20211012 Classification

Tuesday, October 12, 2021 4:04 PM

- Deliverable 0 is due Wednesday
  - PM will have instructions
  - There is a repo created for this class
  - PM will need to create it for you
  - Work with PM to work on structure of deliverable and any details
  - No template or structure, every project is a bit unique
- Done with unsupervised techniques for now, will see more towards the end of the semester
- Starting with classification, then regression, then neural networks

Classification

Boston University CS 506 - Lance Galletti

-

What is Classification?

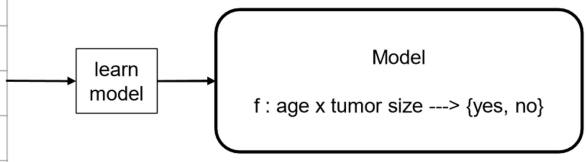
- Given a **training set** where data is labeled with a special **attribute** called a **class** (a discrete value)
- We want to find a **model** for the **class** attribute as a function of the values of the other attributes
- Goal: use this model on unlabeled data to assign a class as accurately as possible

---

- Attribute has a discrete number of values, these are the classifications
- Train your model on some examples

Example

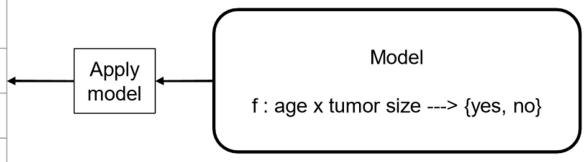
age	Tumor size	malignant?
20	10	no
30	15	yes
40	20	no
50	25	yes



- Class of interest: malignant
  - Two values, yes or no
- Predict based on age and tumor size

Example

age	Tumor size	malignant?
25	5	?
35	10	?
45	25	?



- Apply it to unlabeled samples
- If not discrete, then probably continuous label
  - Linear regression

Classification Tasks

- Predicting tumor cells as benign or malignant
- Classifying images
- Classifying credit card transactions as being legitimate or fraudulent
- Many more

Classification Techniques

- Instance-Based Classifiers
- Decision Trees
- Naive Bayes
- Support Vector Machines
- Neural Networks

- Will cover linear regression before Neural networks

Instance-Based Classifiers

- Use the stored training records to predict the class label of unseen cases
- Rote-learners:
  - Perform classification only if the attributes of the unseen record exactly match a record in our training set
- Nearest Neighbor:
  - Use the k closest records to perform classification

- Laziest method

- Do you have a record with the exact same attributes
- Rote-learners

- Little better: nearest neighbor
  - Find the closest k neighbors

Instance-Based Classifiers

age	Tumor size	malignant?
20	10	no
30	15	yes
40	20	no
50	25	yes

age	Tumor size	malignant?
25	5	?

- Find the closest records

K Nearest Neighbor Classifier

Requires:

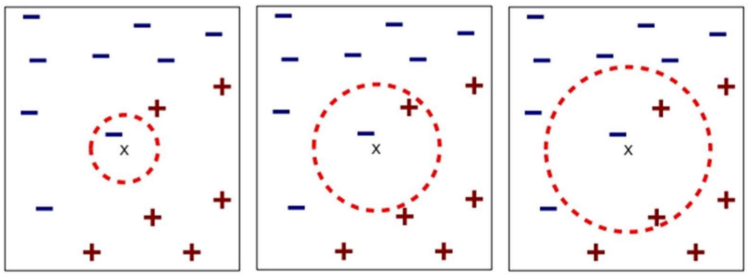
- Training set
- Distance function
- Value for k

How to classify an unseen record:

1. Compute distance of unseen record to all training records
2. Identify the k nearest neighbors
3. Aggregate the labels of these k neighbors to predict the unseen record class (ex: majority rule)

-

K Nearest Neighbor Classifier



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

- Depending on the value of K, could make severely different predictions

K Nearest Neighbor Classifier

Aggregation methods:

- Majority rule
- Weighted majority based on distance ($w = 1/d^2$)

Scaling issues:

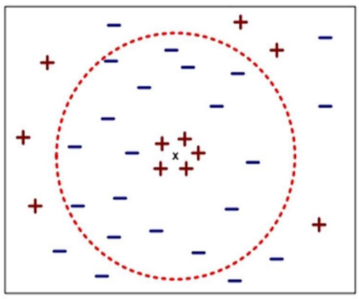
- Attributes should be scaled to prevent distance measures from being dominated by one attribute. Example:
  - Height: 1m -> 2m
  - Income: 10k -> 1million

- Inherent problem with distance functions:
  - Scaling variables
    - Do transforms that preserve the order
    - Mean center, normalize

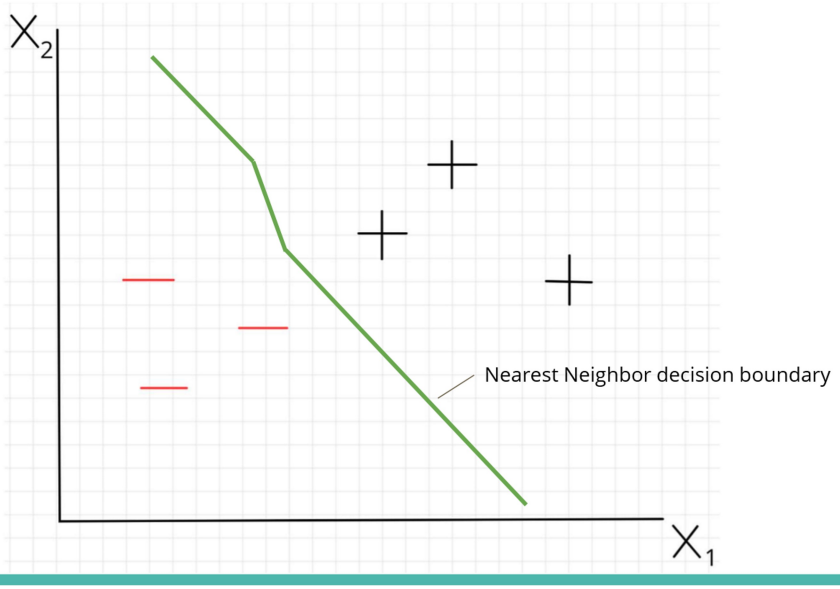
K Nearest Neighbor Classifier

Choosing the value of k:

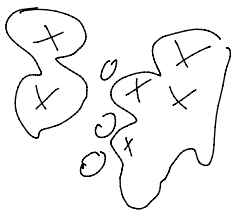
- If k is too small -> sensitive to noise points + overfitting (doesn't generalize well)
- If k is too big -> neighborhood may include points from other classes



- k small
- k large



Lots of points: overfitting



- If you have lots of points, you can overfit and get overly

specific decision boudnaries

K Nearest Neighbor Classifier

Pros:

- Simple to understand why a given unseen record was given a particular class
- Adapts to new attributes

Cons:

- Expensive to classify new points
- KNN can be problematic in high dimensions (curse of dimensionality)

- As long as you have match of attributes between unseen record and records in dataset
  - Can create subset of parameters: non-parametric
    - As opposed to linear regression, which cannot be missing attributes
- In high dimensions, density can be problematic
- **This is probably one of the simplest you can do, should be one of the first things you try on the midterm**
- If we have a lot of duplicates in the dataset
  - You want to know that the duplicates exist, so you can weigh them appropriately
    - Assign an ID, etc

20211012 Classification and Decision Trees

Tuesday, October 12, 2021 4:06 PM

- Midterm starts Friday
- There is already instructions
- There is starter code available Friday
- Start as early as possible

Classification and Decision Trees

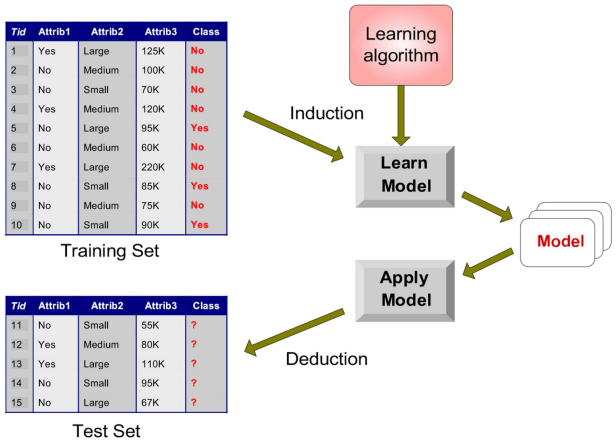
Slides by: Tan, Steinbach, Kumar

-

Classification: Definition

- Given a collection of records (*training set*)
  - Each record contains a set of *attributes*, one of the attributes is the *class*.
- Find a *model* for class attribute as a function of the values of other attributes.
- Goal: previously unseen records should be assigned a class as accurately as possible.
  - A *test set* is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.

Illustrating Classification Task

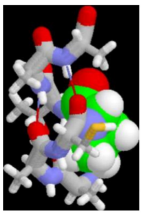


- Will go over a couple algorithms and a couple of different coefficients
  - If you are interested, read through all the slides
- How do you walk through a decision tree?

Examples of Classification Task

- Predicting tumor cells as benign or malignant
- Classifying credit card transactions as legitimate or fraudulent

- Classifying secondary structures of protein as alpha-helix, beta-sheet, or random coil



- Categorizing news stories as finance, weather, entertainment, sports, etc

Classification Techniques

- Decision Tree based Methods
- Neural Networks
- Naïve Bayes and Bayesian Belief Networks
- Support Vector Machines

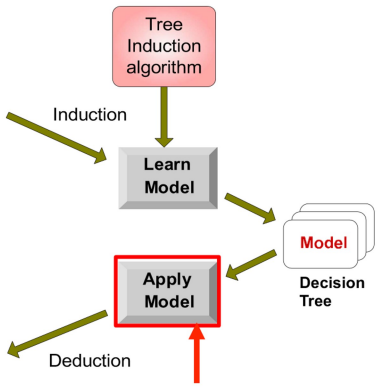
Decision Tree Classification Task

Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

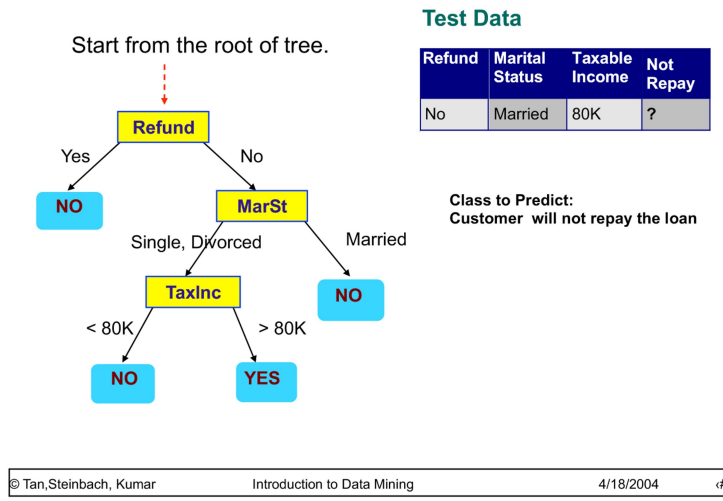
Training Set

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Test Set



Apply Model to Test Data



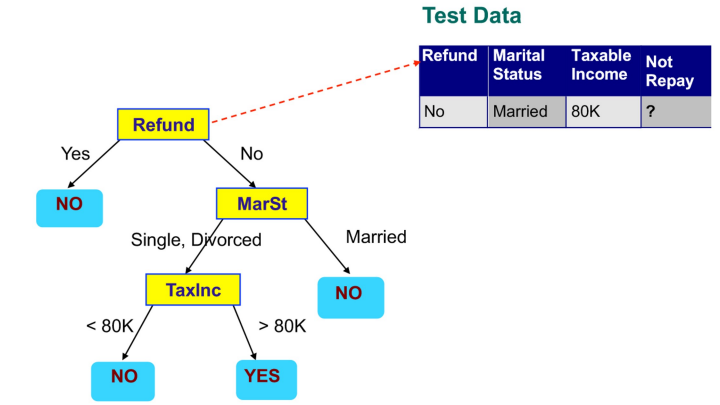
Test Data

Refund	Marital Status	Taxable Income	Not Repay
No	Married	80K	?

Class to Predict:
Customer will not repay the loan

- Unseen record on right that we want to classify using tree
- Refund: NO ->
  - Marital: Yes ->
    - RESULT: NO is classification prediction

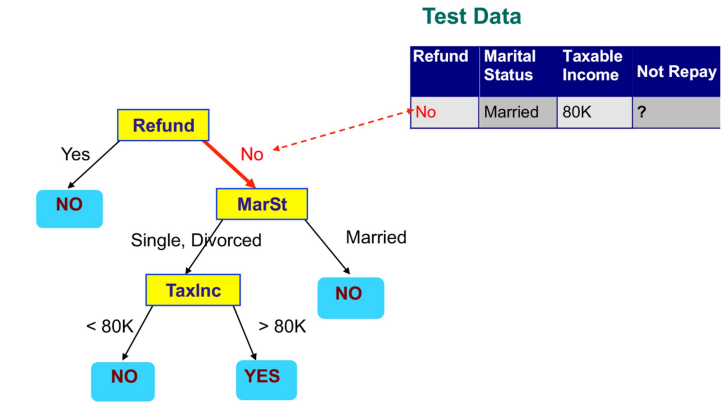
Apply Model to Test Data



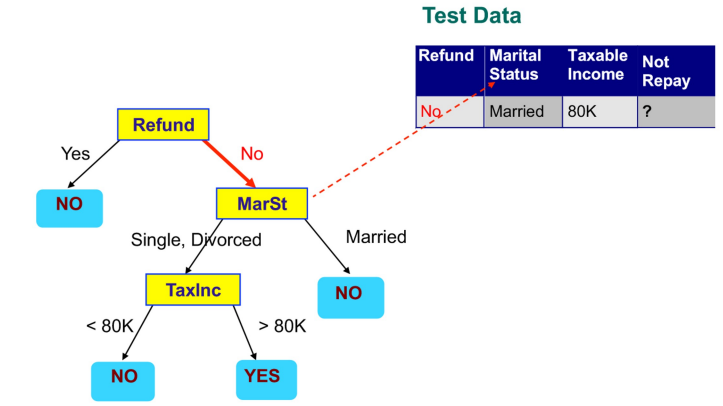
Test Data

Refund	Marital Status	Taxable Income	Not Repay
No	Married	80K	?

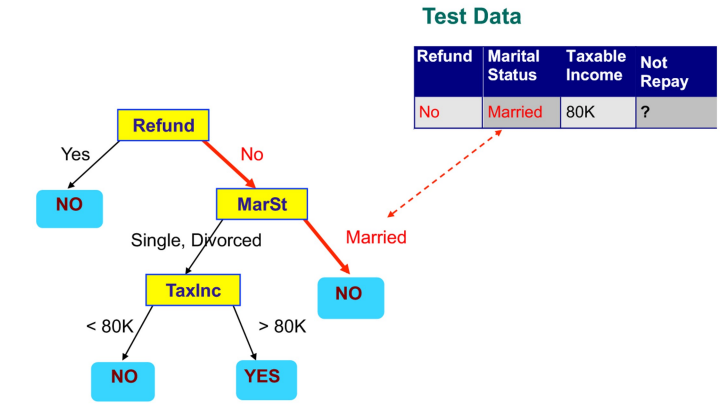
Apply Model to Test Data



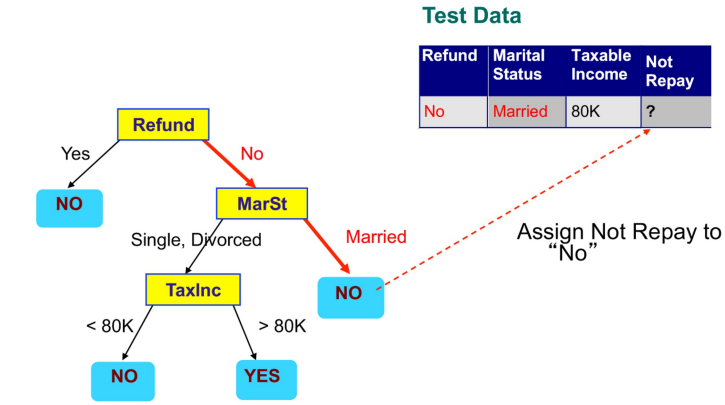
Apply Model to Test Data



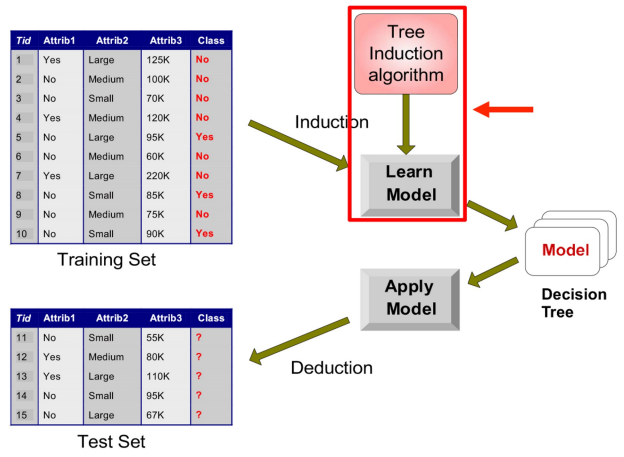
Apply Model to Test Data



Apply Model to Test Data



Decision Tree Classification Task



Tid	Attrib1	Attrib2	Attrib3	Class
1	Yes	Large	125K	No
2	No	Medium	100K	No
3	No	Small	70K	No
4	Yes	Medium	120K	No
5	No	Large	95K	Yes
6	No	Medium	60K	No
7	Yes	Large	220K	No
8	No	Small	85K	Yes
9	No	Medium	75K	No
10	No	Small	90K	Yes

Tid	Attrib1	Attrib2	Attrib3	Class
11	No	Small	55K	?
12	Yes	Medium	80K	?
13	Yes	Large	110K	?
14	No	Small	95K	?
15	No	Large	67K	?

Decision Tree Induction

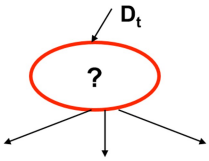
- Many Algorithms:
  - Hunt's Algorithm (one of the earliest)
  - CART
  - ID3, C4.5
  - SLIQ,SPRINT

- How to build this?
- Hunt's is one of the most intuitive,
  - Other trees are more efficient or already pruned for you

General Structure of Hunt's Algorithm

- Let D_t be the set of training records that reach a node t
- General Procedure:
  - If D_t contains records that belong the same class y_t, then t is a leaf node labeled as y_t
  - If D_t is an empty set, then t is a leaf node labeled by the default class, y_d
  - If D_t contains records that belong to more than one class, use an attribute test to split the data into smaller subsets. Recursively apply the procedure to each subset.

Tid	Refund	Marital Status	Taxable Income	Class
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes



- If you hear tree, think recursion
- First: specify terminating conditions
- Dt: set of training records at a specific node
- There was a specific training set for "yes" and a specific training set for "no"
- If Dt only has one class, then you are done
  - You have successfully split up your data
- If Dt is empty, can have a default class that handles this
  - e.g. If you had no records <$80K
    - You have no data that tells it what to be
      - Maybe there is a reasonable default you can use
- Meat of algorithm is the last point in slide
  - Want to avoid generating nodes that are 50-50 split of labels
  - Split on income 80K
    - <80k 1/2 say yes 1/2 say no
    - same for >80K
    - This is a bad split
  - Attribute test tells you what attribute to split on and how to split it

Tree Induction

- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.

- Issues
  - Determine how to split the records
    - How to specify the attribute test condition?
    - How to determine the best split?
  - Determine when to stop splitting

-

Tree Induction

- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.

- Issues
  - Determine how to split the records
    - How to specify the attribute test condition?
    - How to determine the best split?
  - Determine when to stop splitting

How to Specify Test Condition?

- Depends on attribute types
  - Nominal
  - Ordinal
  - Continuous

- Depends on number of ways to split
  - 2-way split
  - Multi-way split

Splitting Based on Nominal Attributes

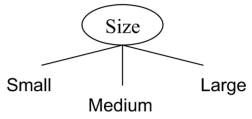
- Multi-way split: Use as many partitions as distinct values.

- Binary split: Divides values into two subsets. Need to find optimal partitioning.

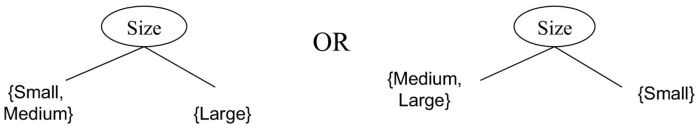
- We could do a multi-way split or a binary split
- Binary tree
  - May be a little simpler?

Splitting Based on Ordinal Attributes

- Multi-way split: Use as many partitions as distinct values.



- Binary split: Divides values into two subsets. Need to find optimal partitioning.



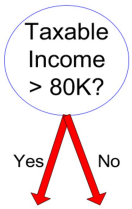
-

Splitting Based on Continuous Attributes

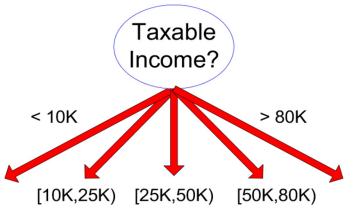
- Different ways of handling
  - Discretization to form an ordinal categorical attribute
    - Static – discretize once at the beginning
    - Dynamic – ranges can be found by equal interval bucketing, equal frequency bucketing (percentiles), or clustering.
  - Binary Decision: (A < v) or (A ≥ v)
    - consider all possible splits and finds the best cut
    - can be more compute intensive

- Discretize variables: create bins
  - This is hard to do
  - Maybe you do this in data pre-processing
- Other way is Binary decision

Splitting Based on Continuous Attributes



(i) Binary split

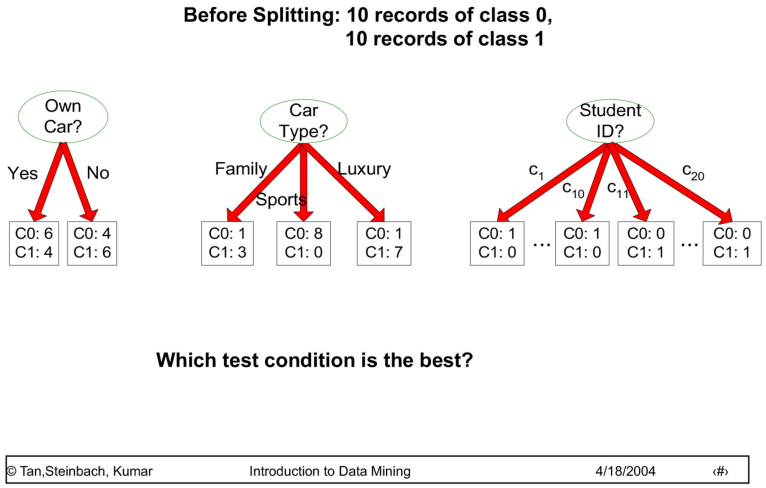


(ii) Multi-way split

Tree Induction

- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.
- Issues
  - Determine how to split the records
    - How to specify the attribute test condition?
    - How to determine the best split?
  - Determine when to stop splitting

How to determine the Best Split



- Want to avoid splits that generate some uncertainty
- Here are 3 examples, which are good splits?
  - There is no context for this data
  - Number of points in dataset in each class in each subset
  - 1st and 3rd are bad splits
    - 1st one is bad split: doesn't provide much info
      - Have basically same amount of uncertainty as previous layer of the tree
    - 3rd one is bad split:
      - Doesn't make sense to split on something that is unique to every record
  - 2nd is good split:
    - Reduced the uncertainty from 10 to 10 to something much more definitive

How to determine the Best Split

- Greedy approach:
  - Nodes with homogeneous class distribution are preferred
- Need a measure of node impurity:

Non-homogeneous,
High degree of impurity

Homogeneous,
Low degree of impurity

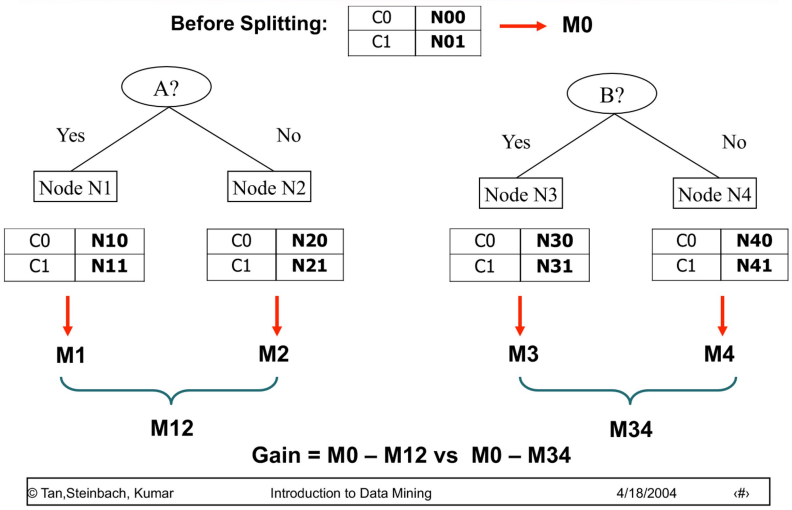
- Define function to mathematically test what is a good split and what is a bad split
- Prefer making the split on the right rather than the split on the left

Measures of Node Impurity

- Gini Index
- Entropy
- Misclassification error

- Take a look at entropy and misclassification on your own, basically the same thing at different scales

How to Find the Best Split



- Want more purity after split
- Want to know how much impurity did you lose after doing a particular split
  - Now you can compare different splits
- M0: node impurity
  - Attribute A splits on a number of nodes
    - Aggregate all into some sort of metric
    - Then we can compare to M0
- Coding this up is not simple
  - There is a lot to keep track of
  - Conceptually, may be easy to understand the process

Measure of Impurity: GINI

- Gini Index for a given node t :

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

(NOTE: $p(j \mid t)$ is the relative frequency of class j at node t).

  - Maximum (1 - $1/n_c$) when records are equally distributed among all classes, implying least interesting information
  - Minimum (0.0) when all records belong to one class, implying most interesting information

C1	0
C2	6
Gini=0.000	

C1	1
C2	5
Gini=0.278	

C1	2
C2	4
Gini=0.444	

C1	3
C2	3
Gini=0.500	

- Want to know what is the relative frequency of class j at node t

- GINI will be maximal when there is even frequency in each class

Examples for computing GINI

$$GINI(t) = 1 - \sum_j [p(j \mid t)]^2$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Gini = 1 – P(C1)² – P(C2)² = 1 – 0 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Gini = 1 – (1/6)² – (5/6)² = 0.278

C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

Gini = 1 – (2/6)² – (4/6)² = 0.444

- 1 - (sum of breakdowns within node)
- Seems to align with our intution
- Gini score of 0: good, pure node
- Gini score of 0.5: bad, very impure node

Splitting Based on GINI

- Used in CART, SLIQ, SPRINT.
- When a node p is split into k partitions (children), the quality of split is computed as,

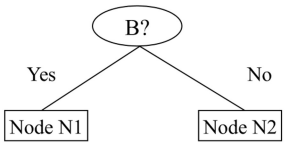
$$GINI_{split} = \sum_{i=1}^{k} \frac{n_i}{n} GINI(i)$$

where, n_i = number of records at child i,
n = number of records at node p.

- GINI split is a weighted sum
  - Weighted by the number of records in that node
- Measure impurity for a given node, then we have a split
- We want to split off a particular value

Binary Attributes: Computing GINI Index

- Splits into two partitions
- Effect of Weighing partitions:
  - Larger and Purer Partitions are sought for.



	Parent
C1	6
C2	6
Gini = 0.500	

Gini(N1)
= 1 – (5/7)² – (2/7)²
= 0.4

Gini(N2)
= 1 – (1/5)² – (4/5)²
= 0.32

	N1	N2
C1	5	1
C2	2	4
Gini=0.3666		

Gini(Children)
= 7/12 * 0.4+
5/12 * 0.32
= 0.3666

- Example. Parent GINI is 0.5
- Combine GINI of node 1 and node 2 to get the GINI of the split
- The gain is the GINI of the parent minus the GINI of the child

Categorical Attributes: Computing Gini Index

- For each distinct value, gather counts for each class in the dataset
- Use the count matrix to make decisions

Multi-way split

	CarType		
	Family	Sports	Luxury
C1	1	2	1
C2	4	1	1
Gini	0.393		

Two-way split
(find best partition of values)

	CarType	
	{Sports, Luxury}	{Family}
C1	3	1
C2	2	4
Gini	0.400	

	CarType	
	{Sports}	{Family, Luxury}
C1	2	2
C2	1	5
Gini	0.419	

-

Continuous Attributes: Computing Gini Index

- Use Binary Decisions based on one value
- Several Choices for the splitting value
  - Number of possible splitting values = Number of distinct values
- Each splitting value has a count matrix associated with it
  - Class counts in each of the partitions, A < v and A ≥ v
- Simple method to choose best v
  - For each v, scan the database to gather count matrix and compute its Gini index
  - Computationally Inefficient! Repetition of work.

Tid	Refund	Marital Status	Taxable Income	Not Repay
1	Yes	Single	125K	No
2	No	Married	100K	No
3	No	Single	70K	No
4	Yes	Married	120K	No
5	No	Divorced	95K	Yes
6	No	Married	60K	No
7	Yes	Divorced	220K	No
8	No	Single	85K	Yes
9	No	Married	75K	No
10	No	Single	90K	Yes

Continuous Attributes: Computing Gini Index...

- For efficient computation: for each attribute,
  - Sort the attribute on values
  - Linearly scan these values, each time updating the count matrix and computing gini index
  - Choose the split position that has the least gini index

Cheat	No		No		No		Yes		Yes		Yes		No		No		No		No	
Taxable Income																				
Sorted Values →	60		70		75		85		90		95		100		120		125		220	
Split Positions →	55		65		72		80		87		92		97		110		122		172		230	
	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>	<=	>
Yes	0	3	0	3	0	3	0	3	1	2	2	1	3	0	3	0	3	0	3	0	3	0
No	0	7	1	6	2	5	3	4	3	4	3	4	3	4	4	3	5	2	6	1	7	0
Gini	0.420		0.400		0.375		0.343		0.417		0.400		*0.300*		0.343		0.375		0.400		0.420	

- Can make this more efficient by ordering data
  - Don't need to keep re-computing the count, just do incremental changes
- Split based on 97 to get the lowest GINI

Alternative Splitting Criteria based on INFO

- Entropy at a given node t:

$$Entropy(t) = -\sum_j p(j \mid t) \log p(j \mid t)$$

(NOTE: $p(j \mid t)$ is the relative frequency of class j at node t).

- Measures homogeneity of a node.
  - Maximum (log n_c) when records are equally distributed among all classes implying least information
  - Minimum (0.0) when all records belong to one class, implying most information
- Entropy based computations are similar to the GINI index computations

- Entropy is a different metric

Examples for computing Entropy

$$Entropy(t) = -\sum_j p(j \mid t) \log_2 p(j \mid t)$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Entropy = – 0 log 0 – 1 log 1 = – 0 – 0 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Entropy = – (1/6) $\log_2$ (1/6) – (5/6) $\log_2$ (1/6) = 0.65

C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

Entropy = – (2/6) $\log_2$ (2/6) – (4/6) $\log_2$ (4/6) = 0.92

Splitting Based on INFO...

- Information Gain:

$$GAIN_{split} = Entropy(p) - \left(\sum_{i=1}^{k} \frac{n_i}{n} Entropy(i) \right)$$

Parent Node, p is split into k partitions;
n_i is number of records in partition i

  - Measures Reduction in Entropy achieved because of the split. Choose the split that achieves most reduction (maximizes GAIN)
  - Used in ID3 and C4.5
  - Disadvantage: Tends to prefer splits that result in large number of partitions, each being small but pure.

Splitting Based on INFO...

- Gain Ratio:

$$GainRATIO_{split} = \frac{GAIN_{Split}}{SplitINFO} \qquad SplitINFO = -\sum_{i=1}^{k} \frac{n_i}{n} \log \frac{n_i}{n}$$

Parent Node, p is split into k partitions
n_i is the number of records in partition i

  - Adjusts Information Gain by the entropy of the partitioning (SplitINFO). Higher entropy partitioning (large number of small partitions) is penalized!
  - Used in C4.5
  - Designed to overcome the disadvantage of Information Gain

Splitting Criteria based on Classification Error

- Classification error at a node t :

$$Error(t) = 1 - \max_i P(i \mid t)$$

- Measures misclassification error made by a node.
  - Maximum (1 - 1/n_c) when records are equally distributed among all classes, implying least interesting information
  - Minimum (0.0) when all records belong to one class, implying most interesting information

Examples for Computing Error

$$Error(t) = 1 - \max_i P(i \mid t)$$

C1	**0**
C2	**6**

P(C1) = 0/6 = 0 P(C2) = 6/6 = 1

Error = 1 – max (0, 1) = 1 – 1 = 0

C1	**1**
C2	**5**

P(C1) = 1/6 P(C2) = 5/6

Error = 1 – max (1/6, 5/6) = 1 – 5/6 = 1/6

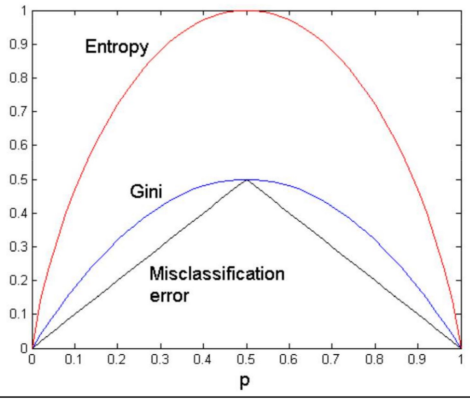
C1	**2**
C2	**4**

P(C1) = 2/6 P(C2) = 4/6

Error = 1 – max (2/6, 4/6) = 1 – 4/6 = 1/3

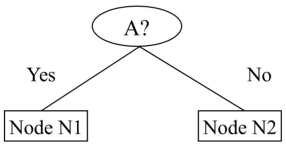
Comparison among Splitting Criteria

For a 2-class problem:



- Misclassification error has the same max as Gini
- All kind of conveying the same thing

Misclassification Error vs Gini



	Parent
C1	7
C2	3
Gini = 0.42	

Gini(N1)
= 1 – (3/3)² – (0/3)²
= 0

Gini(N2)
= 1 – (4/7)² – (3/7)²
= 0.489

	N1	N2
C1	3	4
C2	0	3
Gini=0.361		

Gini(Children)
= 3/10 * 0
+ 7/10 * 0.489
= 0.342

Gini improves !!

Tree Induction

- Greedy strategy.
  - Split the records based on an attribute test that optimizes certain criterion.
- Issues
  - Determine how to split the records
    - How to specify the attribute test condition?
    - How to determine the best split?
  - Determine when to stop splitting

Stopping Criteria for Tree Induction

- Stop expanding a node when all the records belong to the same class
- Stop expanding a node when all the records have similar attribute values
- Early termination (to be discussed later)

- There are times that you may want to terminate the algorithm early
  - Perhaps to avoid overfitting
    - Don't want to go down to a single point per final leaf
    - May improve prediction
  - Save computation time

Decision Tree Based Classification

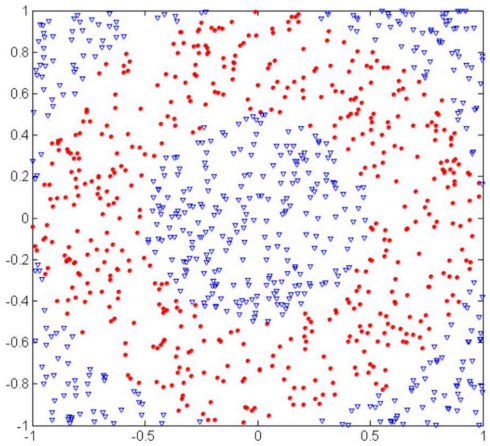
- Advantages:
  - Inexpensive to construct
  - Extremely fast at classifying unknown records
  - Easy to interpret for small-sized trees
  - Accuracy is comparable to other classification techniques for many simple data sets

- More user friendly than some models
  - This is why we are classifying this particular unseen record

Practical Issues of Classification

- Underfitting and Overfitting
- Missing Values
- Costs of Classification

Underfitting and Overfitting (Example)



500 circular and 500 triangular data points.

Circular points:

$0.5 \le \text{sqrt}(x_1^2+x_2^2) \le 1$

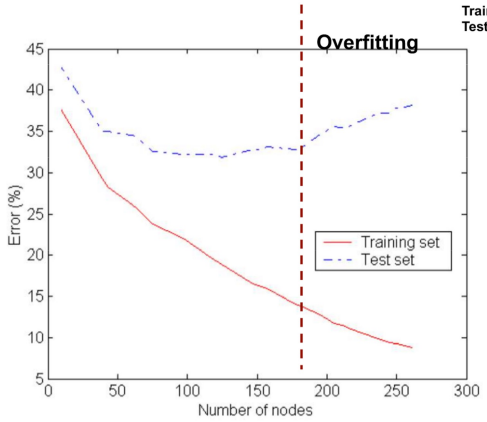
Triangular points:

$\text{sqrt}(x_1^2+x_2^2) > 0.5$ or

$\text{sqrt}(x_1^2+x_2^2) < 1$

- Want to separate blue class from red class

Underfitting and Overfitting

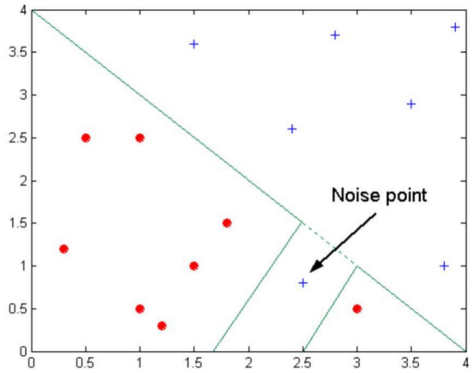


Training set represents the training error
Test set the error on the test data

Underfitting: when model is too simple, both training and test errors are large

- Could measure training set accuracy as your split the data
- But if we then test the test set
  - As we increased the number of nodes, it generally got better until we reach the point where the model is too specific to training set and is too specific
    - This is something you want to avoid
- Will talk about midterm a bit more maybe next time
- How do we catch overfitting?

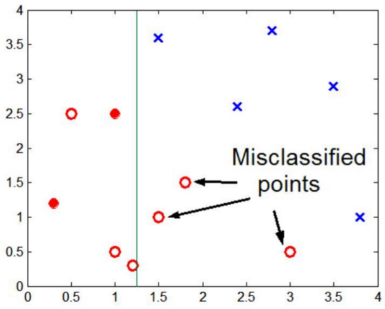
Overfitting due to Noise



Decision boundary is distorted by noise point

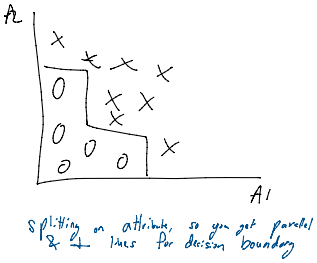
Testing, training, and validating?

Overfitting due to Insufficient Examples



Lack of data points in the lower half of the diagram makes it difficult to predict correctly the class labels of that region

- Insufficient number of training records in the region causes the decision tree to predict the test examples using other training records that are irrelevant to the classification task



Notes on Overfitting

- Overfitting results in decision trees that are more complex than necessary
- Training error no longer provides a good estimate of how well the tree will perform on previously unseen records
- Need new ways for estimating errors

Estimating Generalization Errors

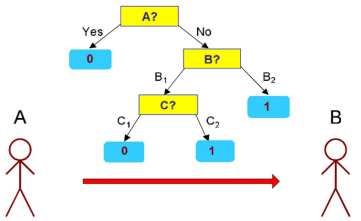
- Re-substitution errors: error on training (Σ e(t))
- Generalization errors: error on testing (Σ e'(t))
- Methods for estimating generalization errors:
  - Optimistic approach: e'(t) = e(t)
  - Pessimistic approach:
    - For each leaf node: e'(t) = (e(t)+0.5)
    - Total errors: e'(T) = e(T) + N × 0.5 (N: number of leaf nodes)
    - For a tree with 30 leaf nodes and 10 errors on training (out of 1000 instances):
      Training error = 10/1000 = 1%
      Generalization error = (10 + 30×0.5)/1000 = 2.5%
  - Reduced error pruning (REP):
    - uses validation data set to estimate generalization error

Occam's Razor

- Given two models of similar generalization errors, one should prefer the simpler model over the more complex model
- For complex models, there is a greater chance that it was fitted accidentally by errors in data
- Therefore, one should include model complexity when evaluating a model

Minimum Description Length (MDL)

X	y
X_1	1
X_2	0
X_3	0
X_4	1
…	…
X_n	1



X	y
X_1	?
X_2	?
X_3	?
X_4	?
…	…
X_n	?

- Cost(Model,Data) = Cost(Data|Model) + Cost(Model)
  - Cost is the number of bits needed for encoding.
  - Search for the least costly model.
- Cost(Data|Model) encodes the misclassification errors.
- Cost(Model) uses node encoding (number of children) plus splitting condition encoding.

How to Address Overfitting

- Pre-Pruning (Early Stopping Rule)
  - Stop the algorithm before it becomes a fully-grown tree
  - Typical stopping conditions for a node:
    - Stop if all instances belong to the same class
    - Stop if all the attribute values are the same
  - More restrictive conditions:
    - Stop if number of instances is less than some user-specified threshold
    - Stop if class distribution of instances are independent of the available features (e.g., using χ^2 test)
    - Stop if expanding the current node does not improve impurity measures (e.g., Gini or information gain).

How to Address Overfitting…

- Post-pruning
  - Grow decision tree to its entirety
  - Trim the nodes of the decision tree in a bottom-up fashion
  - If generalization error improves after trimming, replace sub-tree by a leaf node.
  - Class label of leaf node is determined from majority class of instances in the sub-tree
  - Can use MDL for post-pruning

Example of Post-Pruning

Class = Yes	20
Class = No	10
Error = 10/30	

Training Error (Before splitting) = 10/30

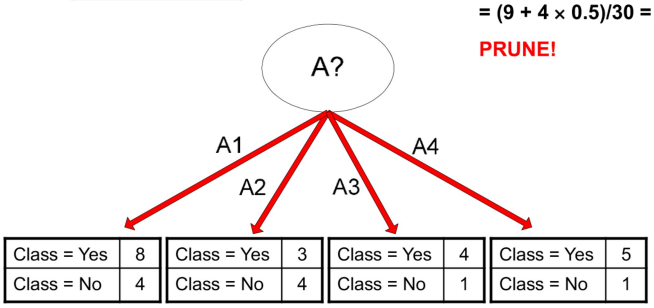
Pessimistic error = (10 + 0.5)/30 = 10.5/30

Training Error (After splitting) = 9/30

Pessimistic error (After splitting)

= (9 + 4 × 0.5)/30 = 11/30

PRUNE!



Examples of Post-pruning

- Optimistic error?

  Don't prune for both cases

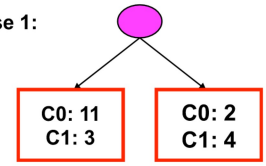
- Pessimistic error?

  Don't prune case 1, prune case 2

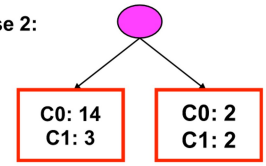
- Reduced error pruning?

  Depends on validation set

Case 1:



Case 2:



Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?
- Methods for Performance Evaluation
  - How to obtain reliable estimates?
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

- Model evaluation and pruning

Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?
- Methods for Performance Evaluation
  - How to obtain reliable estimates?
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Metrics for Performance Evaluation

- Focus on the predictive capability of a model
  - Rather than how fast it takes to classify or build models, scalability, etc.
- Confusion Matrix:

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a	b
	Class=No	c	d

a: TP (true positive)
b: FN (false negative)
c: FP (false positive)
d: TN (true negative)

Metrics for Performance Evaluation…

		PREDICTED CLASS	
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a (TP)	b (FN)
	Class=No	c (FP)	d (TN)

- Most widely-used metric:

$$\text{Accuracy} = \frac{a+d}{a+b+c+d} = \frac{TP+TN}{TP+TN+FP+FN}$$

Limitation of Accuracy

- Consider a 2-class problem
  - Number of Class 0 examples = 9990
  - Number of Class 1 examples = 10

- If model predicts everything to be class 0, accuracy is 9990/10000 = 99.9 %
  - Accuracy is misleading because model does not detect any class 1 example

- Didn't learn anything by classifying everything as Class 0

Cost Matrix

		PREDICTED CLASS	
	C(i|j)	**Class=Yes**	**Class=No**
ACTUAL CLASS	**Class=Yes**	C(Yes|Yes)	C(No|Yes)
	Class=No	C(Yes|No)	C(No|No)

C(i|j): Cost of misclassifying class j example as class i

- Assign a particular cost to each of these

Computing Cost of Classification

Cost Matrix	PREDICTED CLASS		
	C(i|j)	+	-
ACTUAL CLASS	+	-1	100
	-	1	0

Model M_1	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	150	40
	-	60	250

Model M_2	PREDICTED CLASS		
		+	-
ACTUAL CLASS	+	250	45
	-	5	200

Accuracy = 80%
Cost = 3910

Accuracy = 90%
Cost = 4255

- On the left, the accuracy is less but the cost is better
- Maybe you really want to penalize false positives

Cost vs Accuracy

Count	PREDICTED CLASS		
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	a	b
	Class=No	c	d

Accuracy is proportional to cost if
1. C(Yes|No)=C(No|Yes) = q
2. C(Yes|Yes)=C(No|No) = p

N = a + b + c + d

Accuracy = (a + d)/N

Cost	PREDICTED CLASS		
		Class=Yes	Class=No
ACTUAL CLASS	Class=Yes	p	q
	Class=No	q	p

Cost = p (a + d) + q (b + c)
= p (a + d) + q (N – a – d)
= q N – (q – p)(a + d)
= N [q – (q-p) × Accuracy]

Cost-Sensitive Measures

$$\text{Precision (p)} = \frac{a}{a+c}$$

$$\text{Recall (r)} = \frac{a}{a+b}$$

$$\text{F - measure (F)} = \frac{2rp}{r+p} = \frac{2a}{2a+b+c}$$

- Precision is biased towards C(Yes|Yes) & C(Yes|No)
- Recall is biased towards C(Yes|Yes) & C(No|Yes)
- F-measure is biased towards all except C(No|No)

$$\text{Weighted Accuracy} = \frac{w_1 a + w_4 d}{w_1 a + w_2 b + w_3 c + w_4 d}$$

- Be transparent on what your model is good on and what your model is bad at

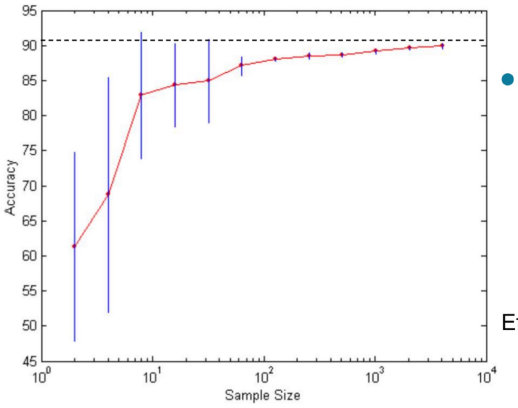
Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?
- Methods for Performance Evaluation
  - How to obtain reliable estimates?
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

Methods for Performance Evaluation

- How to obtain a reliable estimate of performance?
- Performance of a model may depend on other factors besides the learning algorithm:
  - Class distribution
  - Cost of misclassification
  - Size of training and test sets

Learning Curve



- Learning curve shows how accuracy changes with varying sample size
- Requires a sampling schedule for creating learning curve:
  - Arithmetic sampling (Langley, et al)
  - Geometric sampling (Provost et al)

Effect of small sample size:

- Bias in the estimate
- Variance of estimate

Methods of Estimation

- Holdout
  - Reserve 2/3 for training and 1/3 for testing
- Random subsampling
  - Repeated holdout
- Cross validation
  - Partition data into k disjoint subsets
  - k-fold: train on k-1 partitions, test on the remaining one
  - Leave-one-out: k=n
- Stratified sampling
  - oversampling vs undersampling
- Bootstrap
  - Sampling with replacement

Model Evaluation

- Metrics for Performance Evaluation
  - How to evaluate the performance of a model?
- Methods for Performance Evaluation
  - How to obtain reliable estimates?
- Methods for Model Comparison
  - How to compare the relative performance among competing models?

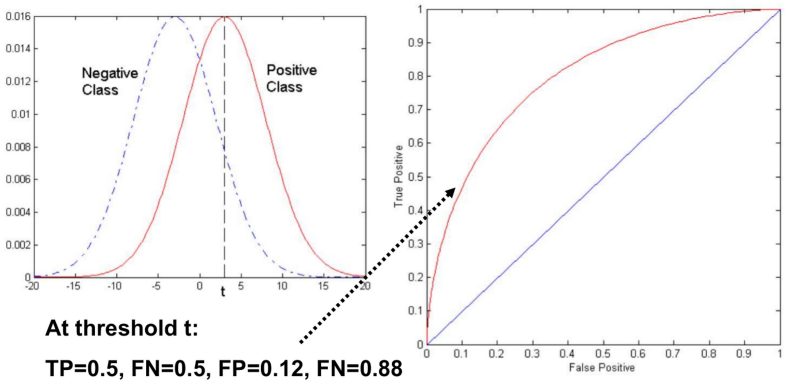
ROC (Receiver Operating Characteristic)

- Developed in 1950s for signal detection theory to analyze noisy signals
  - Characterize the trade-off between positive hits and false alarms
- ROC curve plots TP (on the y-axis) against FP (on the x-axis)
- Performance of each classifier represented as a point on the ROC curve
  - changing the threshold of algorithm, sample distribution or cost matrix changes the location of the point

ROC Curve

- 1-dimensional data set containing 2 classes (positive and negative)

- any points located at x > t is classified as positive



At threshold t:

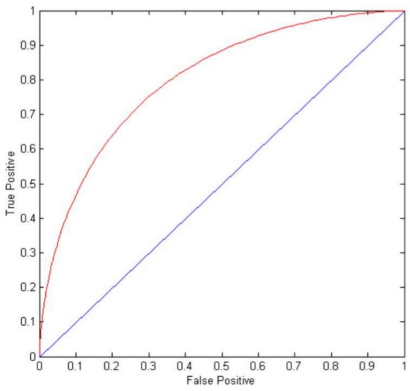
TP=0.5, FN=0.5, FP=0.12, FN=0.88

ROC Curve

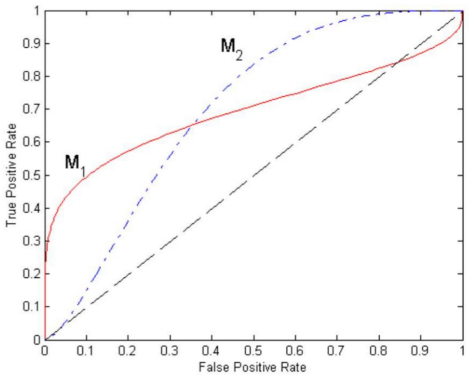
(TP,FP):

- (0,0): declare everything to be negative class
- (1,1): declare everything to be positive class
- (1,0): ideal

- Diagonal line:
  - Random guessing
  - Below diagonal line:
    - prediction is opposite of the true class



Using ROC for Model Comparison



- No model consistently outperform the other
  - M_1 is better for small FPR
  - M_2 is better for large FPR
- Area Under the ROC curve
  - Ideal:
    - Area = 1
  - Random guess:
    - Area = 0.5

How to Construct an ROC curve

Instance	P(+|A)	True Class
1	0.95	+
2	0.93	+
3	0.87	-
4	0.85	-
5	0.85	-
6	0.85	+
7	0.76	-
8	0.53	+
9	0.43	-
10	0.25	+

- Use classifier that produces posterior probability for each test instance P(+|A)
- Sort the instances according to P(+|A) in decreasing order
- Apply threshold at each unique value of P(+|A)
- Count the number of TP, FP, TN, FN at each threshold
- TP rate, TPR = TP/(TP+FN)
- FP rate, FPR = FP/(FP + TN)