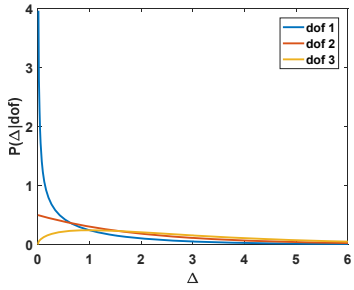
$$\Delta = \sum_{k=1}^{d} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}$$ measure of deviation

where d is the number of split values

- When Δ is small, we are close to the original distribution.

χ^2 Test

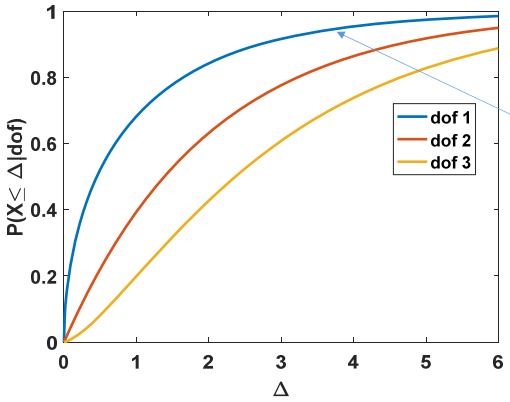
- The test statistic has a distribution that is related to the number of categories – 1. This is referred to as the *degrees of freedom* (dof) and for a binary classifier, the dof is 2-1=1.



The formula for this is beyond our scope, but the plot shows the probability of having a value of Δ assuming that the distributions are identical.

Cumulative density function (CDF)

- Suppose we integrate P(X|dof) up to Delta



With 1 dof, 95% of Δs expected to be less than 3.84

The formula for χ^2 is beyond our scope, but the plot shows the probability of having a value of Δ assuming that the distributions are identical.

χ^2 test example

Let's return to the bird/mammal example:

	p	n	q (+ rate)	B(q)
before question	6	5	6/11	0.99
¬flies	4	1	4/5	0.72
flies	2	4	1/3	0.92

Expected in in each split if the distribution does not change: $\hat{p}_k = p \times \frac{p_k + n_k}{p+n}, \hat{n}_k = n \times \frac{p_k + n_k}{p+n}$

$$\hat{p}_{flies} = 6\frac{2+4}{6+5} = \frac{36}{11}, \hat{n}_{flies} = 5\frac{2+4}{11} = \frac{30}{11}$$

$$\hat{p}_{\neg flies} = 6\frac{4+1}{11} = \frac{30}{11}, \hat{n}_{\neg flies} = 5\frac{4+1}{11} = \frac{25}{11}$$

χ^2 test example

- Compute χ^2 statistic Δ

$$\Delta = \sum_{k=1}^{2} \frac{(p_k - \hat{p}_k)^2}{\hat{p}_k} + \frac{(n_k - \hat{n}_k)^2}{\hat{n}_k}$$

$$= \underbrace{\frac{\left(\frac{4}{5} - \frac{36}{11}\right)^2}{\frac{36}{11}} + \frac{\left(\frac{1}{5} - \frac{30}{11}\right)^2}{\frac{30}{11}}}_{flies} + \underbrace{\frac{\left(\frac{1}{3} - \frac{30}{11}\right)^2}{\frac{30}{11}} + \frac{\left(\frac{2}{3} - \frac{25}{11}\right)^2}{\frac{25}{11}}}_{\neg flies}$$

$$\approx 1.868 + 2.342 + 2.101 + 1.345$$

$$\approx 7.45$$

$\hat{p}_{flies} = \frac{36}{11}, \hat{n}_{flies} = \frac{30}{11}$

$\hat{p}_{\neg flies} = \frac{30}{11}, \hat{n}_{\neg flies} = \frac{25}{11}$

from table:

$p_{flies} = \frac{4}{5}, n_{flies} = \frac{1}{5}$

$p_{\neg flies} = \frac{1}{3}, n_{\neg flies} = \frac{2}{3}$

χ^2 test example

- We have one dof and Δ=7.52. Significant change in distributions?
  - 0.994 of 1 dof problems without significant change have Δ<7.52
  - About a 0.006 chance that this is not significantly different.
  - Good idea to keep this split
- In general:
  - Define an acceptable level of error (e.g. the 0.006) called a p-value.
  - Very common to use p=0.05
  - Look up the χ^2 inverse cdf for the desired p-value.
  - Compute the χ^2 statistic.
  - Prune if the split does not present a significant difference

χ^2 Test

- Python does not have χ^2 routines, but the Scientific Python library does.

```
import sicpy.stats.chi2
dof = 1
p05 = scipy.stats.chi2.ppdf(.95, dof) # inverse CDF: 3.84
```

- Caveat: Only try this on leaf nodes of a constructed tree!
  - Sometimes, multiple levels have more power than a single one.
  - Pruning as we go can prevent us from ever seeing this.

More thoughts on decision trees

- Continuous/integer-valued attributes
  - Don't create infinite branches
  - Select a split point
    - Sort values
    - Keep running total of number of +/- examples for each point in sorted list and pick the separating point that gives the best separation.
- See text for information on multivalued attributes and continuous-valued outputs.

Decision tree summary

Relatively straight-forward learners that

- recursively partition the feature space into hyperplanes,
- are sensitive to overtraining, but have methods to prune,
- and are easy for humans to understand

Do I have a good hypothesis function?

- Assume data are *independent* and *identically distributed* (**iid**)
  - Independent – Examples $e_j=(x_j,y_j)$, $e_k=(x_k,y_k)$ are unrelated to one another when $j \neq k$.

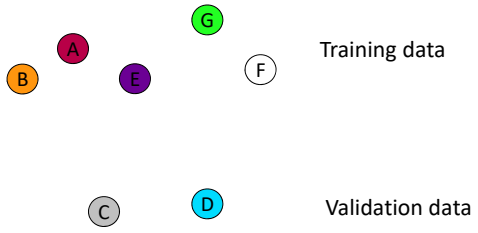
$$P(E_j \mid E_k) = P(E_j)$$

  - Identically distributed – Whatever process generated e_j is also responsible for generating e_k and did not change.

Warning: iid assumptions do not always hold!

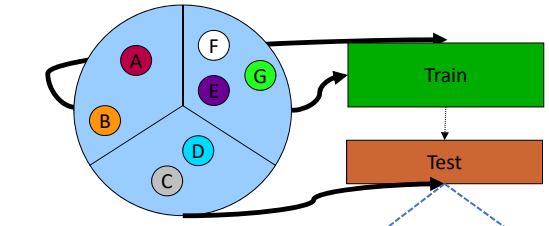
Do I have a good hypothesis function?

- We cross-validate the learner on a separate validation set



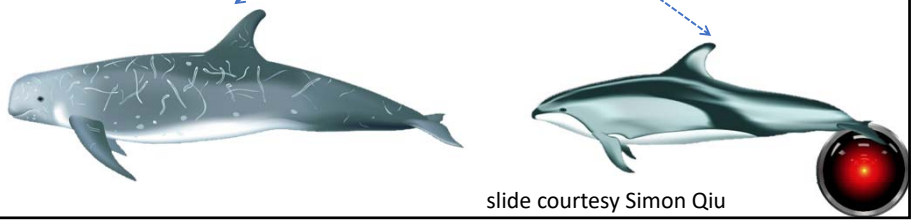
- Problem: We don't exploit all our data

k-fold cross validation



k=3

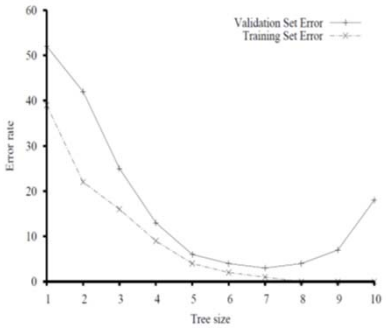
Extreme case:
leave-one-out cross validation (aka jackknife)
k=N



slide courtesy Simon Qiu

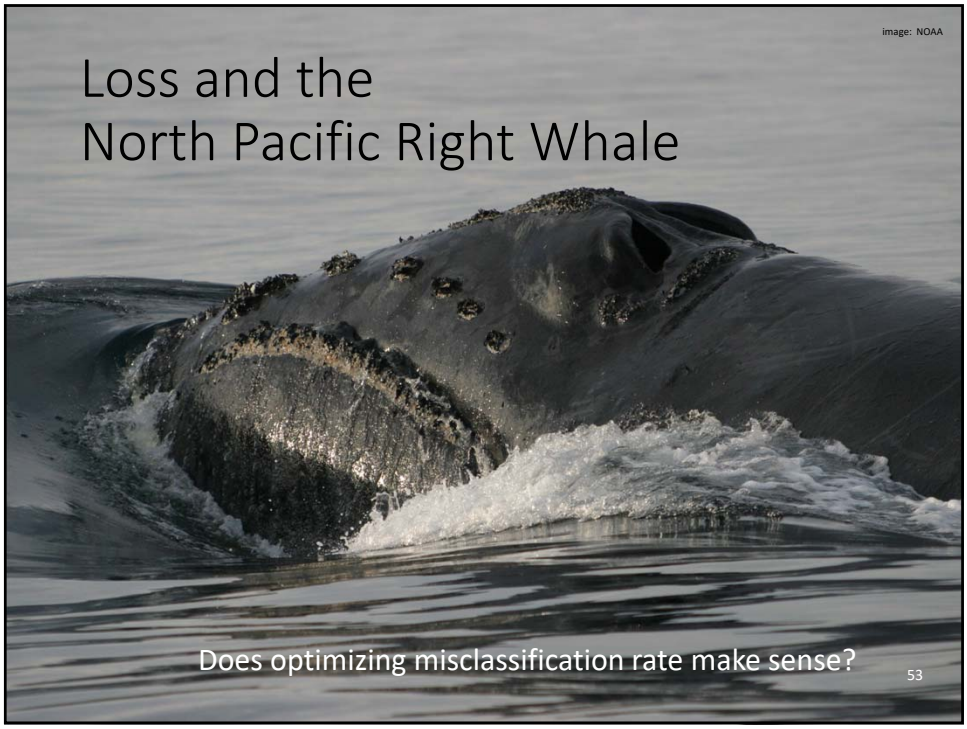
Model selection

- More complex models (e.g. more nodes in a decision tree) learn the training data better, but are they really better?
- For this, we look at validation error



Note: There are also statistics that can help us select models (beyond our scope)

Loss and the North Pacific Right Whale



Does optimizing misclassification rate make sense?

53

Loss

- Loss functions are a form of utility function that provide a cost for misclassifcation

$$L(x, y, \hat{y}) = \text{cost}(\text{predicting h(x)}=\hat{y} \text{ given f(x)}=y)$$

- Suppose that it so useful to find a right whale that we do not mind misclassifying a bunch of non right whales as whales

$$L(x, y = \text{right whale}, \hat{y} = \text{other}) = 10$$
$$L(x, y = \text{right whale}, \hat{y} = \text{right whale}) = 0$$
$$L(x, y = \text{other}, \hat{y} = \text{right whale}) = 1$$
$$L(x, y = \text{other}, \hat{y} = \text{other}) = 0$$

Loss

- Some learners attempt to minimize loss
- Common loss functions

$$L_1(x, y, \hat{y}) = |y - \hat{y}| \quad \text{absolute loss function}$$

$$L_2(x, y, \hat{y}) = (y - \hat{y})^2 \quad \text{squared loss function}$$

$$L_{0/1}(x, y, \hat{y}) = \begin{cases} 0 & y = \hat{y} \\ 1 & \text{otherwise} \end{cases} \quad 0/1 \text{ loss function}$$

Generalization loss

- What is our loss when we use a novel data set ε?
- The expected loss requires the distribution of (X,Y) which we probably do not have:

$$GenLoss_L(h) = \sum_{(x,y)\in\epsilon} L(x, y, h(x))P(x, y)$$

but we can estimate it empirically on a finite set of examples *E* of *N* samples:

$$EmpLoss_L(h) = \frac{1}{N} \sum_{(x,y)\in E} L(x, y, h(x))$$

Note: Generalization loss is frequently referred to as *risk*

Generalization loss

- Selection of our learner *h** now becomes:

$$h^* = \arg\min_{h \in H} EmpLoss_{L,E}(h)$$

- Are we guaranteed *h** = *f*? No:
  - Unrealizability: *f* may not be in *H*
  - Variance: Learners return different f's for different training sets
  - Noise:
    - f may be noisy (e.g. stochastic component – different y's for the same x)
    - The training samples may have mis-measured attributes or incorrect labels
    - Might not have measured important attributes.
  - Complexity: Learner may not achieve a global minimum.

Regularization

- Occam's Razor states less complex models are better.
- Can we incorporate this into our model selection?

$$\text{Cost}(h) = EmpLoss(h) + \lambda Complexity(h)$$
$$h^* = \arg\min_{h \in H} \text{Cost}(h)$$

- The cost function is called a regularization function

Complexity models are beyond our scope, but if you want to know more read about MDL in chapter 20 or information criteria (e.g. AIC,BIC)

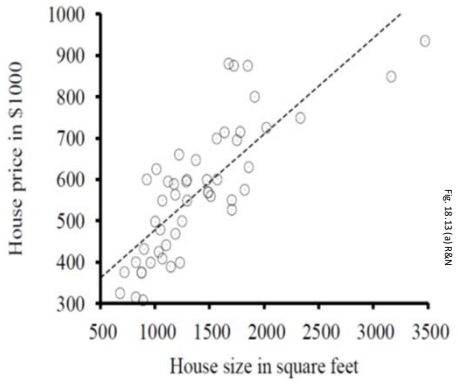
Reducing model complexity

- Learner complexity can be reduced by pruning the feature space:
  - feature selection
  - principle components analysis
  - nonlinear dimension reduction

You are not responsible for 18.5 – The Theory of Learning

Linear models

- Can be used for classification & regression
- We will start with
  - univariate
  - regression



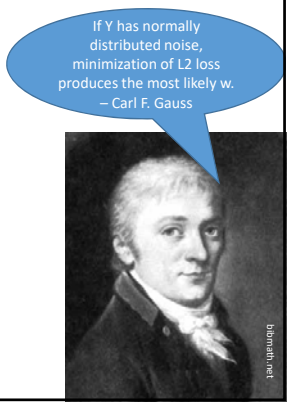
Univariate linear regression

- Linear combinations of weights w_i with input x to predict y:

$$y = w_1 x + w_0$$

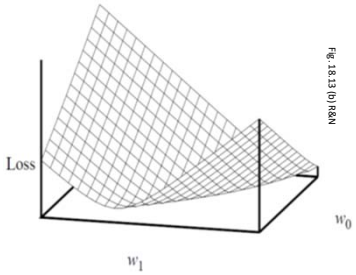
- Objective: Find vector $\vec{w} = \begin{bmatrix} w_0 \\ w_1 \end{bmatrix}$ that minimizes a loss function

$$\begin{aligned} Loss(h_w) &= \sum_{j=1}^{N} L_2(y_j, h_w(x_j)) \\ &= \sum_{j=1}^{N} \left(y_j - h_w(x_j)\right)^2 \\ &= \sum_{j=1}^{N} \left(y_j - (w_1 x_j + w_0)\right)^2 \end{aligned}$$



Minimizing L2 loss

- We want to find $w^* = \arg\min_w Loss(h_w)$
- Derivative of loss function is convex → single global minimum



Weight space plot of loss

Minimizing L2 loss

- Set derivatives of both unknowns to 0

$$\frac{\partial}{\partial w_0}\sum_{j=1}^{N}\left(y_j-(w_1x_j+w_0)\right)^2=0$$

$$\sum_{j=1}^{N}2\left(y_j-(w_1x_j+w_0)\right)\frac{\partial}{\partial w_0}\left(y_j-(w_1x_j+w_0)\right)=0$$

$$\sum_{j=1}^{N}\left(y_j-(w_1x_j+w_0)\right)=\frac{0}{2}$$

$$\sum_{j=1}^{N}y_j-w_1\sum_{j=1}^{N}x_j-Nw_0=0$$

$$w_0=\frac{\sum_{j=1}^{N}y_j-w_1\sum_{j=1}^{N}x_j}{N}$$

$$\frac{\partial}{\partial w_1}\sum_{j=1}^{N}\left(y_j-(w_1x_j+w_0)\right)^2=0$$

do this one at home

Minimizing L2 loss

- We now have 2 equations in w_0 and w_1, and can use algebra to solve:

$$w_0=\frac{\sum_{j=1}^{N}y_j-w_1\sum_{j=1}^{N}x_j}{N}$$

$$w_1=\frac{N\sum_{j=1}^{N}x_jy_j-\sum_{j=1}^{N}x_j\sum_{j=1}^{N}y_j}{N\sum x_j^2-\left(\sum_{j=1}^{N}x_j\right)^2}$$

Beyond linear models

- Not all loss functions have closed form solutions
- In these cases, we return to what we learned about optimization and use a gradient descent method

```
def gradient-descent(w):
  while not converged:
    for i in 1:len(w):
```

$$w_i = w_i - \alpha \frac{\partial}{\partial w_i} Loss(w)$$

- α is the learning rate
  - We called this the step size in optimization
  - Can be fixed, decay over time, or be adaptively set

Beyond linear models

- In general for vector **w**

$$\begin{aligned}\frac{\partial}{\partial w_i} Loss(\vec{w}) &= \frac{\partial}{\partial w_i}\left(y - h_w(x)\right)^2 \\ &= 2\left(y - h_w(x)\right)\cdot\frac{\partial}{\partial w_i}\left(y - h_w(x)\right) \\ &= 2\left(y - h_w(x)\right)\cdot\frac{\partial}{\partial w_i}\left(y - \left(w_1 x + w_0\right)\right)\end{aligned}$$

(here we show a simple linear case for $h_w(x)$)

which yields:

$$\begin{aligned}\frac{\partial}{\partial w_0} Loss(\vec{w}) &= 2\left(y - h_w(x)\right)\cdot\frac{\partial}{\partial w_0}\left(y - \left(w_1 x + w_0\right)\right) \\ &= -2\left(y - h_w(x)\right)\end{aligned}$$

$$\begin{aligned}\frac{\partial}{\partial w_1} Loss(\vec{w}) &= 2\left(y - h_w(x)\right)\cdot\frac{\partial}{\partial w_1}\left(y - \left(w_1 x + w_0\right)\right) \\ &= -2\left(y - h_w(x)\right)x\end{aligned}$$

Beyond linear models

- Remember the gradient descent update rule

$$w_i = w_i - \alpha \frac{\partial}{\partial w_i} Loss(w)$$

- The update rules for the previous example are:

$$\begin{aligned} w_0 &= w_0 - \alpha \frac{\partial}{\partial w_0} Loss(\vec{w}) \\ &= w_0 + \alpha 2\left(y - h_w(x)\right) \\ &= w_0 + \alpha^*\left(y - h_w(x)\right) \end{aligned} \qquad \begin{aligned} w_1 &= w_1 - \alpha \frac{\partial}{\partial w_1} Loss(\vec{w}) \\ &= w_1 + \alpha 2\left(y - h_w(x)\right)x \\ &= w_1 + \alpha^*\left(y - h_w(x)\right)x \end{aligned}$$

we fold the 2 into α^*

- To minimize the sum of losses across (x_j, y_j):

$$w_0 = w_0 + \alpha^* \sum_{j=1}^{N} \left(y_j - h_w(x_j)\right) \qquad w_1 = w_1 + \alpha^* \sum_{j=1}^{N} \left(y_j - h_w(x_j)\right) x_j$$

while $\alpha^* = 2\alpha$, we will frequently just write α

Gradient descent types

- Batch (offline) gradient descent
  - w's update based on all samples
  - convergence guaranteed with small α
  - may be slow

$$w_i = w_i - \alpha \sum_{j=1}^{N} \frac{\partial}{\partial w_i} Loss(w, y_j, x_j)$$

- Stochastic gradient descent
  - update **w** after every example (good for online learning)
  - usually faster
  - No guarantee of convergence with fixed α (may oscillate about a minimum)

$$w_i = w_i - \alpha \frac{\partial}{\partial w_i} Loss(w, y_j, x_j)$$

mini batch – hybrid between two is popular!

Multivariate linear regression

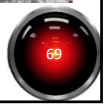
- Observations are multidimensional:
- Weight vector has a component for each x_{ji}:

$$h_{sw}(\vec{x}_j) = w_0 + w_1 x_{j,1} + w_2 x_{j,2} + \cdots + w_D x_{j,D} = w_0 + \sum_{i=1}^{D} w_i x_{j,i}$$

- We frequently eliminate w_0's special case:

Redefine $\vec{x}_j$ as $[1, x_{j,1}, x_{j,2}, \ldots, x_{j,D}]$, then

$$h_{sw}(\vec{x}_j) = \sum_{i=0}^{D} w_i x_{j,i}$$
$$= \vec{w}^T \vec{x}_j \text{ (product of } 1 \times D \text{ and } D \times 1 \text{ vectors)}$$



Multivariate linear regression

- As before, we minimize the L2 loss.
- Gradient descent is very similar to the univariate case:

$$w_i = w_i + \alpha \sum_{j=1}^{N} (y_j - h_{sw}(x_j)) x_{j,i}$$

- However for linear functions we can solve analytically for a closed-form equation that does not require iteration

$$w^* = (X^T X)^{-1} X^T y$$

$$X = \begin{bmatrix} 1 & x_{1,1} & x_{1,2} & \ldots & x_{1,D} \\ 1 & x_{2,1} & x_{2,2} & \ldots & x_{2,D} \\ 1 & & \ldots & & \\ 1 & x_{N,1} & x_{N,2} & \ldots & x_{N,D} \end{bmatrix} \quad y = \begin{bmatrix} y_1 \\ y_2 \\ \ldots \\ y_N \end{bmatrix}$$

Overfit regressions

- Not a problem for univariate linear regression
- Problematic for multivariate
- Regularization provides penalties for increasing complexity

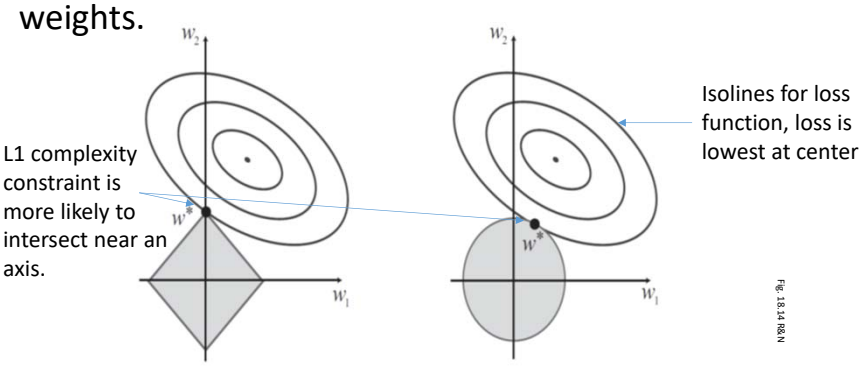
$$Cost(h_w) = EmpLoss(h) + \lambda Complexity(h)$$

$$Complexity(h_w) = L_q(w) = \sum_i |(w_i)|^q$$

we regularize by picking the minimal cost hypothesis.

Regularization of regression

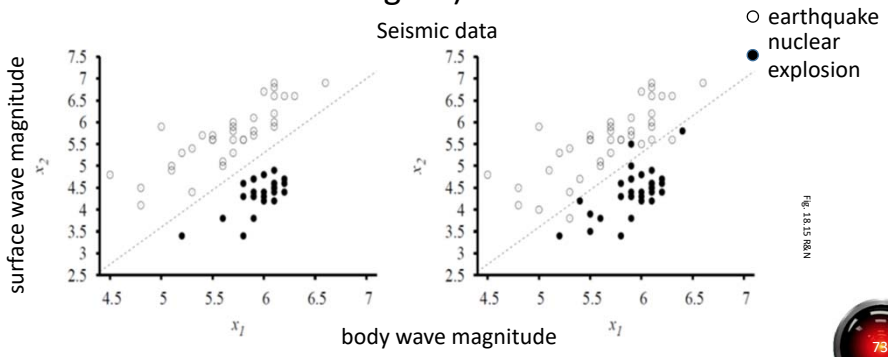
L_1 tends to produce sparse models with many zero weights.



- Minimizing Cost is equivalent to minimizing loss with constraint that complexity ≤ some constant.
- Complexity increases as w* moves away from the origin

Linear classification

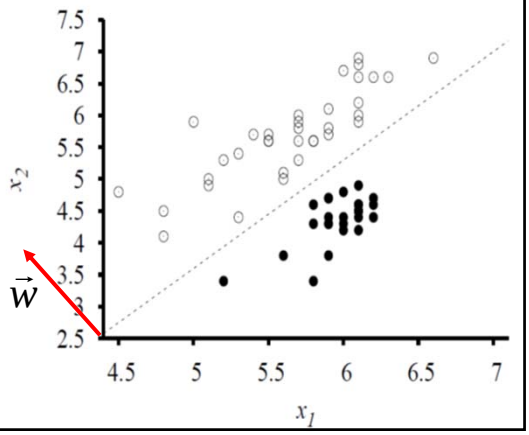
- Regression lines can be used to classify examples
- We look for a linear separating line (hyperplane when data $\in \Re^3$ or higher)



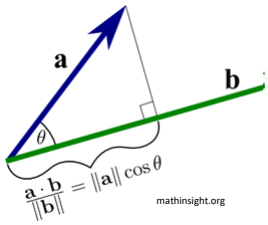
Linear classification

- Suppose we pick a vector *w* perpendicular to the decision boundary
- Consider the dot product between w and an arbitrary point

$$\vec{w} \cdot \vec{x} = \sum_{j=1}^{D} w_j x_j$$

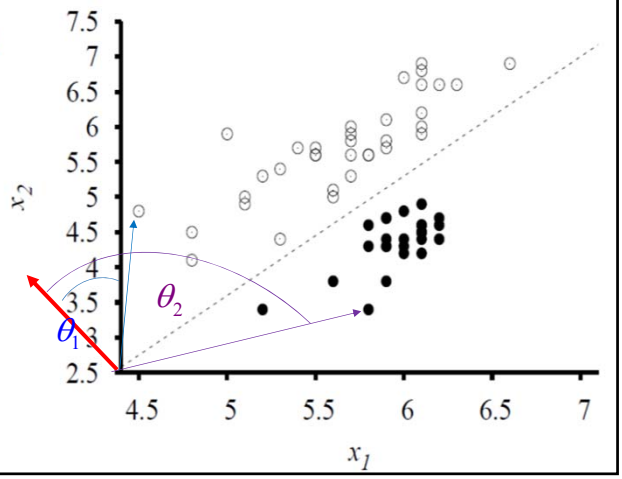


Geometric interpretation of the dot product



$\theta_1 < 90° \Rightarrow \cos(\theta_1) > 0$

$\theta_2 > 90° \Rightarrow \cos(\theta_2) < 0$



Linear classification

- Once we have **w**, we can classify by taking the dot product and looking at the sign.

$$h_w(x) = Threshold(w \cdot x)$$

$$Threshold(z) = \begin{cases} 1 & z \geq 0 \\ -1 & z < 0 \end{cases}$$

- How do we choose **w**?
  - Fisher provides a method to select without iterating, but it won't be useful for later applications…
  - It turns out a variant of the gradient descent rule used for regression is applicable.

  Recall the regression rule $w_i = w_i + \alpha(y - h_w(x))x_i$

Linear classification – Perceptron learning rule

- Unlike the regression rule, we are not trying to minimize squared error, but loss.
- We only apply the update when we would have gotten this wrong

```
def perceptron-learning(w,x,y):
  prediction = threshold(w·x)
  if y != prediction:
    # Oops! Need w·x larger if y =1.
    #  Increase w when x positive,
    #  Decrease when x negative
    # Opposite when w·x = -1
    w = w + αyx
```

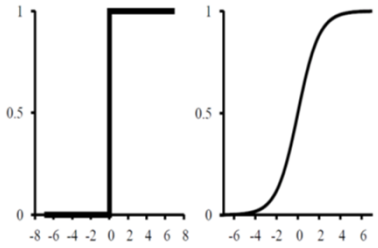
Perceptron learning rule

- Iterate through data (1 iteration = 1 epoch) and repeat until no errors
- Convergence may be slow, but guaranteed for *linearly separable data*.

- Most data sets are not linearly separable.
- When non linearly separable examples are presented in random order, we will converge to a stable classifier if α decays linearly with epoch number.

Linear classifier with logistic regression function

- Hard vs. soft thresholds



$$h_w(x) = Logistisic(w \cdot x) = \frac{1}{1+e^{-w \cdot x}}$$

- Has nicer properties (e.g. differentiable)

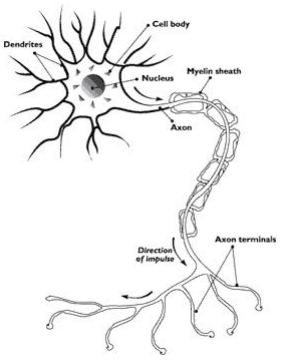
Learning with logistic regression

- Remember that our update rule was: $w_i = w_i - \alpha \frac{\partial}{\partial w_i} Loss(w)$
- When the hard threshold is replaced with logistic regression:
  - It is easier if we use labels [0,1] instead of [-1,1] as this is the range covered by the function
  - The derivative of the loss needs to be recomputed (see text for derivation)

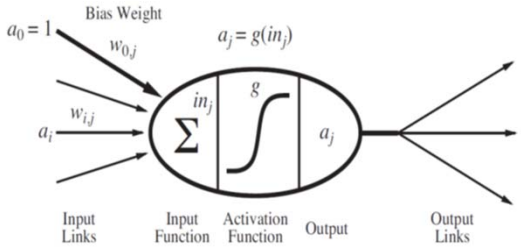
$$w_i = w_i - \alpha(y - h_w(x))h_w(x)(1 - h_w(x))x_i$$

  - This results in a smoother and more predictable learning curve

Connectionist networks (artificial neural networks)



Neuron
National Institute on Drug Abuse

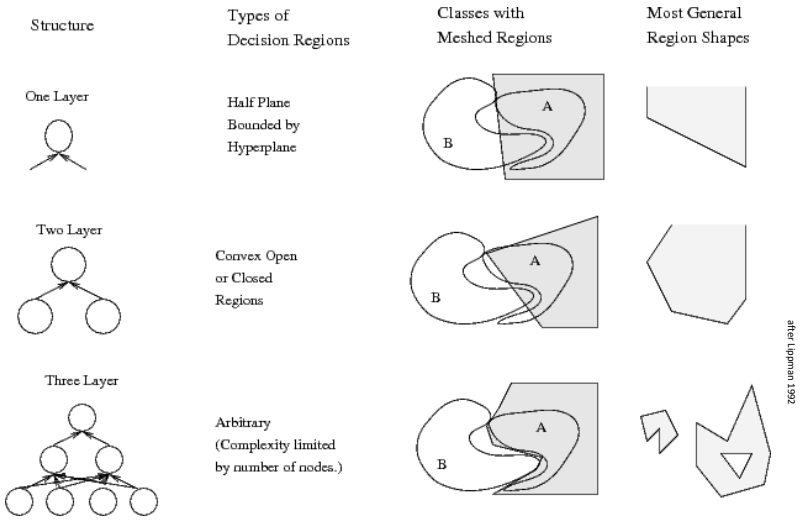


Model of a neuron
Fig. 18.19 R&N

Connectionist networks

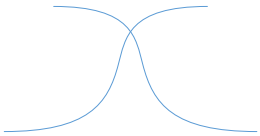
- Activations functions for perceptrons are nonlinear:
  - hard threshold
  - logistic regression (frequently called sigmoid function)
- Linking perceptrons together provides complex function modeling capability

Decision boundary capability as a function of network depth

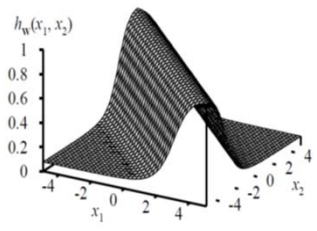


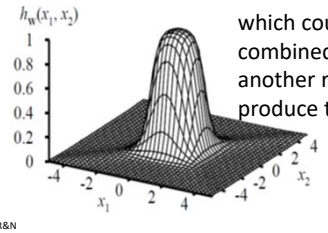
An intuitive view of neural nets

- Suppose we combine two perceptrons whose output functions are reversed



- This could be used to model a ridge in output space



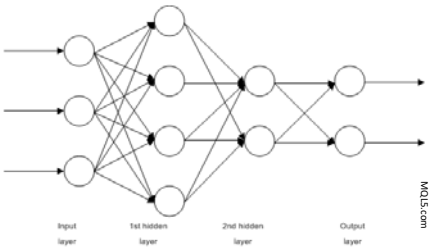


which could be combined with another ridge to produce this

Fig. 18.23 R&N

Learning in a neural network

- Consider input vector **x**
- Output vector **a**

$\vec{x}$  $\vec{a}$

Multilayer Feed-forward network

Learning in a neural network

- Similar to the regression problem, for output **a** and desired output **y**, we can find the loss gradient for each output node

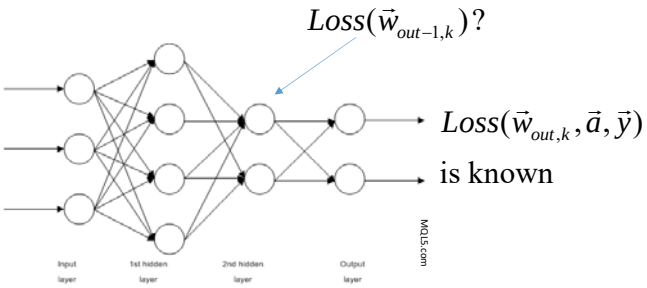
$$\frac{\partial}{\partial w} Loss(w) = \frac{\partial}{\partial w}\left|y - h_w(x)\right|^2 = \frac{\partial}{\partial w}\sum_{k=1}^{D}(y_k - a_k)^2 = \sum_{k=1}^{D}\frac{\partial}{\partial w}(y_k - a_k)^2$$

$$a_k = \frac{1}{1+e^{-w_{output,k}\cdot input}}$$

and use the perceptron learning rule for the sum of the gradients at the output layer.

Back-propagation

- What should the targets be for the previous input layer?



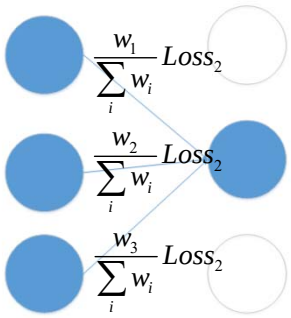
Multilayer Feed-forward network

Back propagating error (overview)

- Error of the k[th] output: $Err_k = y_k - a_k$
- We can compute the gradient for any input node (in) and apply the regression rule.
- This gives us a new set of weights for the output node.

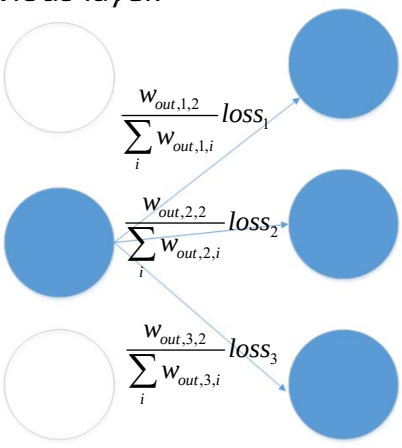
Back propagating error (overview)

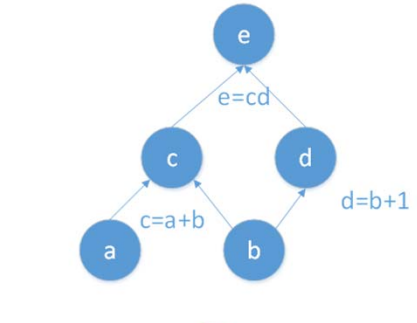
- After applying the update to the output layer, there still exists loss
- We assign a portion of the loss to each of the input nodes based on their weight.
- This contribution is computed for each node of the current layer

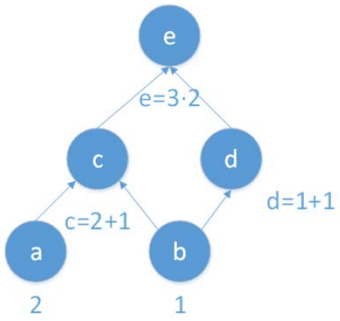


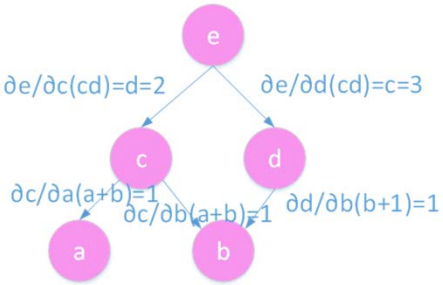
Back propagating error (overview)

- Now we can look at the sum of losses attributable to each node in the previous layer.
- The sum of these provides us with a loss to minimize.
- Repeat recursively









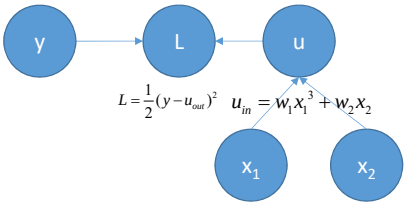
$$\frac{\partial e}{\partial b}=\frac{\partial e}{\partial c}\frac{\partial c}{\partial b}+\frac{\partial e}{\partial d}\frac{\partial d}{\partial b}=2\cdot 1+3\cdot 1=5$$

Concrete example

Example based on Christopher Olah's blog post

Activation fn example for backprop

$u_{out}=\sigma(u_{in})$



partial derivatives

$$\frac{\partial L}{\partial u_{out}}=\frac{2}{2}(y-u_{out})(-1)=u_{out}-y$$

$$\frac{\partial u_{out}}{\partial u_{in}}=\sigma(u_{in})(1-\sigma(u_{in}))$$

$$\frac{\partial u_{in}}{\partial u_{x_1}}=w_1 3x_1^2$$

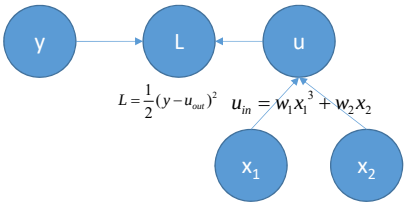
$$\frac{\partial u_{in}}{\partial u_{x_2}}=w_2$$

$$\frac{\partial u_{in}}{\partial u_{w_1}}=x_1^3$$

$$\frac{\partial u_{in}}{\partial u_{w_2}}=x_2$$

Activation fn example for backprop

$u_{out} = \sigma(u_{in})$



To update w_1 we use the chain rule:

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial u_{out}} \frac{\partial u_{out}}{\partial u_{in}} \frac{\partial u_{in}}{\partial u_{w_1}} = (u_{out} - y)\sigma(u_{in})(1 - \sigma(u_{in}))x_1^3$$

from previous slide

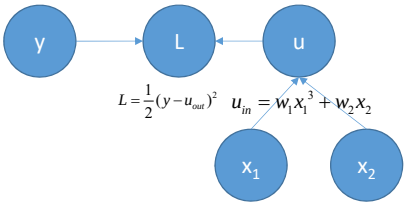
$$\frac{\partial L}{\partial u_{out}} = u_{out} - y$$

$$\frac{\partial u_{out}}{\partial u_{in}} = \sigma(u_{in})(1 - \sigma(u_{in}))$$

$$\frac{\partial u_{in}}{\partial u_{w_1}} = x_1^3$$

Activation fn example for backprop

$u_{out} = \sigma(u_{in})$



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial u_{out}} \frac{\partial u_{out}}{\partial u_{in}} \frac{\partial u_{in}}{\partial u_{w_1}} = (u_{out} - y)\sigma(u_{in})(1 - \sigma(u_{in}))x_1^3$$

Concrete example

$$y = 0, w = \begin{bmatrix} .02 \\ .01 \end{bmatrix}, x = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

implies

$$u_{in} = w_1 x^3 + w_2 x_2 = .02 \cdot 3^3 + .01 \cdot 5 = .59$$

$$u_{out} = \frac{1}{1 + e^{-u_{in}}} = \frac{1}{1 + e^{-.23}} = .6434$$

$$L = \frac{1}{2}(y - u_{out})^2 = .5 \cdot (0 - .6434)^2 = .2070$$

$$\frac{\partial L}{\partial u_{out}} = y - u_{out} = .6434 - 0 = .6434$$

$$\frac{\partial u_{out}}{\partial u_{in}} = \sigma(u_{in})(1 - \sigma(u_{in})) = \sigma(.59)(1 - \sigma(.59))$$

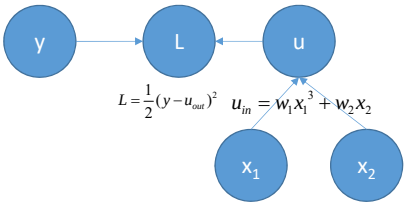
$$= \frac{1}{1 + e^{-.59}}\left(1 - \frac{1}{1 + e^{-.59}}\right) = .6434(1 - .6434) = .2294$$

$$\frac{\partial u_{in}}{\partial u_{w_1}} = x_1^3 = 3^3 = 27$$

$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial u_{out}} \frac{\partial u_{out}}{\partial u_{in}} \frac{\partial u_{in}}{\partial u_{w_1}} = .6434 \cdot .2294 \cdot 27 = 3.9851$$

Activation fn example for backprop

$u_{out} = \sigma(u_{in})$



$$\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial u_{out}} \frac{\partial u_{out}}{\partial u_{in}} \frac{\partial u_{in}}{\partial u_{w_1}} = (u_{out} - y)\sigma(u_{in})(1 - \sigma(u_{in}))x_1^3$$

Suppose we have a learning rate ε=.01

$$w_1 = w_1 - \varepsilon \frac{\partial L}{\partial w_1} = .02 - .01 \cdot 3.9851 = -0.0599$$

Update of w_2 is left as an exercise, but loss with only w_1 changed:

$$y = 0, w = \begin{bmatrix} -.06 \\ .01 \end{bmatrix}, x = \begin{bmatrix} 3 \\ 5 \end{bmatrix}$$

implies

$$u_{in} = w_1 x^3 + w_2 x_2 = -.06 \cdot 3^3 + .01 \cdot 5 = -1.57$$

$$u_{out} = \frac{1}{1 + e^{-u_{in}}} = \frac{1}{1 + e^{-(-1.57)}} = .1722$$

$$L = \frac{1}{2}(y - u_{out})^2 = .5 \cdot (0 - .1722)^2 = .0148 \quad \text{old L=.2070}$$

Neural net summary

- Supervised learner
  - Training labels either
    - High value for class (n classes → n output nodes)
    - Encoding of class information
  - Iterative training typically using a gradient descent algorithm (e.g. back propagation)
- Classification
  - Present features to input nodes
  - Interpret output nodes for category

Neural net summary

- Disadvantages
  - frequently hard to interpret
  - Many parameters require large data sets
  - Doesn't do well with imbalanced examples
  - Slow to train
  - Overfits easily and regularization is important
- Advantages
  - Flexible, nonlinear learner
  - Deep architectures are very powerful

Non-parametric models

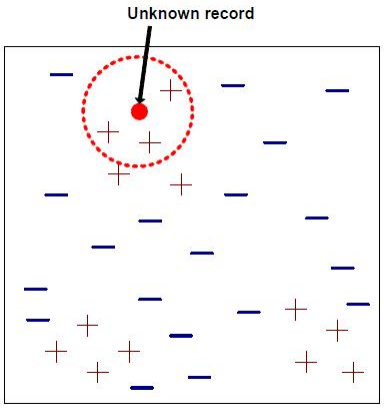
- Neural nets and decision trees have models with parameters
  - decision node parameters: attribute and cut-point/categories for sub-trees
  - neural nets: weights and connections
- Non-parametric models
  - Cannot be characterized by a *bounded* set of parameters
  - Simplest case:
    Look at every example and use it to classify a novel example. (Parameters $\propto$ #training examples)
  - Called instance- or memory-based learning

Nearest neighbor models

- Use a distance metric to find the k closest neighbors, e.g. for continuous attributes:

$$L^P(\vec{x}_j, \vec{x}_q) = \left(\sum_{i=1}^{D} |x_{j,i} - x_{q,i}|^p \right)^{\frac{1}{p}}$$

- Use the plurality of labels that are the k closest



Nearest neighbor models

- The good
  - Simplicity
  - Effective technique for low-dimensional data
- The Bad – Searching is expensive with large training sets, but we can mitigate for this:
  - trees – Similar to a decision tree (split on value, may at times need to search both sides)
  - Locally sensitive hash tables
    - Hash functions
      - set of projections on to lines (similar to linear classification examples)
      - Line projections are discretized into buckets
    - Can be much more effective than tree approach

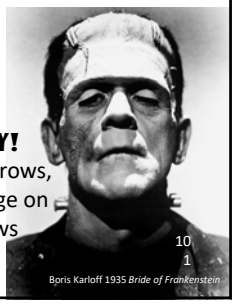
Nearest neighbor models

- and the Ugly
  - N points uniformly distributed in an $\Re^D$ unit hypercube.
  - To capture r=.01 of the observations, what edge length *l* would we need in a random sample?
  - Samples are randomly distributed and total volume is 1, so we need a volume of r (.01).

$$l^d = r \rightarrow l = r^{1/D}$$

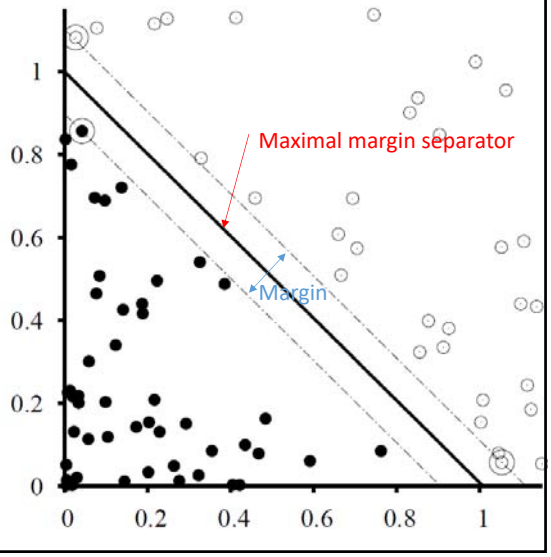
- $d = 1 \rightarrow l = .01^{1/D} = .01$
- $d = 10 \rightarrow l = .01^{1/10} = .63$
- $d = 100 \rightarrow l = .01^{1/100} = .96$

THE CURSE OF DIMENSIONALITY!
As the dimension grows, the size of each edge on the hypercube grows as well!



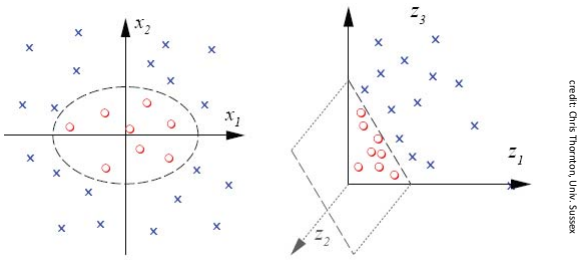
Support vector machines (SVMs)

- A margin is the distance to the closest examples on either side of a hyperplane.
- SVM approaches attempt to maximize the margin



Support vector machines

- Can only separate linear problems, but a kernel function can project the data into a higher dimensional space where perhaps the data can be better separated



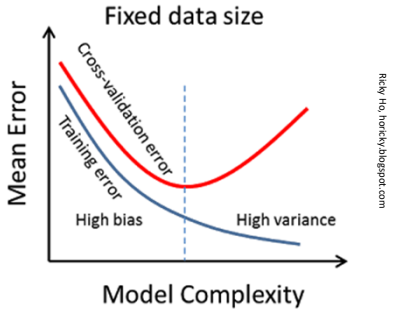
credit: Chris Thornton, Univ. Sussex

Support vector machines

- Maximal margins com are computed as functions of training examples
- Consequently
  - SVMs are nonparametric techniques
  - In practice, only a small subset of training examples, the support vectors, are required
- The training algorithm is beyond our scope, but is essentially an optimization problem.

Bias and variance

- Error in learning comes from two sources: bias and variance



Bias – Large when learners make consistently incorrect predictions
Variance – Large when different training sets result in different predictions

Ensemble learning

- Ensemble learners frequently are a collection of weak learners that are combined to form a robust classifier

  Weak learner – A simple learning algorithm that is likely to have a high bias (e.g. a single node, or stump, of a decision tree)

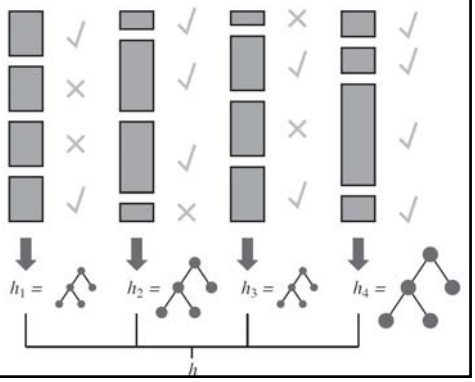
- Ensemble learners typically use collections of weak classifiers to reduce both bias and variance.

Adaptive Boosting (ADABOOST)

- Type of ensemble learning algorithm
- Use decision stumps as the weak learner
- Examples are weighted. Loss is greater for examples with higher weight

Adaptive boosting

- Start with uniform weights
- Learn the decision tree stump
  - Redistribute weights: misclassified training examples get more weight
  - Produce a classification weight as a function of error
  - Iterate until k learners are produced



Adaptive boosting

- Classification
  - Classify an example by each of the k weak learners
  - Use plurality of *weighted* decisions

- A very interesting tidbit…
  Letting k grow beyond the point of having all training examples lets ADABOOST frequently continue to improve generalization scores.

  Some interpret this as ADABOOST being robust to overtraining.