Classification: Decision Trees, Naïve Bayes, and Accuracy

COSC 757: Spring 2016

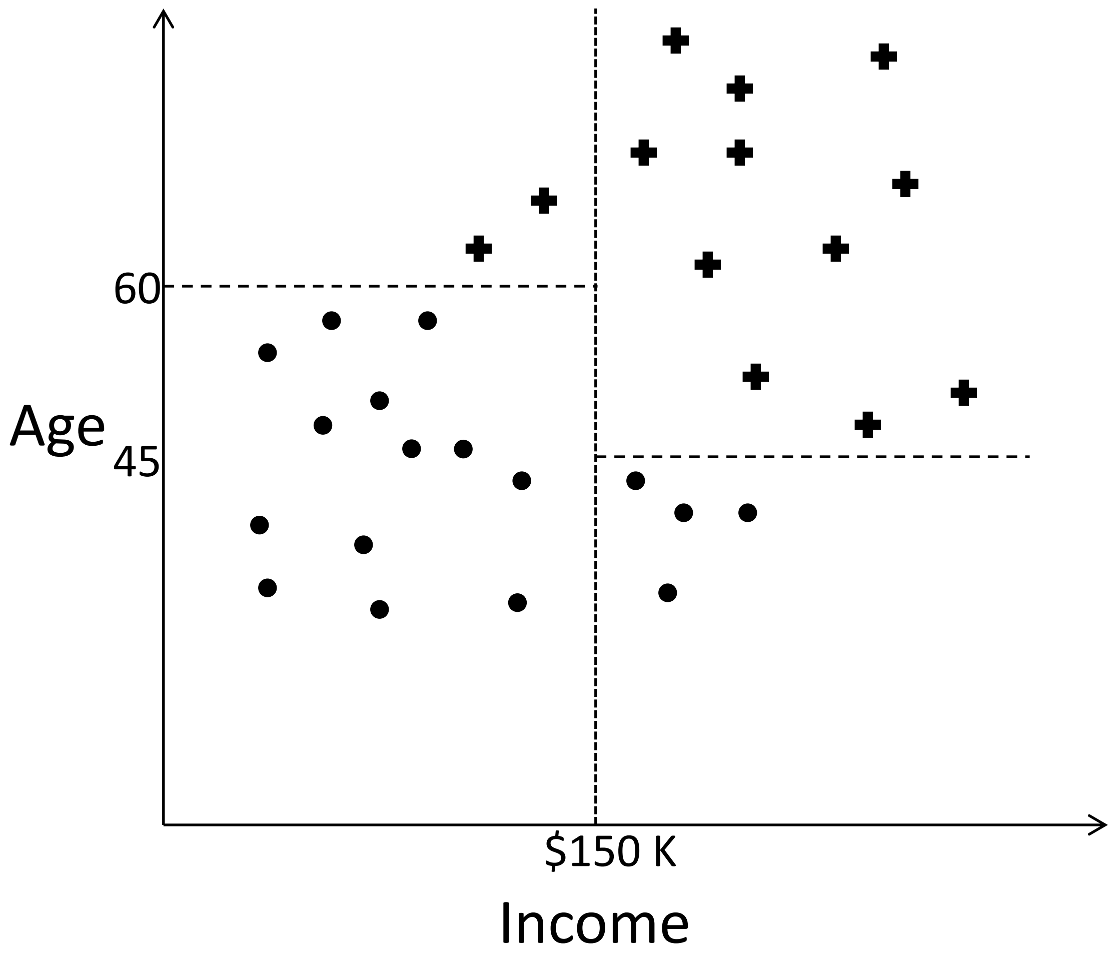
Decision Trees

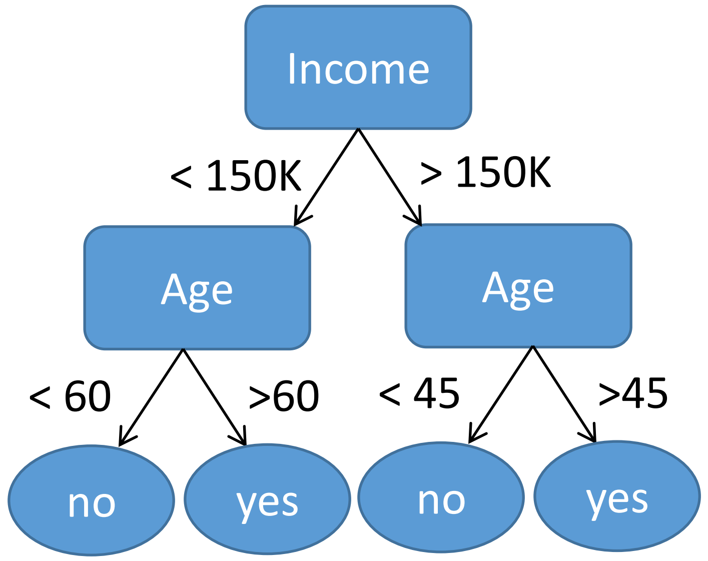
- Uses a flowchart-like tree structure.
    - Each node represents a test on an attribute
    - Each branch represents an outcome
    - Each leaf node holds a class label
- Intuitive classifier
- Very simple but with randomization can be the best!

Requirements for Using Decision Trees

- Supervised learning requires a pre-classified target variable for training
- The training dataset should be rich and varied, providing the algorithm with a representative cross section of the data
- The target attribute class must be discrete

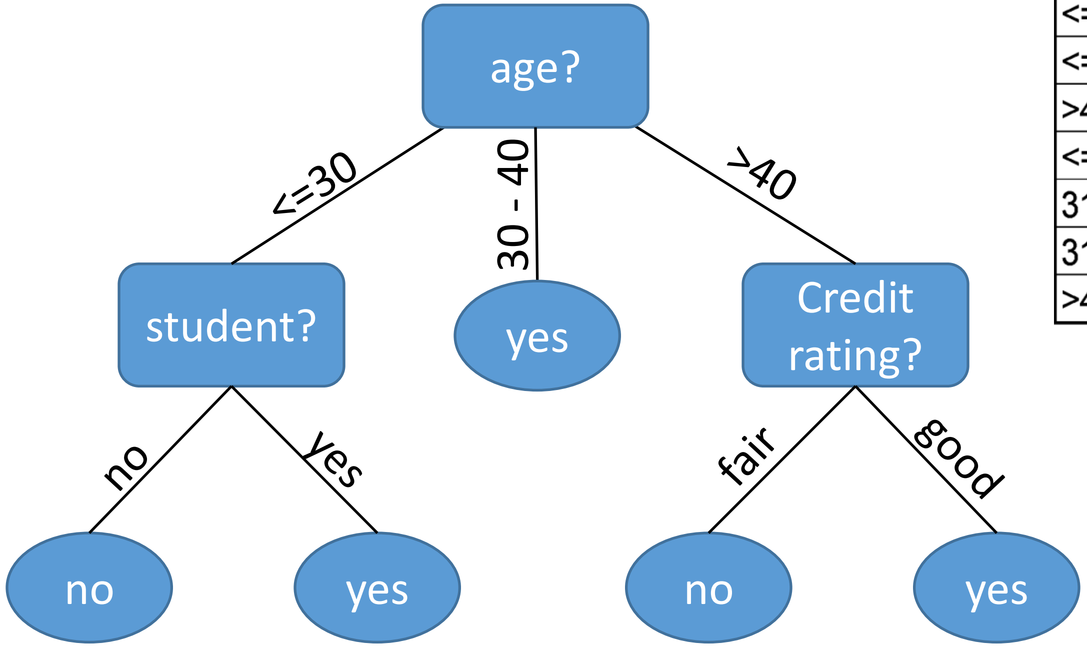
Decision Trees: Basic Concept





Decision Tree: Another Example

- Target Attribute: Buys_computer
- Resulting tree:



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 for Decision Tree Induction

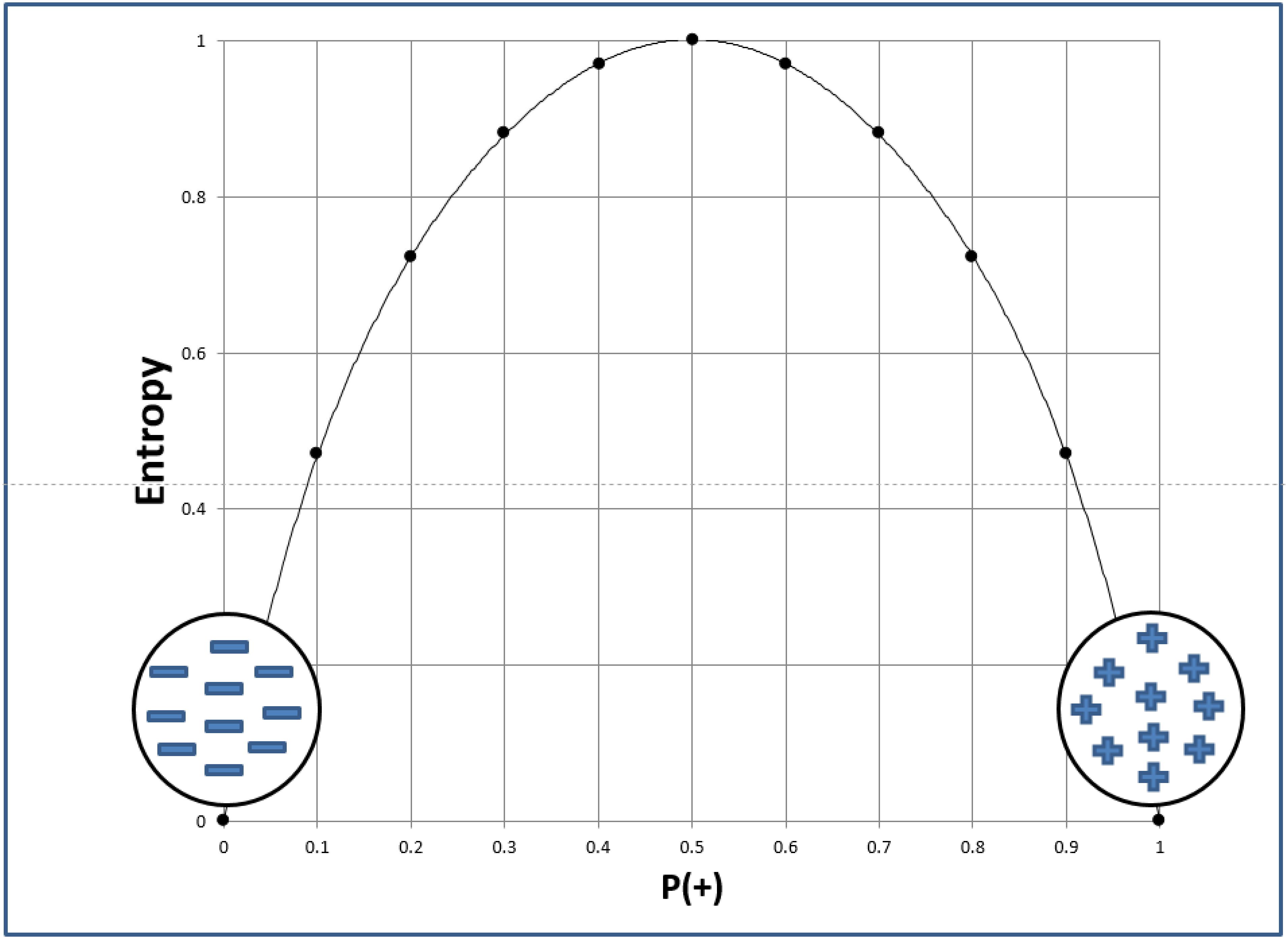
- Basic algorithm (a greedy (nonbacktracking) algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - 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

Algorithms and Attribute Selection

- Which attribute best partitions the data?
- A number of measures are used to determine the best attribute for partitioning
- Algorithms:
  - ID3 (Information Gain)
  - C4.5 (Gain Ratio)
  - CART (Gini Index)

Att (ID3)

- Se
- Le
  b
- Ex
  in
- Ir
  pa
- Ir



Attribute Selection

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

$$Info(D) = \left(-\frac{9}{14}log_2\frac{9}{14} - \frac{5}{14}log_2\frac{5}{14}\right) = 0.940$$

$$Info_{age}(D) = \frac{5}{14}\times\left(-\frac{2}{5}log_2\frac{2}{5} - \frac{3}{5}log_2\frac{3}{5}\right) + \frac{4}{14}\times\left(-\frac{4}{4}log_2\frac{4}{4}\right) + \frac{5}{14}\times\left(-\frac{3}{5}log_2\frac{3}{5} - \frac{2}{5}log_2\frac{2}{5}\right)$$

= 0.694

$$Gain(age) = 0.940\ - 0.694 = 0.246$$

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$

Computing Information-Gain for Continuous Attributes

- Let attribute A be a continuous-valued attribute
- Must determine the *best split point* for A
  - Sort the value A in increasing order
  - Typically, the midpoint between each pair of adjacent values is considered as a possible *split point*
    - $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}
  - The point with the *minimum expected information requirement* for A is selected as the split-point for A
- Split:
  - D1 is the set of tuples in D satisfying A ≤ split-point, and D2 is the set of tuples in D satisfying A > split-point

Gain Ratio for Attribute Selection (C4.5)

- Information gain measure is biased towards attributes with a large number of values
- C4.5 (a successor of ID3) uses gain ratio to overcome the problem (normalization to information gain)

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

  – GainRatio(A) = Gain(A)/SplitInfo(A)

- Ex.

$$SplitInfo_{income}(D) = -\frac{4}{14} \times \log_2\left(\frac{4}{14}\right) - \frac{6}{14} \times \log_2\left(\frac{6}{14}\right) - \frac{4}{14} \times \log_2\left(\frac{4}{14}\right) = 1.557$$

  – gain_ratio(income) = 0.029/1.557 = 0.019

- The attribute with the maximum gain ratio is selected as the splitting attribute

Gini Index (CART, IBM Intelligent Miner)

- If a data set *D* contains examples from *n* classes, gini index, *gini*(*D*) is defined as $$Gini(D) = \sum_{i=1}^{m} p_i^{2} \qquad |C_{i,D}|/|D|$$

  where p_i is the relative frequency of class *j* in *D*
- If a data set *D* is split on A into two subsets D_1 and D_2, the *gini* index *gini*(*D*) is defined as $$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$
- Reduction in Impurity: $$\Delta Gini(A) = Gini(D) - Gini_A(D)$$
- The attribute that provides the largest reduction in impurity) is chosen to split the node (*need to enumerate all the possible splitting points for each attribute*)

Computation of Gini Index

- Ex. D has 9 tuples in buys_computer = "yes" and 5 in "no"

$$gini(D) = 1 - \left(\frac{9}{14}\right)^2 - \left(\frac{5}{14}\right)^2 = 0.459$$

- Suppose the attribute income partitions D into 10 in D_1: {low,medium} and 4 in D_2

$$gini_{income \in \{low,medium\}}(D) = \left(\frac{10}{14}\right) Gini(D_1) + \left(\frac{4}{14}\right) Gini(D_1)$$

$$\begin{aligned} &= \frac{10}{14}\left(1 - \left(\frac{7}{10}\right)^2 - \left(\frac{3}{10}\right)^2\right) + \frac{4}{14}\left(1 - \left(\frac{2}{4}\right)^2 - \left(\frac{2}{4}\right)^2\right) \\ &= 0.443 \\ &= Gini_{income \in \{high\}}(D). \end{aligned}$$

$Gini_{\{low,high\}}$ is 0.458; $Gini_{\{medium,high\}}$ is 0.450. Thus, split on the {low,medium} (and {high}) since it has the lowest Gini index resulting in the largest reduction in impurity.

Comparing Attribute Selection Measures

- The three measures, in general, return good results but
  - **Information gain**:
    - biased toward multivalued attributes
  - **Gain ratio**:
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - **Gini index**:
    - biased toward multivalued attributes
    - provides binary split
    - has difficulty when # of classes is large
    - tends to favor tests that result in equal-sized partitions and purity in both partitions

Comparing Attribute Selection Measures: Example

Customer	Savings	Assets	Income	Credit Risk
1	Medium	High	High	Good
2	Low	Low	Medium	Bad
3	High	Medium	Low	Bad
4	Medium	Medium	Medium	Good
5	Low	Medium	High	Good
6	High	High	Low	Good
7	Low	Low	Low	Bad
8	Medium	Medium	High	Good

Gain(Savings)	0.360073065
Gain(Assets)	**0.548794941**
Gain(Income)	0.468036283
GainRatio(Savings)	0.230627112
GainRatio(Assets)	**0.365863294**
GainRatio(Income)	0.299777647

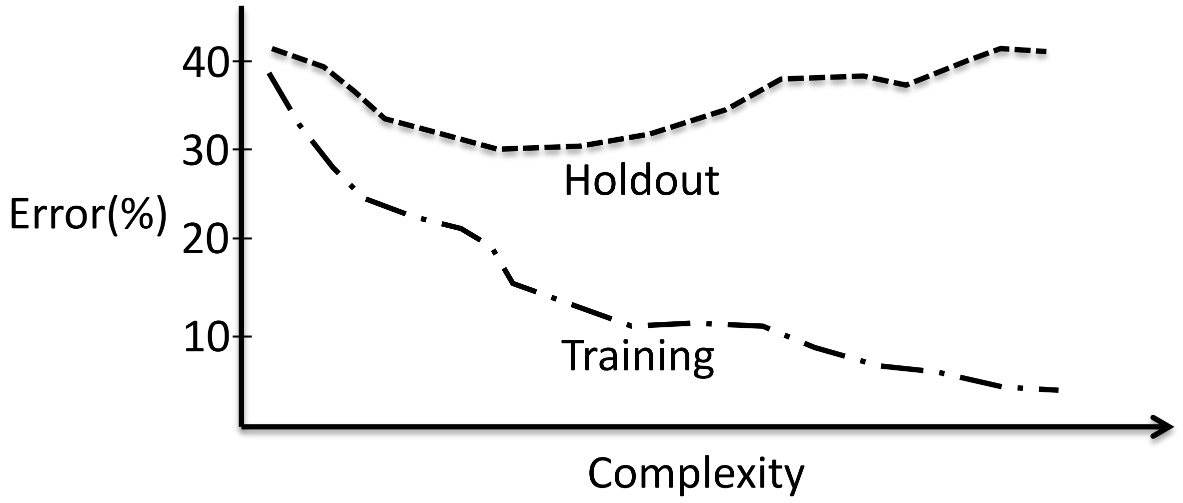
Candidate Split	Left Child Node	Right Child Node	Gini(D)	Gini(A)	Impurity
1	Savings = Low	Savings = {Medium,High}	0.46875	0.666666667	-0.197916667
2	**Savings = Medium**	**Savings = {Low,High}**	**0.46875**	**0.5**	**-0.03125**
3	Savings = High	Savings = {Low,Medium}	0.46875	0.75	-0.28125
4	Assets = Low	Assets = {Medium,High}	0.46875	0.625	-0.15625
5	Assets = Medium	Assets = {Low,High}	0.46875	0.5625	-0.09375
6	Assets = High	Assets = {Low,Medium}	0.46875	0.625	-0.15625
7	Income = Low	Income = {Medium, High}	0.46875	0.666666667	-0.197916667
8	Income = Medium	Income = {Low, High}	0.46875	0.75	-0.28125
9	**Income = High**	**Income = {Low, Medium}**	**0.46875**	**0.5**	**-0.03125**

Other Attribute Selection Measures

- CHAID: a popular decision tree algorithm measure based on χ^2 test for independence
- C-SEP: performs better than information gain and gini index in certain cases
- G-statistic: has a close approximation to χ^2 distribution
- MDL (Minimal Description Length) principle (i.e., the simplest solution is preferred):
  - The best tree is the one that requires the fewest # of bits to both (1) encode the tree, and (2) encode the exceptions to the tree
- Multivariate splits (partition based on multiple variable combinations)
  - CART: finds multivariate splits based on a linear comb. of attrs.
- Which attribute selection measure is the best?
  - Most give good results, none is significantly superior than others

Overfitting

- Overfitting: the tendency of data mining procedures to tailor models to the training data
  - An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Model does not represent general population
  - Poor accuracy for unseen samples



Avoiding Overfitting in Decision Trees

- Two approaches to avoid overfitting
  - Prepruning: *Halt tree construction early*-do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - Postpruning: *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"

Enhancements to Basic Decision Tree Induction

- Allow for **continuous-valued attributes**
  - Dynamically define new discrete-valued attributes that partition the continuous attribute value into a discrete set of intervals
- Handle **missing attribute values**
  - Assign the most common value of the attribute
  - Assign probability to each of the possible values
- **Attribute construction**
  - Create new attributes based on existing ones that are sparsely represented
  - This reduces fragmentation, repetition, and replication

Bayesian Classification

- Simple probabilistic classifier
- Based on Bayes’ theorem
- Assumes strong (naïve) independence between features
  - For example:
    - A fruit can be considered an apple if it is red, round and about 10 cm in diameter
    - Red, round, and diameter are all independent

Bayes' Theorem

- Let **X** be a data sample ("*evidence*"): class label is unknown
- Let H be a *hypothesis* that X belongs to class C
- Classification is to determine P(H|**X**), (*posteriori probability),* the probability that the hypothesis holds given the observed data sample **X**
- **Example:** Given the evidence of age and income predict of someone buys a computer
- P(H) (*prior probability*), the initial probability
  - **X** will buy computer, regardless of age, income, …
- P(**X**): probability that sample data is observed
  - Probability that **X** is 31..40, medium income
- P(**X**|H) (likelihood), the probability of observing the sample **X**, given that the hypothesis holds
  - Given that **X** will buy computer, the prob. that X is 31..40, medium income

Bayes' Theorem

- Given training data **X**, *posteriori probability of a hypothesis* H, P(H|**X**), follows the Bayes theorem

$$P(H|\mathbf{X}) = \frac{P(\mathbf{X}|H)P(H)}{P(\mathbf{X})}$$

- Informally, this can be written as

  posteriori = likelihood x prior/evidence

- Predicts that **X** belongs to C_i iff the probability $P(C_i|\mathbf{X})$ is the highest among all the $P(C_k|X)$ for all the *k* classes
- Practical difficulty: <u>requires the initial knowledge of many probabilities, significant computational cost</u>

Towards Naïve Bayesian Classifier

- Let D be a training set of tuples and their associated class labels, and each tuple is represented by an n-D attribute vector **X** = $(x_1, x_2, \ldots, x_n)$
- Suppose there are *m* classes $C_1, C_2, \ldots, C_m$.
- Classification is to derive the maximum posteriori, i.e., the maximal $P(C_i|\mathbf{X})$
- This can be derived from Bayes' theorem

$$P(C_i|\mathbf{X}) = \frac{P(\mathbf{X}|C_i)P(C_i)}{P(\mathbf{X})}$$

- Since P(X) is constant for all classes, only

$$P(C_i|\mathbf{X}) = P(\mathbf{X}|C_i)P(C_i)$$

needs to be maximized

Derivation of Naïve Bayes Classifier

- A simplified assumption: attributes are **conditionally independent** (i.e., no dependence relation between attributes):

$$P(\mathbf{X}|C_i) = \prod_{k=1}^{n} P(x_k|C_i) = P(x_1|C_i) \times P(x_2|C_i) \times \ldots \times P(x_n|C_i)$$

- This greatly reduces the computation cost: Only counts the **class distribution**
- If A_k is categorical, $P(x_k|C_i)$ is the # of tuples in C_i having value x_k for A_k divided by $|C_{i,D}|$ (# of tuples of C_i in D)
- If A_k is continous-valued, $P(x_k|C_i)$ is usually computed based on a Gaussian distribution with a mean μ and standard deviation σ

and $P(x_k|C_i)$ is

$$g(x,\mu,\sigma) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

$$P(\mathbf{X}|C_i) = g(x_k, \mu_{C_i}, \sigma_{C_i})$$

Naïve Bayesian Classifier: Training Dataset

Class:

C1:buys_computer = ‘yes’

C2:buys_computer = ‘no’

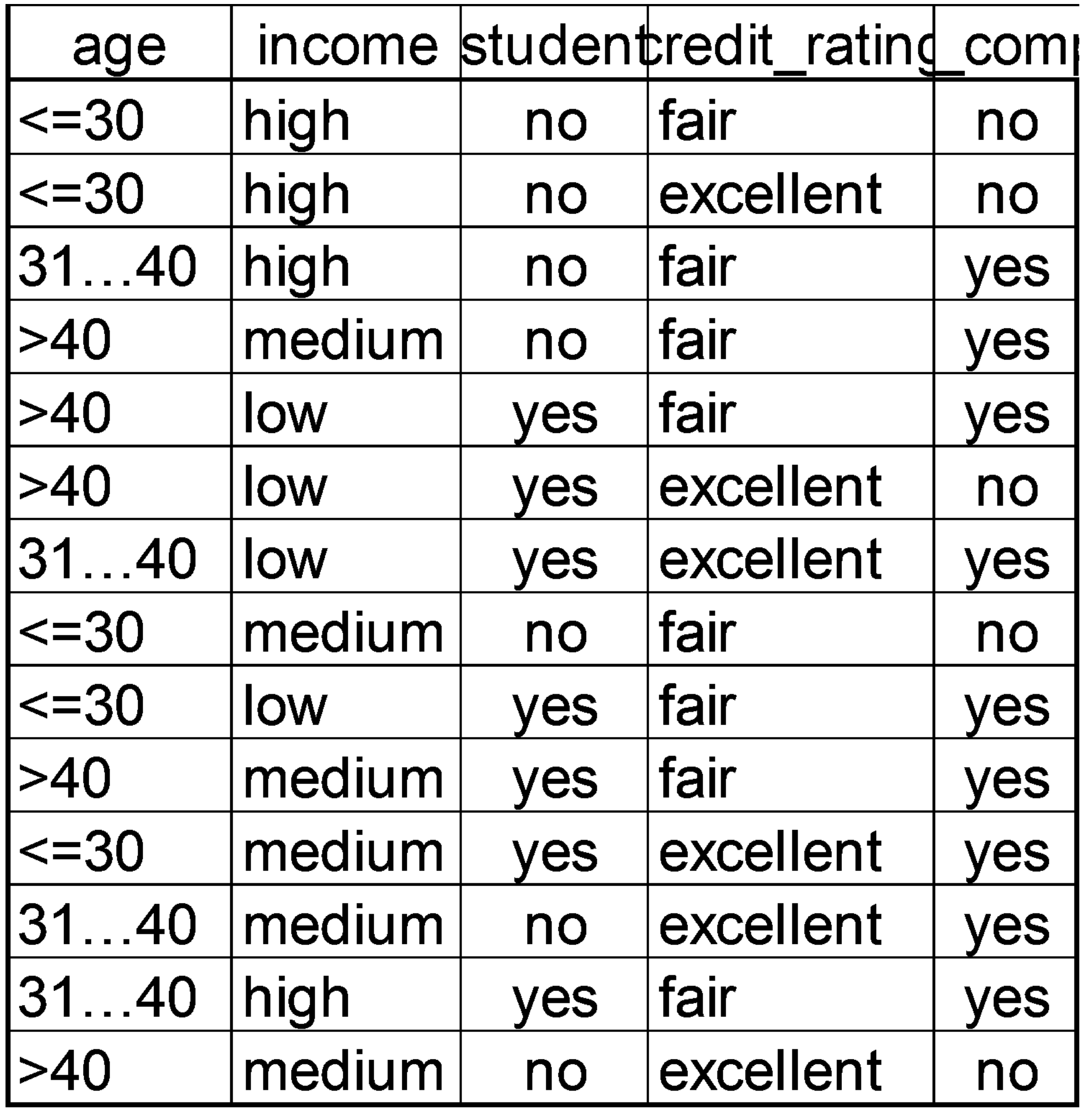
Data sample

X = (age <=30,

Income = medium,

Student = yes

Credit_rating = Fair)



age	income	student	credit_rating	_comp
<=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

Naïve Bayesian Classifier: An Example

- $P(C_i)$: P(buys_computer = “yes”) = 9/14 = **0.643**
  P(buys_computer = “no”) = 5/14= **0.357**
- Compute $P(X|C_i)$ for each class
  P(age = “<=30” | buys_computer = “yes”) = 2/9 = 0.222
  P(age = “<= 30” | buys_computer = “no”) = 3/5 = 0.6
  P(income = “medium” | buys_computer = “yes”) = 4/9 = 0.444
  P(income = “medium” | buys_computer = “no”) = 2/5 = 0.4
  P(student = “yes” | buys_computer = “yes) = 6/9 = 0.667
  P(student = “yes” | buys_computer = “no”) = 1/5 = 0.2
  P(credit_rating = “fair” | buys_computer = “yes”) = 6/9 = 0.667
  P(credit_rating = “fair” | buys_computer = “no”) = 2/5 = 0.4
- **X = (age <= 30 , income = medium, student = yes, credit_rating = fair)**

$P(X|C_i)$: P(X|buys_computer = “yes”) = 0.222 x 0.444 x 0.667 x 0.667 = **0.044**
P(X|buys_computer = “no”) = 0.6 x 0.4 x 0.2 x 0.4 = **0.019**

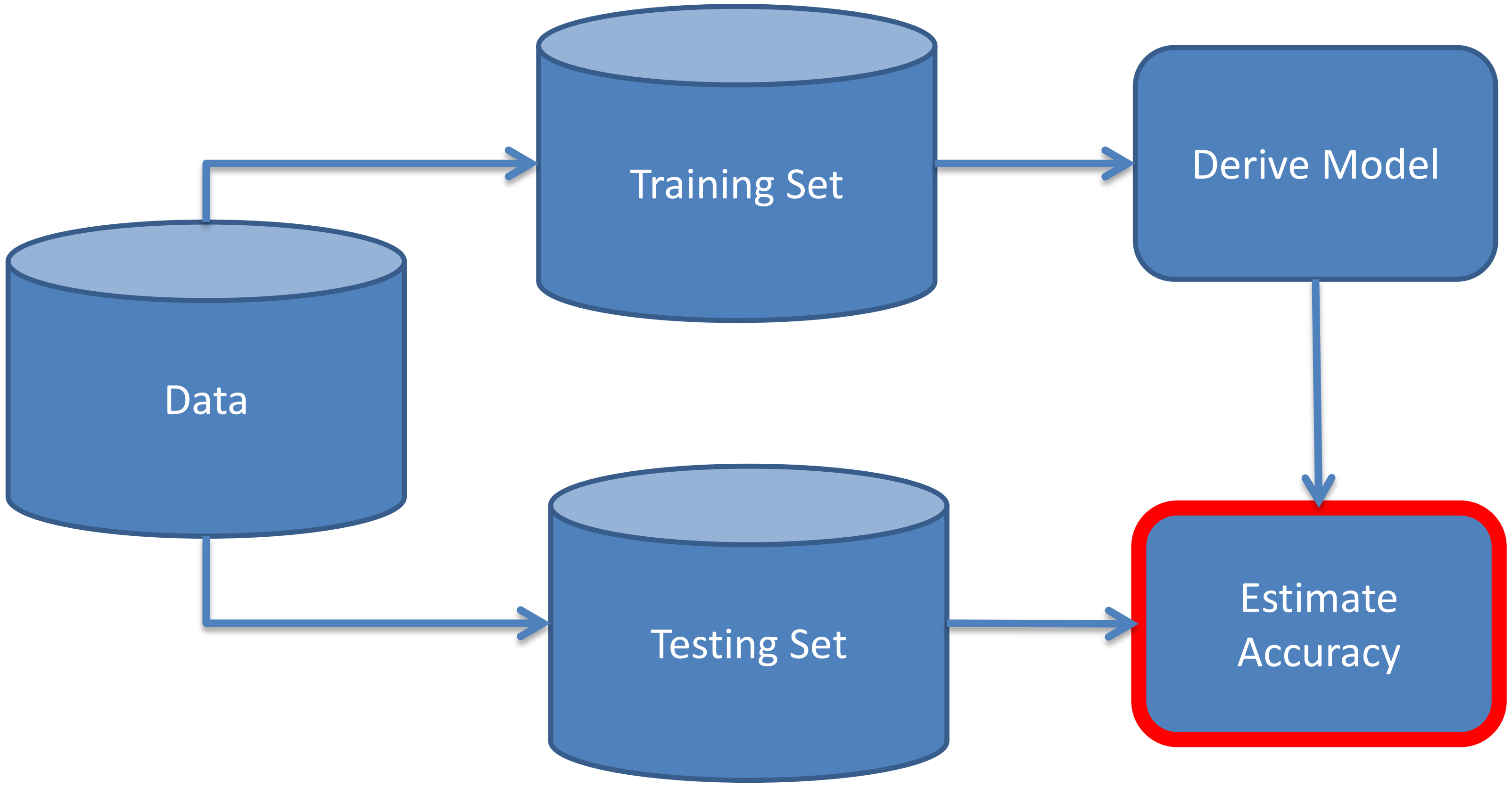
$P(X|C_i)*P(C_i)$: P(X|buys_computer = “yes”) * P(buys_computer = “yes”) = 0.028
P(X|buys_computer = “no”) * P(buys_computer = “no”) = 0.007

Therefore, X belongs to class (“buys_computer = yes”)

Evaluating Classifiers

- How well can a classifier be expected to perform on new data?
- Choice of performance measure
- How close is the estimated performance to the true performance?
- Evaluating Classifiers:
  - Performance Metrics
  - Experimental Design

Review: Supervised Learning



Confusion Matrix

Confusion Matrix:

Actual class\Predicted class	C_1	$\neg C_1$
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- Given *m* classes, an entry, $\boldsymbol{CM_{i,j}}$ in a **confusion matrix** indicates # of tuples in class *i* that were labeled by the classifier as class *j*
- May have extra rows/columns to provide totals

Accuracy and Error Rate

- **Classifier Accuracy,** or recognition rate:
  - percentage of test set tuples that are correctly classified
  - **Accuracy = (TP + TN)/All**
- **Error rate:**
  - *1 – accuracy*, or
  - **Error rate = (FP + FN)/All**

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

Sensitivity and Specificity

- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
- **Sensitivity**: True Positive recognition rate
  - **Sensitivity = TP/P**
- **Specificity**: True Negative recognition rate
  - **Specificity = TN/N**
  - **False Alarm = FP/TN+FP**
- **Accuracy is a function of sensitivity and specificity**
  - **accuracy = sensitivity*P/(P+N) + specificity*N/(P+N)**

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

Precision and Recall

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?

$$recall = \frac{TP}{TP + FN}$$

- Perfect score is 1.0
- Inverse relationship between precision & recall

F- Measures

- ***F* measure (*F_1* or *F*-score)**:
- Accuracy measure which considers both precision and recall
- Score between 0 and 1
- Traditional F measure is the harmonic mean of precision and recall

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- ***$F_ß$*:** weighted measure of precision and recall
  - assigns ß times as much weight to recall as to precision

$$F_\beta = \frac{(1+\beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

Evaluation Metrics: Example 1

Actual\Predicted	Student = Yes	Student = No	Total
Student = Yes	120	10	130
Student = No	5	100	105
Total	125	110	235

- Accuracy = (120 + 100) / 235 = **0.936**
- Error Rate = 1 - 0.936 = **0.064**
- Sensitivity = 120 / 130 = **0.923**
- Specificity = 100 / 105 = **0.952**
- Precision = 120 / 125 = **0.96**
- Recall = 120 / 130 = **0.923**
- F measure = (2 * 0.96 * 0.923) / (0.96 + 0.923) = **0.941**
- F_2 measure = ((1+4) * 0.96 * 0.923) / (4 * 0.96 + 0.923) = **0.93**
- $F_{0.5}$ measure = ((1+0.25) * 0.96 * 0.923) / (0.25 * * 0.96 + 0.923) = 0.952

Evaluation Metrics: Example 2

Actual\Predicted	Cancer = Yes	Cancer = No	Total
Cancer = Yes	90	210	300
Cancer = No	140	9560	9700
Total	230	9770	10000

- Accuracy = (90 + 9560)/10000 = 0.965
- Error Rate = 1 - 0.965 = 0.035
- Sensitivity = 90 / 300 = 0.3
- Specificity = 9560 / 9700 = 0.986
- Precision = 90/230 = 0.3913
- Recall = 90/300 = 0.3
- F measure = (2 * 0.3913 * 0. 3) / (0.3913 + 0.3) = 0.3
- F_2 measure = ((1+4) * 0.3913 * 0.3) / (4 * 0.3913 + 0.3) = 0.3
- $F_{0.5}$ measure = ((1+0.25) * 0.3913 * 0.3) / (0.25 * 0.3913 + 0.3) = 0.368

Measuring Classifier Performance

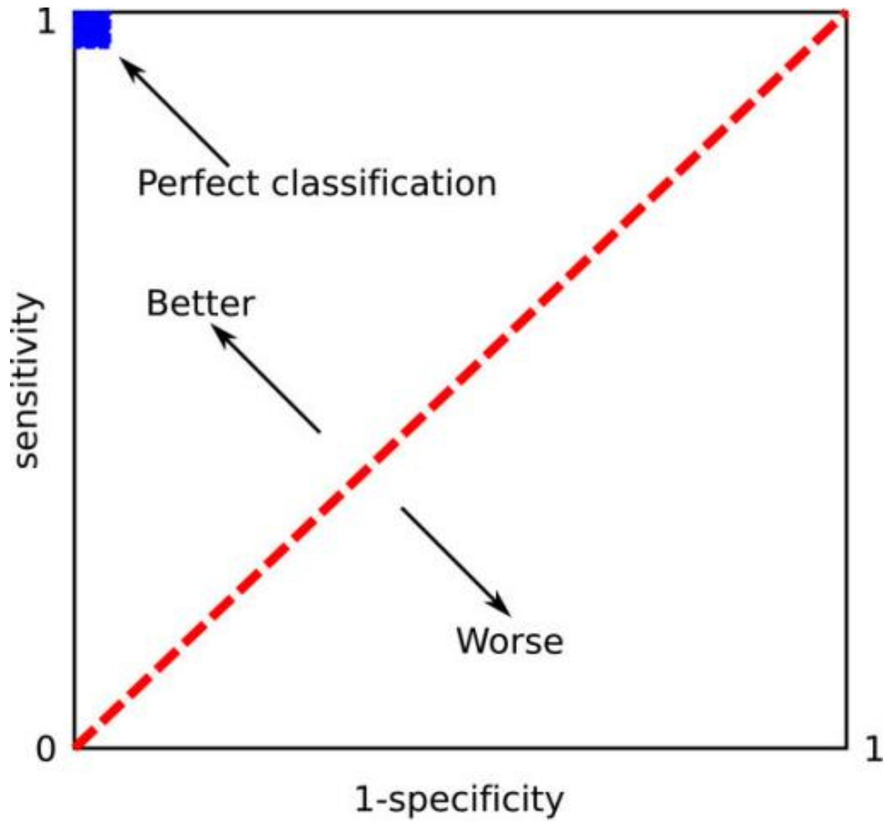
- TP, FP, TN, FN provide the relevant information
- No single measure tells the whole story
- A classifier with 90% accuracy is useless of 90% of the population does not have cancer and the 10% that do are misclassified
- Use of multiple measures recommended
- When you write up your results, always include the formula to avoid confusion!

ROC Curves

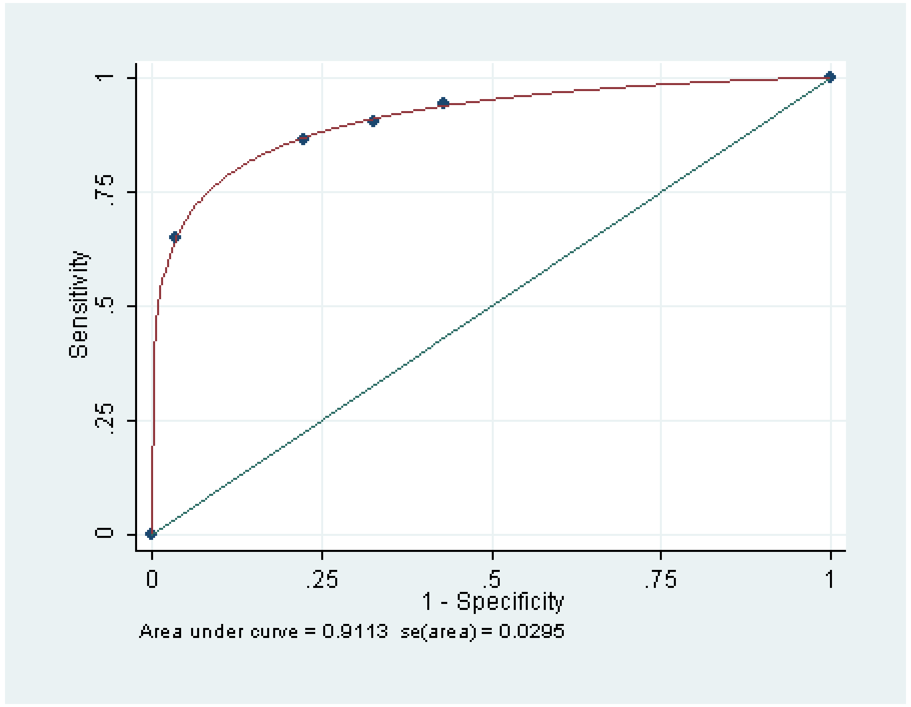
- **ROC** (Receiver Operating Characteristics) curves: for visual comparison of classification models
- Originated from signal detection theory
- Shows the trade-off between the true positive rate and the false positive rate
- The area under the ROC curve is a measure of the accuracy of the model
- Rank the test tuples in decreasing order: the one that is most likely to belong to the positive class appears at the top of the list
- The closer to the diagonal line (i.e., the closer the area is to 0.5), the less accurate is the model

ROC Space

- ROC curve is a plot of TPR against FPR which depicts relative trade-offs between benefits (true positives) and costs (false positives)



ROC Curves



- Vertical axis represents the true positive rate
- Horizontal axis rep. the false positive rate
- The plot also shows a diagonal line
- The closer an ROC curve is to the diagonal line, the less accurate the model

Ensemble Methods: Increasing Accuracy

- Ensemble methods
  - Use a combination of models to increase accuracy
  - Combine a series of k learned models, M_1, M_2, …, M_k, with the aim of creating an improved model M*
- Popular ensemble methods
  - Bagging: averaging the prediction over a collection of classifiers
  - Boosting: weighted vote with a collection of classifiers
  - Random Forests
  - Ensemble: combining a set of heterogeneous classifiers

Bagging: Bootstrap Aggregation

- **Analogy: Diagnosis based on multiple doctors' majority vote**
- Training
  - Given a set D of *d* tuples, at each iteration *i*, a training set D_i of *d* tuples is sampled with replacement from D (i.e., bootstrap)
  - A classifier model M_i is learned for each training set D_i
- Classification: classify an unknown sample **X**
  - Each classifier M_i returns its class prediction
  - The bagged classifier M* counts the votes and assigns the class with the most votes to **X**
- Prediction: can be applied to the prediction of continuous values by taking the average value of each prediction for a given test tuple
- Accuracy
  - Often significantly better than a single classifier derived from D
  - For noise data: not considerably worse, more robust
  - Proved improved accuracy in prediction

Boosting

- **Analogy: Consult several doctors, based on a combination of weighted diagnoses—weight assigned based on the previous diagnosis accuracy**
- How boosting works?
  - **Weights** are assigned to each training tuple
  - A series of k classifiers is iteratively learned
  - After a classifier M_i is learned, the weights are updated to allow the subsequent classifier, M_{i+1}, to **pay more attention to the training tuples that were misclassified** by M_i
  - The final classifier **M* combines the votes** of each individual classifier, where the weight of each classifier's vote is a function of its accuracy
- Boosting algorithm can be extended for numeric prediction
- Comparing with bagging: Boosting tends to have greater accuracy, but it also risks overfitting the model to misclassified data

Adaboost (Freund and Shapire, 1997)

- Given a set of *d* class-labeled tuples, ($\mathbf{X_1}$, y_1), …, ($\mathbf{X_d}$, y_d)
- Initially, all the weights of tuples are set the same (1/d)
- Generate k classifiers in k rounds. At round i,
  - Tuples from D are sampled (with replacement) to form a training set D_i of the same size
  - Each tuple's chance of being selected is based on its weight
  - A classification model M_i is derived from D_i
  - Its error rate is calculated using D_i as a test set
  - If a tuple is misclassified, its weight is increased, otherwise it is decreased
- Error rate: err($\mathbf{X_j}$) is the misclassification error of tuple $\mathbf{X_j}$. Classifier M_i error rate is the sum of the weights of the misclassified tuples:

$$error(M_i) = \sum_{j}^{d} w_j \times err(\mathbf{X_j})$$

- The weight of classifier M_i's vote is $\log \frac{1 - error(M_i)}{error(M_i)}$

Random Forest (Breiman 2001)

- Random Forest:
  - Each classifier in the ensemble is a *decision tree* classifier and is generated using a **random selection of attributes at each node** to determine the split
  - During classification, each tree votes and the most popular class is returned
- Two Methods to construct Random Forest:
  - Forest-RI (*random input selection*): Randomly select, at each node, F attributes as candidates for the split at the node. The CART methodology is used to grow the trees to maximum size
  - Forest-RC (*random linear combinations*)*:* Creates new attributes (or features) that are a linear combination of the existing attributes (reduces the correlation between individual classifiers)
- Comparable in accuracy to Adaboost, but **more robust to errors and outliers**
- Insensitive to the number of attributes selected for consideration at each split, and faster than bagging or boosting

General Random Forest Algorithm

Each tree is constructed using the following algorithm:

1. Let the number of training cases be *N*, and the number of variables in the classifier be *M*.
2. We are told the number *m* of input variables to be used to determine the decision at a node of the tree; *m* should be much less than *M*.
3. Choose a training set for this tree by choosing *n* times with replacement from all *N* available training cases (i.e. take a bootstrap sample). Use the rest of the cases to estimate the error of the tree, by predicting their classes.
4. For each node of the tree, randomly choose *m* variables on which to base the decision at that node. Calculate the best split based on these *m* variables in the training set.
5. Each tree is fully grown and not pruned (as may be done in constructing a normal tree classifier).

For prediction a new sample is pushed down the tree. It is assigned the label of the training sample in the terminal node it ends up in. This procedure is iterated over all trees in the ensemble, and the average vote of all trees is reported as random forest prediction.