Supervised Learning: Regression & Classifiers

Fall 2010

Problem Statement

- Given training data of the form:

$$\{(x^i, y^i)\ldots(x^m, y^m)\}$$

- X: the space of input features/attributes
- Y: the space of output values (target variable)
- A pair (x^i, y^i) is called a training example (the superscript "(i)" is just the instance number in the training set).
- We want to learn a function h : X → Y that is a "good" predictor for the corresponding value of y.

Notations

$$\{(x_1^{\ 1}, x_2^{\ 1}, \ldots, x_n^{\ 1}, y^1)\}$$

$$\ldots$$

$$\{(x_1^{\ m}, x_2^{\ m}, \ldots, x_n^{\ m}, y^m\}$$

- m training examples.
- n features.
- i_{th} example.
- j_{th} coefficient.

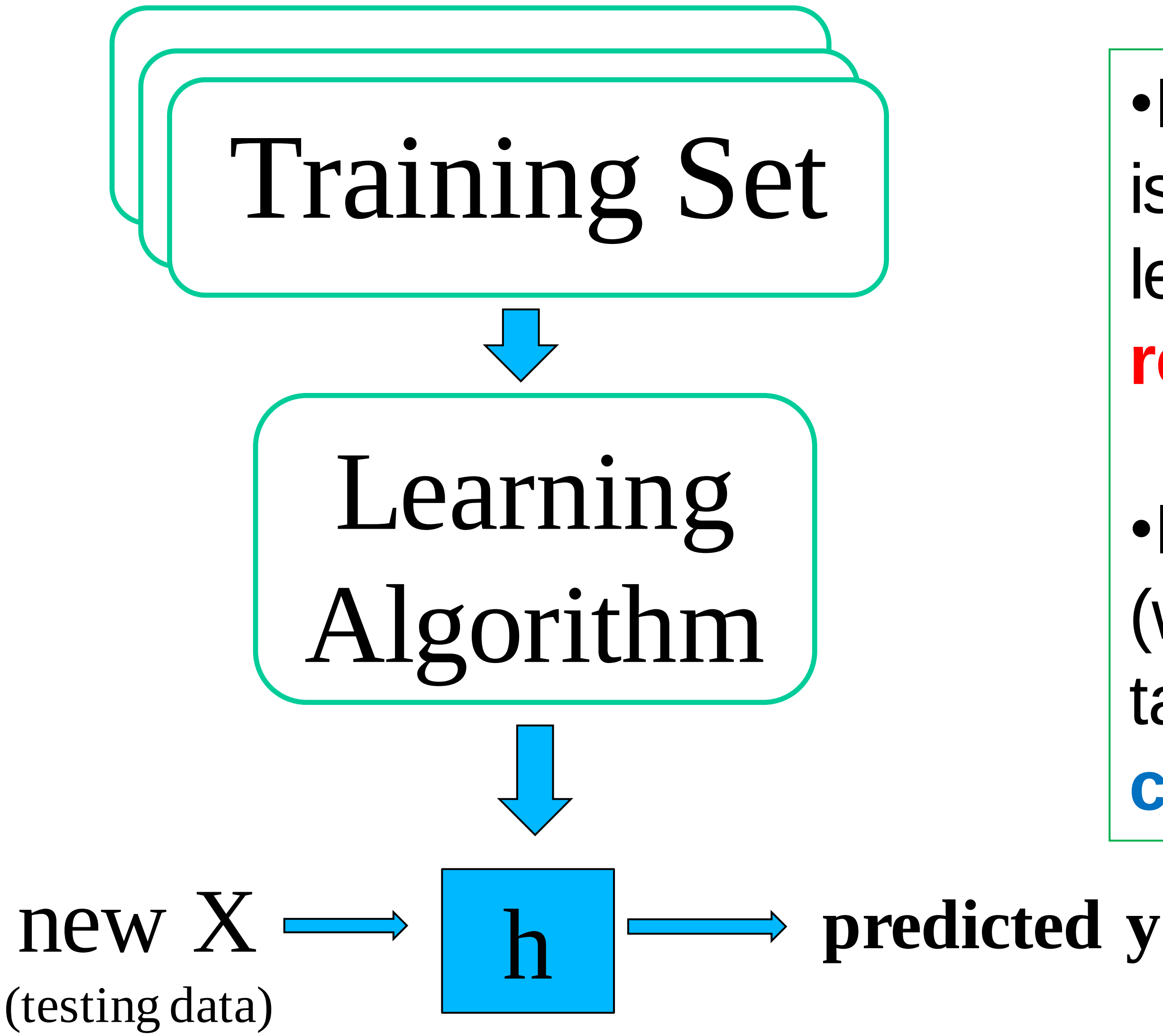
Almost always, m >> n (number of examples) (equations) is significantly larger than the number of unknown coefficients). Hence, we are looking for approximate solutions.

Applications

- Document Classification
  - e.g., spam filtering
- Speech and Face Recognition
- Loan Approval
- Medical Diagnosis

And many many more…

Supervised Learning



- If the target variable (Y) is continuous, the learning problem is a **regression** problem.

- If the target is discrete (we will focus on binary targets), it is a **classification** problem.

Linear Regression by Example

- Our training data contain 2 features and a continuous target variable. We would like to predict the prices of other houses as a function of the size of their living areas and number of bedrooms.

Living area (sq. feet)	#bedrooms	Price (1000$s)
2104	3	400
1600	3	330
2400	3	369
…		

Example from: http://www.stanford.edu/class/cs229/notes/cs229-notes1.pdf

We are looking for a linear function of the form:

$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

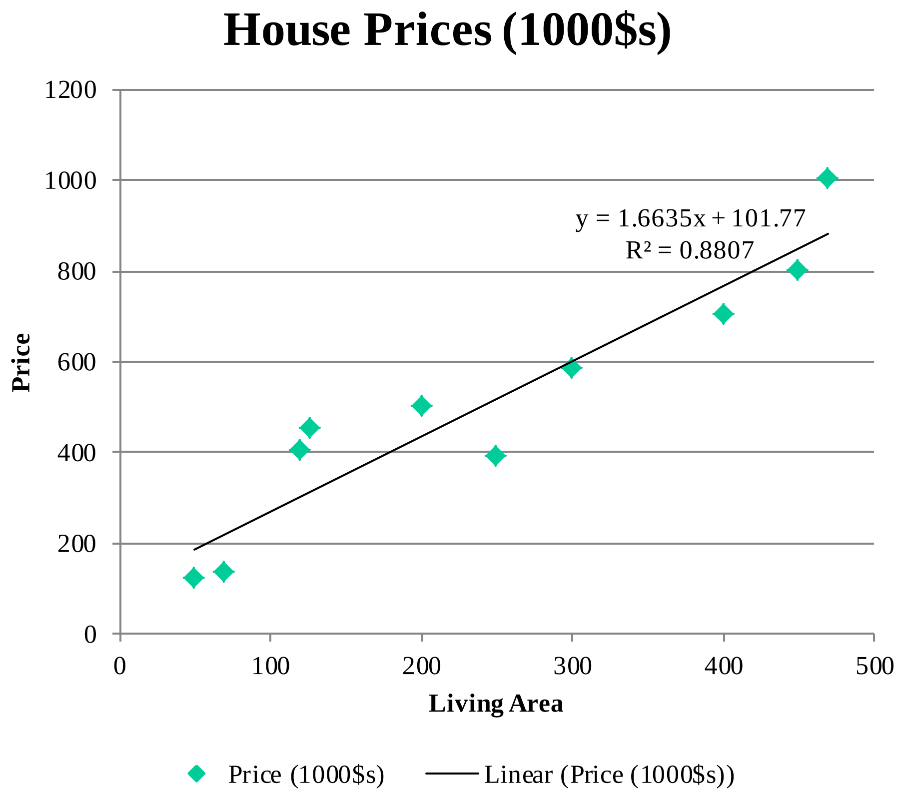
Linear Regression

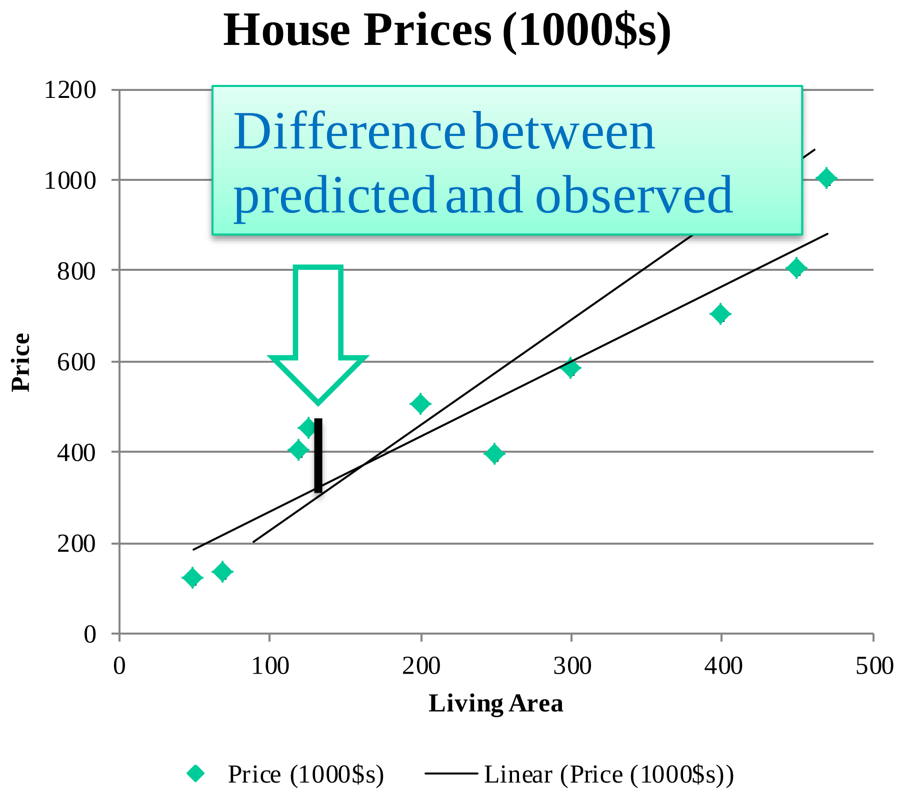
$$h_\theta(x) = \theta_0 + \theta_1 x_1 + \theta_2 x_2$$

- $\theta_i's$ are the parameters (also called weights or coefficients) we would like to learn. Once we learn the parameters, we can plug in a new “living area in sq. feet” and “#bedrooms”, and $h_\theta(x)$ would predict the price of the house.

Best Regression Fit

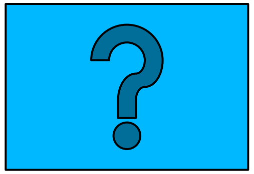
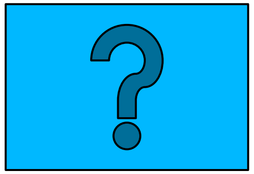
- With one input feature and one target variable, the interpretation is fitting a line to the data.





Which line fits the data better?

Finding Good Parameters

- Need to define a cost/loss function:
- 1) $J(\theta) = \sum_{i=1}^{m} (h_\theta(x^i) - y^i)$ 
  - Problem: plus/minus – a bad line can end up as the perfect fitting.
- 2) $J(\theta) = \sum_{i=1}^{m} |(h_\theta(x^i) - y^i)|$  Least Absolute Errors
  - Problem: Not very common since it's not convenient mathematically (out of the scope of this class). See the Wikipedia article for the details. http://en.wikipedia.org/wiki/Least_absolute_deviations

Least Squares

Illinois Institute of Technology

- A very common cost function:

$$J(\theta) = \frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^i) - y^i)^2$$

- This is the least square cost function.
- $(h_\theta(x^i) - y^i)$ is the error rate the learning algorithm makes based on the correct values of y in the training data. **We want to minimize the cost function** – i.e., finding the weights that would minimize it (also called finding the best fit).

Batch Gradient Descent

•The gradient descent algorithm starts with some initial θ, and repeatedly performs an update.

$$\theta_j := \theta_j - \alpha \frac{d}{d\theta_j} J(\theta)$$

Done for every jth coefficient

- α is the learning rate (how much we progress in every step)
- The algorithm repeatedly takes a step in the direction of steepest decrease of J.

For one training example we get:

$$\frac{d}{d\theta_j} J(\theta) = 2 \cdot \frac{1}{2} (h_\theta(x) - y) \frac{d}{d\theta_j} (h_\theta(x) - y) =$$

$$= (h_\theta(x) - y) \cdot \frac{d}{d\theta_j} \left(\sum_{i=0}^{n} \theta_i x_i - y\right) \quad = (h_\theta(x) - y) \cdot x_j$$

Batch Gradient Descent

Illinois Institute of Technology

- Finally we get:

Repeat until convergence:{

$$\theta_j = \theta_j + \alpha \sum_{i=1}^{m} (y^i - h_\theta(x^i)) x_j^i$$

}

For a single update, we iterate over the entire training data!

- This is repeated for every j (every feature). m is the number of instances in the training set.

BGD Example

Living Area (x1)	**#bedrooms (x2)**	**Price(1000$s) (y)**
20	3	40
16	3	33

i-th Row	θ_0	θ_1	θ_2	**Error**
Alpha=1				
1-m	1 (guess)	1 (guess)	1 (guess)	$h(X) = 1 + 1x_1 + 1x_2$
1	0	(40-24)*20	(40-24)*3	16
2	0	(33-20)*16	(33-20)*3	13
1-m	1	1+1*****(40-24)*20 +1*(33-20)*16 = 1 + 320 + 208 = **529**	1+1*****(40-24)*3 + 1*(33-20)*3 = **88**	

$$h(X) = 1 + 529x_1 + 88x_2$$

BGD Example

Living Area (x1)	#bedrooms (x2)	Price(1000$s) (y)
20	3	40
16	3	33

i-th Row	θ_0	θ_1	θ_2	Error
Alpha=0.001				
1-m	1 (guess)	1 (guess)	1 (guess)	
1	**0**	(40-24)*20	(40-24)*3	16
2	0	(33-20)*16	(33-20)*3	13
1-m	1	1+0.001*****(40-24)*20 + 0.001*(33-20)*16 = 1 + 0.32 + 0.208 = **1.528**	1+0.001*****(40-24)*3 + 0.001*(33 -20)*3 = 0.048+0.039 = **1.087**	

$$h(X) = 1 + 1.528x_1 + 1.087x_2$$

BGD Example

Living Area (x1)	**#bedrooms (x2)**	**Price(1000$s) (y)**
20	3	40
16	3	33
18	3	36

New →

Iteration #	θ_1	θ_1	θ_2
0	1	1	1
1	1	1.528	1.087
2	1	1.70639..	1.04014..
3	1	1.76666..	1.04362..
4	1	1.78702..	1.04483..

Guess

$$h(X) = 1 + 18 + 3 = 22$$

$$h(X) = 1 + 1.528 \cdot 18 + 1.087 \cdot 3 = 31.76$$

$$h(X) = 1 + 1.7 \cdot 18 + 1.04 \cdot 3 = 34.72$$

$$h(X) = 1 + 1.76 \cdot 18 + 1.043 \cdot 3 = 35.8$$

$$h(X) = 1 + 1.78 \cdot 18 + 1.044 \cdot 3 = 36.17$$

Stochastic Gradient Descent

Illinois Institute of Technology

- The previous algorithm is called batch gradient descent since it looks at every example in the entire training set on every step.

- Alternatively, stochastic gradient descent iterates over the training set, and for each example it updates the parameters according to that specific example only. **(often much faster – good for very large datasets)**

Loop{
  for i=1 to m, {

$$\theta_j = \theta_j + \alpha(y^i - h_\theta(x^i))x_j^{\,i}$$

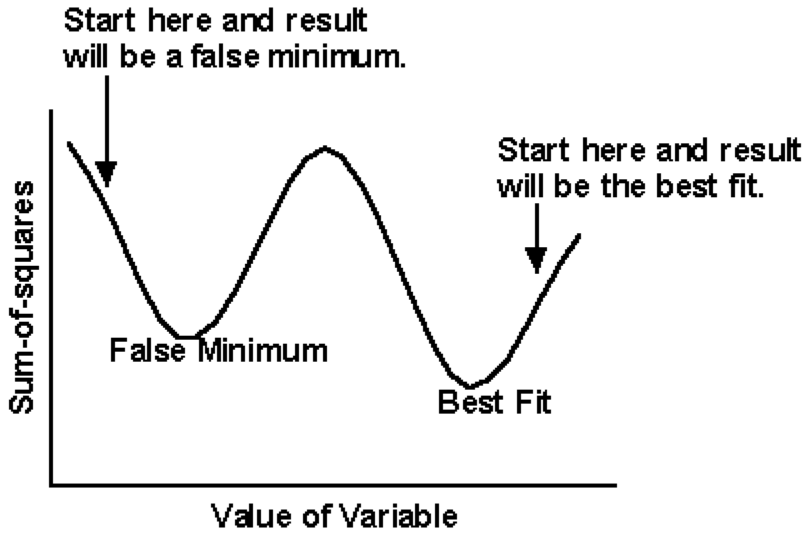
  }
}

$\sum_{i=1}^{m}$ Removed!

Global Minimum

Illinois Institute of Technology

- In the ideal case, the GD algorithm stops in a global minimum. However, this is not always the case – it depends on the function and initial guess.



http://www.graphpad.com/curvefit/2549a220.gif

- **Practical solution**: start with many different guesses and choose the one the produces the minimum (you should expect the global minimum to be produced by many guesses).

Alternatives to Least Squares

Illinois Institute of Technology

- Least squares can be sensitive to outliers, especially in noisy data. A common alternative is the **Locally Weighted Least Squares**:
  - Fitting is done “online”.
  - Slower than the “offline” ordinary LS.

$$J(\theta)=\frac{1}{2}\sum_{i=1}^{m}(h_\theta(x^i)-y^i)^2 w^i(x)$$

$w^i(x)$: We would like to choose this error function to be larger when x^i is close to x

Locally Weighted Least Squares

- With the ordinary LS, we build one general model for the data. After finding $h_\theta(x)$, we do not need the training data anymore.
- With LWLS, we build a **new model** for **each new input** we want to predict a y value to. The idea is to work on local parts.

$w^i(x)$: Defines how we choose neighbors. For example:

1) $\dfrac{1}{|x - x^i| + 1}$

2) $\exp\left(-\dfrac{(x^i - x)^2}{2\sigma^2}\right)$

if x^i and x are close: W → 1
if x^i and x are far apart: W → 0

Controls the # of neighbors

Logistic Regression

Illinois Institute of Technology

- In LR we predict the likelihood that Y is equal to 1 given certain values of X (which also gives us the probability that Y is equal to 0).

Age	Gender	Height	Play Basketball?
22	M	1.85m	Yes
24	M	1.92m	Yes
60	F	1.66m	No
…	…	…	…
19	M	1.80	?

Logistic Regression

Illinois Institute of Technology

In classification problems we replace the linear regression with logistic regression to achieve a better learning algorithm. We only change the form of the function h as follows:

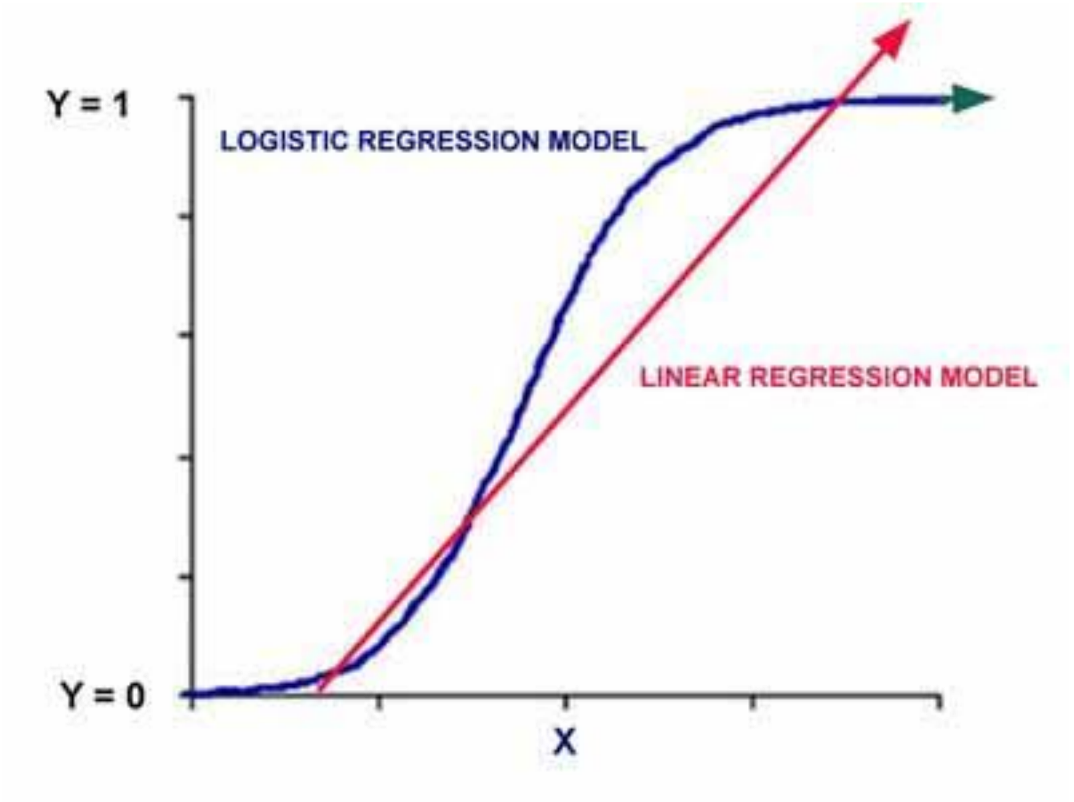
$$\theta^T x = \theta_0 + \sum_{j=1}^{n} \theta_j x_j$$

$$h(x) = \frac{1}{1 + e^{-\theta^T x}}$$

The Logistic Function

To find the best fit, we can use the same SGD algorithm we used for linear regression. The only difference is that the function h is different now – outputs a number between 0 and 1.

Logistic Regression Function



Taken from http://faculty.chass.ncsu.edu/garson/PA765/logistic.htm

Maximum Likelihood Estimation

- MLE is a common method for estimating model parameters. In our case, MLE is used to find the **coefficients that make the observed data as probable as possible.**
- Our hypothesis function h has a different meaning now: we are trying to estimate the probability that Y is equal to 1 given X.

$$P(y=1 | X;\theta) = h_\theta(x)$$

$$P(y=0 | X;\theta) = 1 - h_\theta(x)$$

- For convenience, we write the above two as follows: (this is identical)

$$P(y | X;\theta) = h_\theta(x)^y (1 - h_\theta(x))^{1-y}$$

Maximum Likelihood Estimation

Likelihood of the data

$$L(\theta) = P(Y \mid X;\theta) = \prod_i P(y^i \mid x^i;\theta)$$

assuming iid

$$= \prod_i h_\theta(x^i)^{y^i}(1 - h_\theta(x^i))^{1-y^i}$$

- Next, for convenience, we maximize the logarithm of the likelihood instead of the likelihood itself (we get rid of the product and have summation instead). We can do so because the logarithm function is monotonically increasing and we do not care about the value itself, just the parameters that maximize the whole thing.

Maximum Likelihood Estimation

- We again use the Gradient Descent algorithm, but since we want to maximize the function instead of minimizing it, we have the following: (gradient ascent)

$$\theta_j := \theta_j + \alpha \frac{d}{d\theta_j} l(\theta)$$

+ instead of -

The log likelihood that we want to maximize (not the same function as before).

- Although we started with a different model, we end up with the same learning algorithm as before (but h now is different):

$$\theta_j = \theta_j + \alpha(y^i - h_\theta(x^i))x_j{}^i$$

The same SGD!

Mahout LR Implementation

TrainLogistic has a main function that can be run to do a logistic regression

```
org.apache.mahout.classifier.sgd.TrainLogistic
  --input in.csv --output out
  --passes <input passes> --rate <learning rate>
  --features <number of target feature>
  --target <target variable>
  --categories <number target categories possible>
  --predictors <predictor variables>
  --types <predictor types (numeric, word, or text)>
```

Mahout LR Example

Illinois Institute of Technology

- TrainLogistic performs *passes* passes on the input csv and outputs the results to *output*
- Input data must be transformed to a csv and represented as numeric or text attributes
- Tennis data with target class of whether to play

Outlook	Temperature	Humidity	Windy	Play
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

Outlook	Temperature	Humidity	Windy	Play
0	2	1	0	0
0	2	1	1	0
1	2	1	0	1
2	1	1	0	1
2	0	0	0	1
2	0	0	1	0
1	0	0	1	1
0	1	1	0	0
0	0	0	0	1
2	1	0	0	1
0	1	0	1	1
1	1	1	1	1
1	2	0	0	1
2	1	1	1	0

Mahout LR Example

- Run a LR on tennis data
- Two categories: play and don’t play
- Other attributes are the predictors

```
org.apache.mahout.classifier.sgd.TrainLogistic
  --input tennis.csv --output out
  --passes 100 --rate 50
  --features 4
  --categories 2
  --predictors outlook temperature humidity windy
  --types numeric
  --target play
```

Mahout LR Example – Output

- Mahout produces a model file and text output
- The model contains similar information and a copy of TrainLogistic's runtime parameters in the JSON format

4 *// number of features*

play ~ 2.510*Intercept Term + -0.601*outlook + -0.627*temperature + -0.601*humidity + -0.601*windy

Intercept Term 2.50953

humidity -0.60090

outlook -0.60090

temperature -0.62739

windy -0.60090

-0.627386850 2.509528194 0.000000000 -0.600897574

Mahout LR Example – Classification

- We can check our results by plugging them in
- $h(x) = \frac{1}{1+e^{\sum_{i=0}^{n} -\theta_i x_i}}$
- $h(x) = \frac{1}{1+e^{-(\theta_0+\theta_1 x_1+\theta_2 x_2+\theta_3 x_3+\theta_4 x_4)}}$
- $\theta_i = weight\ for\ the\ i\ th\ feature$
- $x_i = value\ of\ the\ i\ th\ feature\ in\ the\ instance\ being\ classified$
- We classify the record as “don’t play” or “play” based on whether h(x) is closer to 0 or 1
- Plug in values for the second record
  - $h(x) = 0.16$
  - The record is closer to 0, so we classify it as “don’t play”
  - The record’s actual label is “don’t play,” so our classification was correct

Mahout LR Code

```
public double classifyScalar(Vector instance) {
  if (numCategories() != 2) {
    throw new IllegalArgumentException("Can only call
                                        classifyScalar with two categories");
  }

  // apply pending regularization to whichever coefficients matter
  regularize(instance);

  // result is a vector with one element so just use dot product
  double r = Math.exp(beta.getRow(0).dot(instance));
  return r / (1 + r);
}
```

Naïve Bayesian Classifier

- Classifier based on Bayes Theorem.
- Combines the impact/probability of each feature on the class label.
- Naïve: assumes the independence between the features.
  - Shape and color of a fruit determining the fruit
  - Education and salary determining the life style (independence??)

Naïve Bayesian Classifier

Given a hypothesis, calculating the probability of correctness of that hypothesis.

Hypothesis: x_1 , x_2 is a Peach.

Calculate the probability that x_1 , x_2 is a Peach.

P(H: x_1 , x_2 is a Peach)

P(H: x_1 , x_2 is an Apricot)

………

1. Calculate each of these probabilities.
2. Choose the highest probability.

Bayes Theorem

P(H|X) Posterior Probability of hypothesis H

X : x_1 , x_2 , …, x_n

Shows the confidence/probability that suppose X, then the hypothesis is true.

x_1 : shape = round, x_2 : color = orange

H: x_1 , x_2 is a Peach.

P(H) Prior Probability of hypothesis H

Probability that regardless of data the hypothesis is true.

Regardless of color and shape, it is a Peach.

Bayes Theorem

P(X|H) Posterior Probability of X conditioned on hypothesis H

- Given H is true (X is a Peach), calculate probability that X is round and orange.

P(X) Prior Probability of X

- Probability that sample is round and orange.

Bayes Theorem

$$\underbrace{P=(H|X)}_{\text{Posterior Probability of class } C_i} = \frac{\overbrace{P(X|H)}^{\text{Posterior Probability of X}}\;\overbrace{P(H)}^{\text{Prior Probability of class } C_i}}{\underbrace{P(X)}_{\text{Prior Probability of X}}}$$

Naïve Bayesian Classifier

- Hypothesis H is the class C_i.
- $P(X)$ can be ignored as it is constant for all classes.
- Assuming the independence assumption, $P(X|C_i)$ is:

$$P(X \mid C_i) = \prod_{k=1}^{n} P(x_k \mid C_i)$$

- Thus:

$$P(C_i \mid X) = P(C_i) \prod_{k=1}^{n} P(x_k \mid C_i)$$

- $P(C_i)$ is the ratio of total samples in class C_i to all samples.

Naïve Bayesian Classifier

- For Categorical attribute:

  $P(x_k|C_i)$ is the frequency of samples having value x_k in class C_i.

- For Continuous (numeric) attribute:

  $P(x_k|C_i)$ is calculated via a Gaussian density function.

Naïve Bayesian Classifier

- Having pre-calculated all $P(x_k | C_i)$, to classify an unknown sample X:
  - » Step 1: For all classes calculate $P(C_i | X)$.
  - » Step 2: Assign sample X to the class with the highest $P(C_i | X)$.

Play-tennis example: estimating $P(x_i|C)$

(Example from: Tom Mitchell "Machine Learning")

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

P(p) = 9/14
P(n) = 5/14

outlook	
P(sunny|p) = 2/9	**P(sunny|n) = 3/5**
P(overcast|p) = 4/9	**P(overcast|n) = 0**
P(rain|p) = 3/9	**P(rain|n) = 2/5**
temperature	
P(hot|p) = 2/9	**P(hot|n) = 2/5**
P(mild|p) = 4/9	**P(mild|n) = 2/5**
P(cool|p) = 3/9	**P(cool|n) = 1/5**
humidity	
P(high|p) = 3/9	**P(high|n) = 4/5**
P(normal|p) = 6/9	**P(normal|n) = 1/5**
windy	
P(true|p) = 3/9	**P(true|n) = 3/5**
P(false|p) = 6/9	**P(false|n) = 2/5**

Play-tennis example: estimating $P(C_i|X)$

(Example from: Tom Mitchell "Machine Learning")

An incoming sample: X = <sunny, cool, high, true>

$P(play|X) = P(X|p) \cdot P(p) =$
$P(p) \cdot P(sunny|p) \cdot P(cool|p) \cdot P(high|p) \cdot P(true|p) =$
$9/14 \cdot 2/9 \cdot 3/9 \cdot 3/9 \cdot 3/9 = .0053$

$P(Don't\ play\ |X) = (X|n) \cdot P(n) =$
$P(p) \cdot P(sunny|n) \cdot P(cool|n) \cdot P(high|n) \cdot P(true|n) =$
$5/14 \cdot 3/5 \cdot 1/5 \cdot 4/5 \cdot 3/5 = .0206$

Class *n* (don't play) has higer probability than class *p* (play) for sample X.

Naïve Bayes - Mahout

- Mahout provides a parallel Naïve Bayes implementation using Hadoop
- Not a general purpose implementation
- Intended for classifying text
- classifier.bayes.TrainClassifier & TestClassifier

Naïve Bayes – Text Classification

- Documents are classified into categories
- Terms and their number of occurrences in the document are considered features
- The category of the document is the class label

Naïve Bayes – Text Classification

- Features (term frequencies) weights are based on tf-idf from information retrieval
- $tf_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$, where
  - $n_{i,j}$ is the term count (number of occurrences) of term i in document j
  - $\sum_k n_{k,j}$ is the total number of terms in document j

Naïve Bayes – Text Classification

- $idf_i = log\frac{|D|}{|D_i|}$, where
  - |D| is the total number of documents
  - $|D_i|$ is the number of documents with term I
- Weighting the term frequency features helps put emphasis on important terms and helps reduce the impact of common words

Naïve Bayes – Mahout Example

- Twenty newsgroups data set contains postings from twenty USENET newsgroups
- Problem: predict which newsgroup a post belongs to based on the post’s text
- Class label: the newsgroup each post is from

- org.apache.mahout.classifier.bayes.TrainClassifier

  --input 20news --output model
- org.apache.mahout.classifier.bayes.TestClassifier

  --model model --testDir 20news

Naïve Bayes – Mahout Example

TestClassifier output

Summary

Correctly Classified Instances : 18369 97.5621%
Incorrectly Classified Instances : 459 2.4379%
Total Classified Instances : 18828

===

Confusion Matrix

----

a	b	c	d	e	f	g	h	i	j	k	l	m	n	o	p	q	r	s	t	<--Classified as
994	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	994 a = rec.motorcycles
0	976	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	2	1	980 b = comp.windows.x
7	0	929	1	0	0	0	0	0	0	0	0	1	0	2	0	0	0	0	0	940 c = talk.politics.mideast
0	0	0	905	0	0	1	0	0	0	0	0	0	0	0	0	3	0	1	0	910 d = talk.politics.guns
4	1	4	27	388	1	0	1	0	5	1	1	2	2	149	7	2	33	0	0	628 e = talk.religion.misc
3	0	0	0	0	985	0	1	0	0	0	0	0	1	0	0	0	0	0	0	990 f = rec.autos
0	0	0	0	0	0	993	1	0	0	0	0	0	0	0	0	0	0	0	0	994 g = rec.sport.baseball
0	0	0	0	0	0	1	998	0	0	0	0	0	0	0	0	0	0	0	0	999 h = rec.sport.hockey
0	0	0	0	0	0	0	0	956	0	2	0	0	0	0	0	0	0	2	1	961 i = comp.sys.mac.hardware
0	0	0	0	0	0	0	0	0	981	0	0	5	0	0	1	0	0	0	0	987 j = sci.space
0	0	0	0	0	0	0	0	0	0	978	0	1	0	0	0	0	0	2	1	982 k = comp.sys.ibm.pc.hardware
1	0	3	36	0	1	2	1	0	5	0	697	4	0	3	3	19	0	0	0	775 l = talk.politics.misc
0	2	0	0	0	0	0	0	0	0	2	0	966	0	0	0	0	0	2	1	973 m = comp.graphics

. . .

Classification with Decision Tree Induction

This algorithm makes Classification Decision for a test sample with the help of tree like structure (Similar to Binary Tree OR k-ary tree)

- Nodes in the tree are attribute names of the given data
- Branches in the tree are attribute values
- Leaf nodes are the class labels

Supervised Algorithm (Needs Dataset for creating a tree)

Greedy Algorithm (favourite attributes first)

Building Decision Tree

Two step method

- Tree Construction
  1. Pick an attribute for division of given data
  2. Divide the given data into sets on the basis of this attribute
  3. For every set created above - repeat 1 and 2 until you find leaf nodes in all the branches of the tree - Terminate
- Tree Pruning (Optimization)
  - Identify and remove branches in the Decision Tree that are not useful for classification
    - Pre-Pruning
    - Post Pruning

Assumptions and Notes for Basic Algorithm

Attributes are categorical

  if continuous-valued, they are discretized in advanced

Examples are partitioned recursively based on selected attributes

Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)

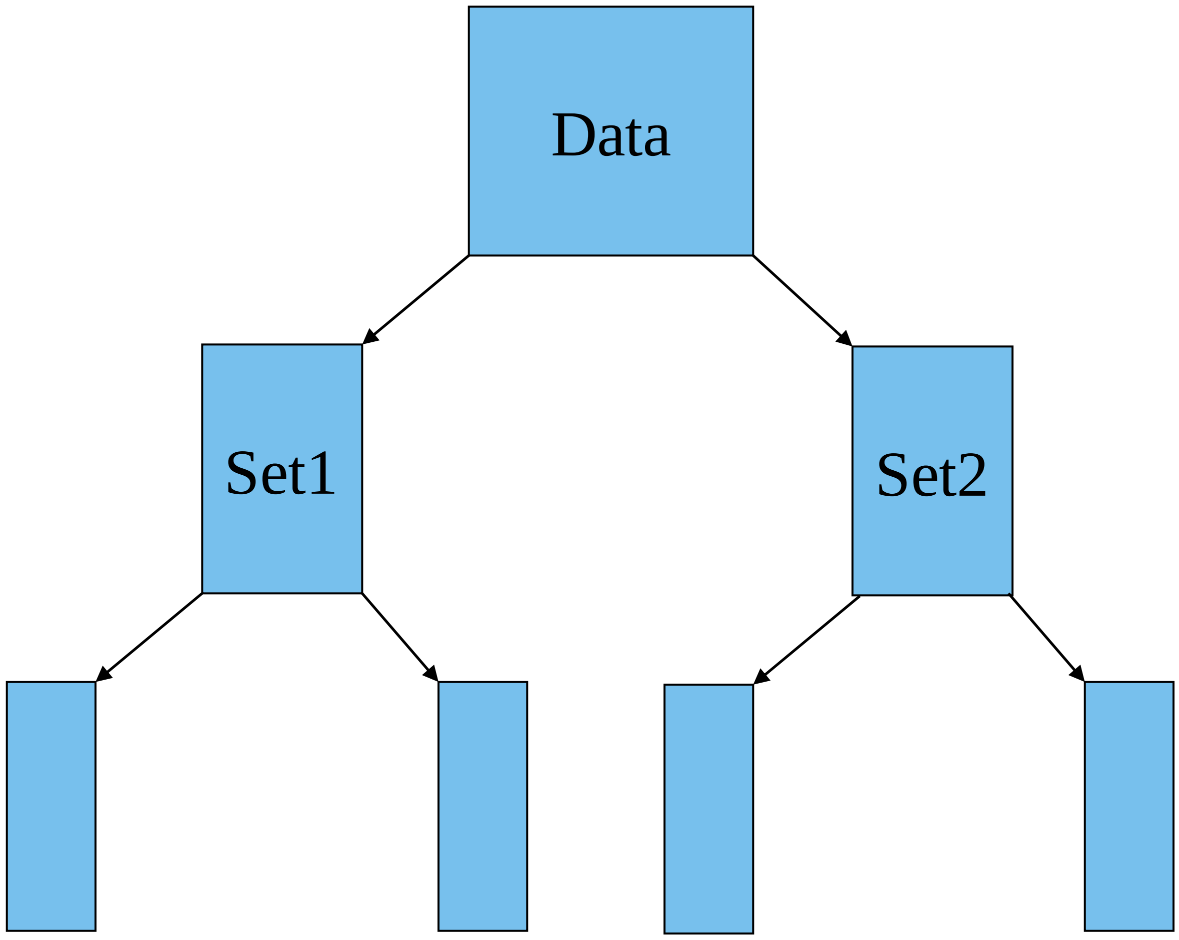
At start, all the training examples are at the root

Algorithm at work….
(Tree Construction - Step 1 & 2)

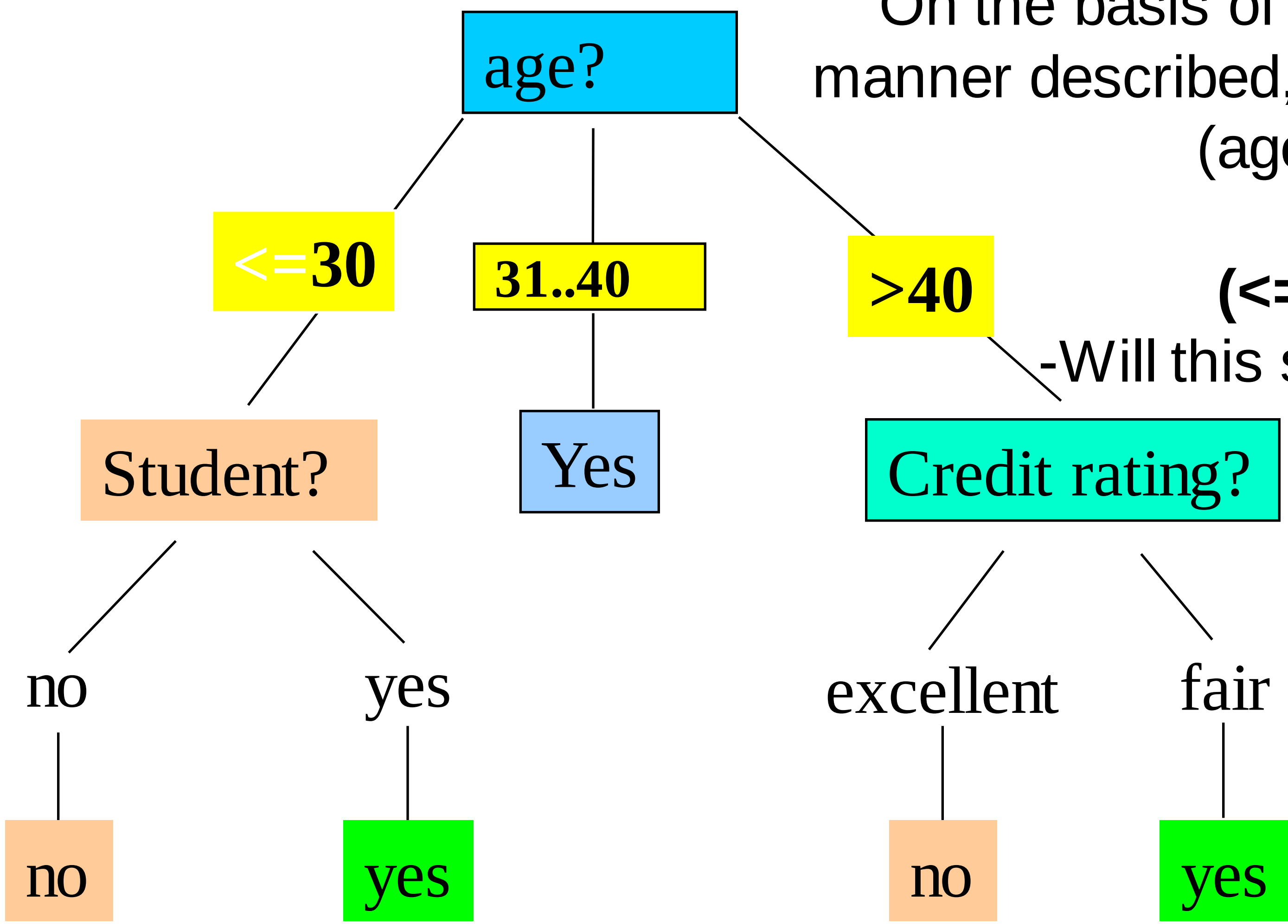
age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Algorithm in action….



Final Decision Tree



On the basis of tree constructed in the manner described, classify a test sample (age, student, creditrating, buys_computer)

(<=30, yes, excellent, ?)

-Will this student buy computer?

Tree Construction (Termination Conditions)

All samples for a given node belong to the same class

There are no remaining attributes for further partitioning

– majority voting is employed for classifying the leaf

There are no samples left

Attribute Selection Advancements

We want to find the most "useful" attribute in classifying a sample. Two measures of usefulness:

- Information Gain
  - Attributes are assumed to be categorical
- Gini Index (IBM IntelligentMiner)
  - Attributes are assumed to be contineous
  - Assume there exist several possible split values for each attribute

How to calculate Information "Gain"

In a given Dataset, assume there are two classes, P and N (yes and no from example)

Let the set of examples S contain p elements of class P and n elements of class N

The amount of information, needed to decide if an arbitrary example in S belongs to P or N is defined as

$$I(p,n) = -\frac{p}{p+n}\log_2\frac{p}{p+n} - \frac{n}{p+n}\log_2\frac{n}{p+n}$$

Entropy

- Entropy measures the impurity of a set of samples.
  - It is lowest, if there is at most one class present, and it is highest, if the proportions of all present classes are equal. That is,
    - If all examples are positive or all negative, entropy is low (zero).
    - If half are positive and half are negative, entropy is high (1.0)

Information Gain in Decision Tree Induction

Assume that using attribute A a set *S* will be partitioned into sets {*S1*, *S2* , …, *Sv*}

If *Si* contains *pi* examples of *P* and *ni* examples of *N*, the entropy, or the expected information needed to classify objects in all subtrees *Si* is

$$E(A)=\sum_{i=1}^{v}\frac{p_i+n_i}{p+n}I(p_i,n_i)$$

The encoding information that would be gained by branching on *A*. *This is the expected reduction in entropy if we go with A.*

$$Gain(A)=I(p,n)-E(A)$$

Play-tennis example: which attribute do we take first

Outlook	Temperature	Humidity	Windy	Class
sunny	hot	high	false	N
sunny	hot	high	true	N
overcast	hot	high	false	P
rain	mild	high	false	P
rain	cool	normal	false	P
rain	cool	normal	true	N
overcast	cool	normal	true	P
sunny	mild	high	false	N
sunny	cool	normal	false	P
rain	mild	normal	false	P
sunny	mild	normal	true	P
overcast	mild	high	true	P
overcast	hot	normal	false	P
rain	mild	high	true	N

I (Humidity[9+,5-]) = .940

Humidity = high [3+,4-] E=0.985
Humidity=normal [6+,1-] E = .592
Gain(S, Humidity) = .940 – 7/14(.985) – (7/14).592 = .151

Windy = false [6+,2-], E = .811
Windy = true [3+,3-], E = 1.0

Gain (S, Windy) = .940 – (8/14)(.811 – (6/14)(1.0) = .048

Humidity split into two classes , one with a great split of 6+ and 1-. The other was not so great of 3+,3-
Wind split into two classes, one with an Ok split of 6+2-
And the other was terrible of 3+,3- (max entropy of 1.0).

So Humidity is the best attribute between these two.

Gain(S,outlook) = .246
Gain(S,humidity) = .151
Gain(S,wind) = .048
Gain(S,Temperature) = .029

Gini Index (IBM IntelligentMiner

If a data set T contains examples from n classes, gini index, $gini(T)$ is defined as

$$gini(T) = 1 - \sum_{j=1}^{n} p_j^2$$

where p_j is the relative frequency of class j in T.

If a data set T is split into two subsets $T1$ and $T2$ with sizes $N1$ and $N2$ respectively, the *gini* index of the split data contains examples from n classes, the *gini* index $gini(T)$ is defined as

$$gini_{split}(T) = \frac{N_1}{N} gini(T_1) + \frac{N_2}{N} gini(T_2)$$

The attribute provides the smallest $ginisplit(T)$ is chosen to split the node (*need to enumerate all possible splitting points for each attribute*).

Extracting Classification Rules

Represent the knowledge in the form of IF-THEN rules

One rule is created for each path from the root to a leaf

Each attribute-value pair along a path forms a conjunction

The leaf node holds the class prediction

Rules are easier for humans to understand

Example

IF *age* = "<=30" AND *student* = "*no*" THEN *buys_computer* = "*no*"

IF *age* = "<=30" AND *student* = "*yes*" THEN *buys_computer* = "*yes*"

IF *age* = "31…40" THEN *buys_computer* = "*yes*"

IF *age* = ">40" AND *credit_rating* = "*excellent*" THEN *buys_computer* = "*yes*"

IF *age* = "<=30" AND *credit_rating* = "*fair*" THEN *buys_computer* = "*no*"

Overfitting

Generated Decision Tree is said to *overfit* the training data if,

- *It results in poor accuracy to classify test samples*
- *It has too many branches, that reflect anomalies due to noise or outliers*

Two approaches to avoid overfitting –

- Tree Pre-Pruning – Halt tree construction early – that is, do not split a node if the goodness measure falls below a threshold
  - It is difficult to choose appropriate threshold
- Tree Post-Pruning - Remove branches from a “fully grown” tree—get a sequence of progressively pruned trees
  - Use a set of data different from the training data to decide which is the “best pruned tree”

Classifier Accuracy Estimation

Why estimate a classifier accuracy?

- Comparing classifiers for the given dataset (Different classifiers will favor different domain of datasets)
- One needs to estimate how good the prediction will be.

Methods of estimating accuracy

- **Holdout** – randomly partition the given data into two independent sets and use one for training (typically 2/3rd) and the other for testing (1/3rd)
- **k-fold cross-validation** – randomly partition the given data into 'k' mutually exclusive subsets (folds). Training and testing is performed k times.

Accuracy Improvement

Methods

- Bagging (Bootstrap aggregation) – Number of trees are constructed on subsets of given data and majority voting is taken from these trees to classify a test sample.
- Boosting – attaching weights (importance) to the training samples and optimizing the weights during training and further using these weights to classify the test sample. Advantage – avoids outliers

Further Reading

1. Robust Regression - alternatives to Least Squares.
    - Robust regression and outlier detection, By Peter J. Rousseeuw and Annick M. Leroy. Book. 1987.
    - http://en.wikipedia.org/wiki/Robust_regression#Least_squares_alternatives

2. Excellent source for everything we covered and more:
    - http://www.stanford.edu/class/cs229/notes/cs229-notes1.pdf
    - Also includes the statistical justification for least squares (LS has the same meaning as MLE under a few assumptions about the distribution).

3. Vowpal Wabbit (Fast Online Learning)
    - http://hunch.net/~vw/

4. Stochastic Gradient Descent Examples
    - http://leon.bottou.org/projects/sgd

5. CS229 Lecture Notes
    - http://www.stanford.edu/class/cs229/notes/cs229-notes1.pdf

References

1. B. Carpenter: Lazy Sparse Stochastic Gradient Descent for Regularized Multinomial Logistic Regression.
2. J. Langord, A. Smola, and M. Zinkevich: Slow Learners are Fast.
3. K. Weinberger, A. Dasgupta, J. Langord, A. Smola, and J. Attenberg: Feature Hashing for Large Scale Multitask Learning. In Proceedings of the 26th International Conference on Machine Learning, 2009.
4. J. Rennie, L. Shih, J. Teevan, and D. Karger: Tackling the Poor Assumptions of Naive Bayes Text Classifiers. In Proceedings of the Twentieth International Conference on Machine Learning (ICML-2003), 2003.