Lesson 8 KNN

Outline

- One more classification algorithm: KNN
- Evaluation metrics: How can we measure accuracy?
- Evaluating a classifier's accuracy:
  - Holdout method
  - Cross-validation
  - Bootstrap
- One technique to compare classifiers' performances:
  - ROC Curves

Eager Learners vs Lazy Learners

- Eager learners, when given a set of training tuples, will construct a generalization model before receiving test tuples to classify.
- Lazy learners simply store data (or do only a little minor processing) and wait until given a test tuple.
- Lazy learner: less time in training but more in predicting.
- Question: Are Naïve Bayes method and Decision Tree algorithms lazy or eager learners?
  - Answer: They are eager learners.

Lazy Learner

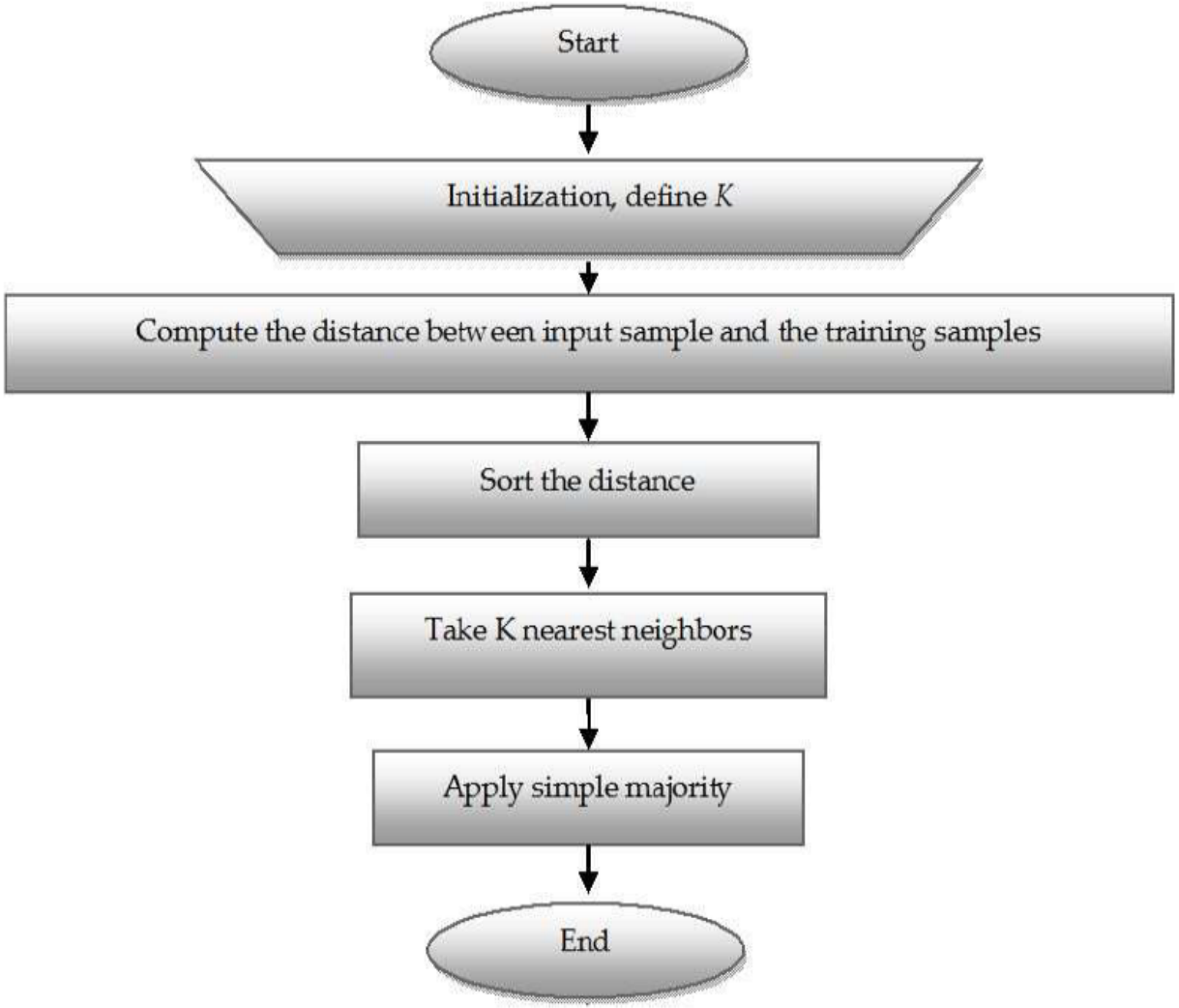
- Also called instance based learner.
- K-nearest-neighbor classifier (KNN) is a lazy learner.
  - The class label of a new tuple is determined by the class labels of K nearest neighbor tuples, by majority voting.

KNN Algorithm: Pseudo-code

Algorithm KNN(X, Y, K, s)
Input training data X, class labels Y of X,
K neighbors to consider, unknown sample s
Output class label for s

for i <- 1 **to** |X| **do**
compute distance $d(X_i , s)$
I <- indices for the K smallest distances $d(X_i , s)$
return majority label for $\{Y_i \mid$ where i is in I$\}$

KNN Algorithm



Distance (Dissimilarity) Measures

- Manhattan (city block) distance
  - consider two data items, A and B, each with two attribute values, (x1, y1) and (x2, y2), respectively.

$$d(A,B) = |x1 - x2| + |y1 - y2|$$

- Euclidean distance
  - consider two data items, A and B, each with two attribute values, (x1, y1) and (x2, y2), respectively.

$$d(A,B) = \sqrt{|x1 - x2|^2 + |y1 - y2|^2}$$

- These are special cases of Minkowski distance
  - In general, the Minkowski distance between the two p-dimensional data items is given by the following formula, where *q* is a positive integer. (x1, y1, z1 … are the attribute values.) Manhattan: q = 1. Euclidean: q = 2.

$$d(A,B) = \sqrt[q]{|x1 - x2|^q + |y1 - y2|^q + |z1 - z2|^q + \cdots}$$

Distance Measures Examples

	age	height	weight	salary
P1	20	65	160	50000
P2	25	70	180	40000
P3	30	60	200	70000
P4	35	65	170	125000
P5	40	75	150	100000

- The Euclidean distance between P2 and P1 is calculated as follows:
  - $d(P2, P1) = \sqrt{|20-25|^2 + |65-70|^2 + |160-180|^2 + |50000-40000|^2}$
    $= 10000.02$
- The Manhattan distance between P2 and P1 is calculated as follows:
  - $d(P2, P1) = |20\text{-}25| + |65\text{-}70| + |160\text{-}180| + |50000\text{-}40000| = 10030$

Normalization

- Distance between neighbors could be dominated by some attributes with wider range of values. For example, if your dataset has just two attributes, X and Y, and X has values that range from 1 to 1000, while Y has values that only go from 1 to 100, then Y's influence on the distance function will usually be overpowered by X's influence.
- *Min-max normalization* can be used to transform a value v of a numeric attribute A to v' in the range [0,1] by computing

$$v' = \frac{v - minA}{max_A - minA}$$

- What if attributes are categorical?
  - Main idea is turn categorical data to numeric data. See the post below:
  - https://www.quora.com/How-can-I-use-KNN-for-mixed-data-categorical-and-numerical

Worked Example

- Consider a dataset having two variables: height and weight. Each sample is classified as N (Normal) or U (Underweight).

weight	height	class
51	167	U
62	182	N
69	176	N
64	173	N
65	172	N
56	174	U
58	169	N
57	173	N
55	170	N

Worked Example

- On the basis of the given data we want to classify a data sample with weight 57 and height 170.

weight	height	class
51	167	U
62	182	N
69	176	N
64	173	N
65	172	N
56	174	U
58	169	N
57	173	N
55	170	N

Worked Example

- Compute Euclidean distance of unknown data point from all points as shown below.

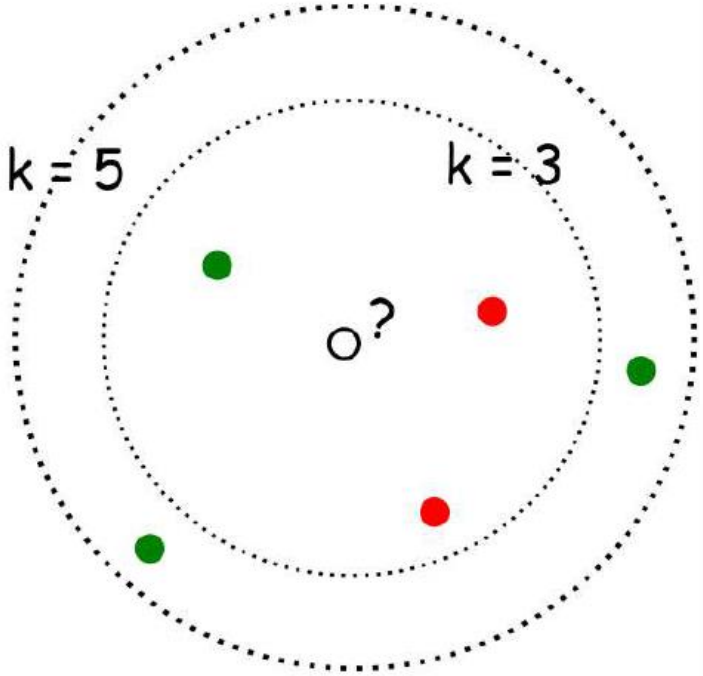
weight	height	class	distance
51	167	U	6.71
62	182	N	13.00
69	176	N	13.42
64	173	N	7.62
65	172	N	8.25
56	174	U	4.12
58	169	N	1.41
57	173	N	3.00
55	170	N	2.00

Worked Example

- Suppose we use K=3. The last three data points are nearest neighbors. The majority of neighbors point towards 'Normal'. So the KNN algorithm will classify (57, 170) as 'Normal'.

weight	height	class	distance
51	167	U	6.71
62	182	N	13.00
69	176	N	13.42
64	173	N	7.62
65	172	N	8.25
56	174	U	4.12
58	169	N	1.41
57	173	N	3.00
55	170	N	2.00

Issues about K value



- When K = 3, the new object will be classified as red.
- When K = 5, the new object will be classified as green.

How to choose K?

- If K is too small it is sensitive to noise points.
- Larger K works well but computational costly.
- Commonly used K:
  - K=sqrt(n) where n is the total number of data points.
  - Choose an odd value of K for 2 class problem.

R code

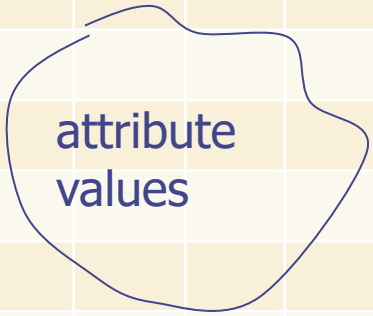
- knn(train, test, cl, k)
  - *train*: matrix or data frame of training set cases.
  - *test*: matrix or data frame of test set cases.
  - *cl*: factor of true classifications of training set.
  - *k*: number of neighbors considered.
- See the demo code.

Summary on KNN

- Strengths
  - Very simple and intuitive.
  - Good classification if the number of samples is large enough.
- Weaknesses
  - Takes more time to classify a new example.
    - need to calculate and compare distance from new example to all other examples.
  - Choosing K may be tricky.
  - Need large number of samples for accuracy.
  - Irrelevant features and noise can be very detrimental

Model Evaluation

Id	A1	A2	A3	A4	Actual Class	Predicted Class	
1					Y	Y	TP
2					Y	N	FN
3					N	N	TN
4					Y	N	FN
5					N	N	TN
6					N	N	TN
7					N	Y	FP
8					Y	Y	TP
9					N	N	TN
10					N	N	TN



Assume:
Y is positive
N is negative

7 out of 10 were correctly classified:

accuracy = 70%

Classifier Evaluation Metrics: Confusion Matrix

Actual class\Predicted class	Positive	Negative
Positive	**True Positives (TP)**	**False Negatives (FN)**
Negative	**False Positives (FP)**	**True Negatives (TN)**

- May have extra rows/columns to provide totals
- Confusion matrix of the example in the previous slide:

Actual class / Predicted class	Y	N	Total
Y	2	2	4
N	1	5	6
Total	3	7	10

Accuracy and Error Rate

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

C: buys_computer = yes (P)
¬C: buys_computer = no (N)

- Classifier Accuracy: percentage of test set tuples that are correctly classified

  Accuracy = (TP + TN)/All

- Error rate: *1 – accuracy*, or

  Error rate = (FP + FN)/All

Sensitivity and Specificity

Class Imbalance Problem:

- One class may occur infrequently, e.g. fraud, or HIV-positive. In that case, the main class of interest (the positive class) is represented by a very small fraction of dataset.
- The model below has high overall accuracy but low "sensitivity".
- Example: 9900 N's (no cancer) and 100 P's (cancer)
  - A model correctly classifies all N's and 20 P's
  - Model accuracy = 9920 / 10000 = 99.2% => very high
  - But, 80% of cancer patients were misdiagnosed as non-cancer patient – therefore, low (20%) "sensitivity".

Sensitivity and Specificity

- If it is important to correctly diagnose cancer patients, we need a different measure. In this example, we need TP/P.

- *Sensitivity*: True Positive recognition rate (also called recall)

  Sensitivity = TP/P

- *Specificity*: True Negative recognition rate

  Specificity = TN/N

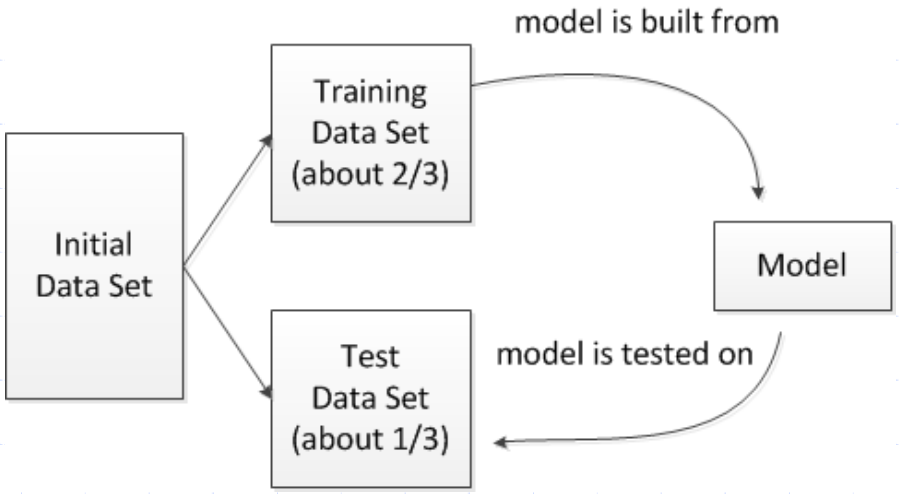
Precision and F-measure

- Closely related to sensitivity and specificity is another performance measure – Precision.
- Precision = TP/(TP+FP)
- Remember from last slide: Recall=Sensitivity = TP/P
- A measure of model performance that combines precision and recall into a single number is known as F-measure. The F-measure combines precision and recall using the harmonic mean, a type of average that is used for rate of change.

$$F\text{-}measure = \frac{2 * precision * recall}{recall + precision}$$

Evaluating Classifier Accuracy: Holdout Method

- Given data is randomly partitioned into two independent sets
  - Training set (e.g., 2/3) for model construction
  - Test set (e.g., 1/3) for accuracy estimation



- A model is built from the training set and it is tested on the test set. The accuracy is computed as the number of tuples that are correctly classified divided by the total number of test set tuples.

Evaluating Classifier Accuracy: k-fold Cross-Validation Method

- Randomly partition the data into k mutually exclusive subsets, D_1, D_2, …, D_k, each approximately equal size (k = 10 is most popular)
- At i-th iteration, use D_i as test set and others as training set

  Iter. 1: {D_2, …, D_{10}} used for training, D_1 used for testing
  Iter. 2: {D_1, D_3,…, D_{10}} used for training, D_2 used for testing
  …
  Iter. 10: {D_1, …, D_9} used for training, D_{10} used for testing

  Accuracy = (# tuples correctly classified) / (total # tuples)

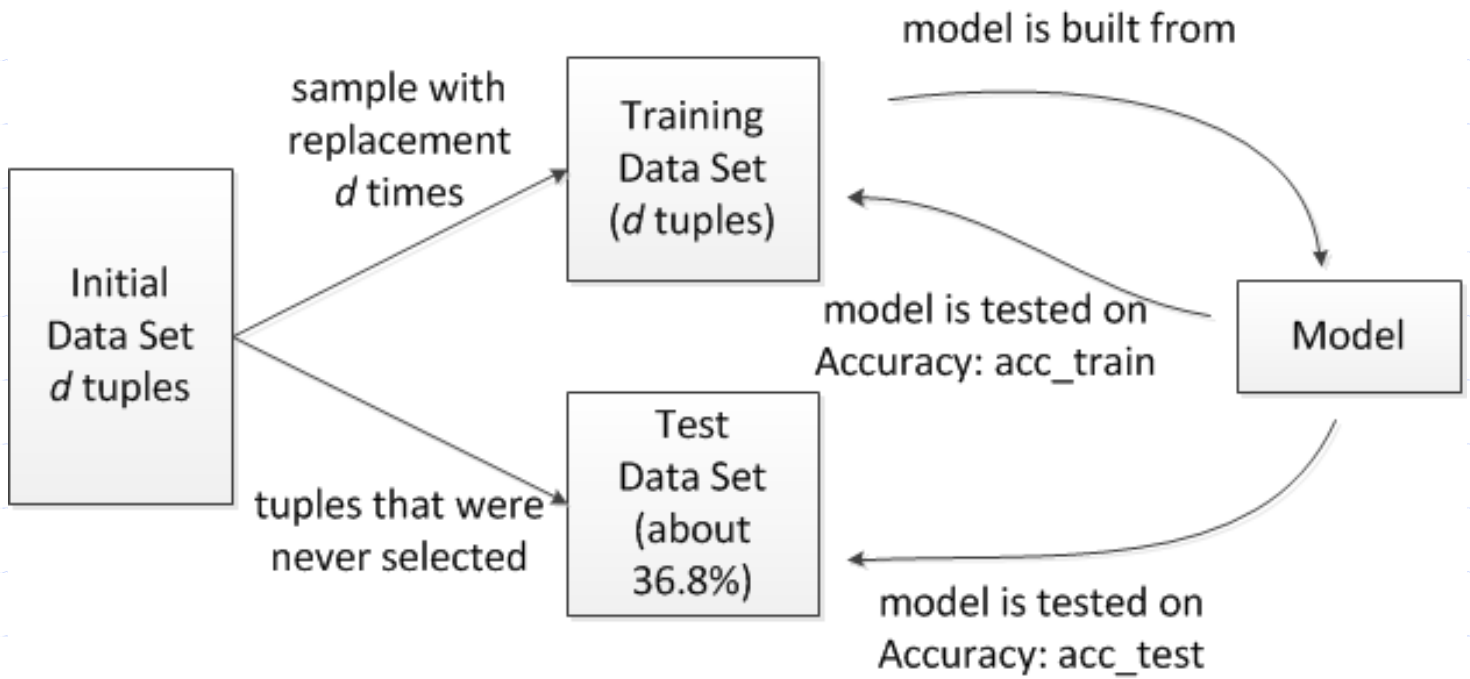
- Stratified cross-validation: folds are stratified so that class distribution in each fold is approx. the same as that in the initial data.

Evaluating Classifier Accuracy: Bootstrap

- Works well with small data sets
- Several bootstrap methods. A common one is **.632 bootstrap**
  - A data set with *d* tuples is sampled *d* times, with replacement, resulting in a training set of *d* samples.
    - each time a tuple is sampled it is re-added to the training set
  - Since samples are replaced, the same tuple can be sampled multiple times.
  - So, the training sample can have duplicates.
  - The data tuples that did not make it into the training set end up forming the test set.

Evaluating Classifier Accuracy: Bootstrap

- Bootstrap illustration:



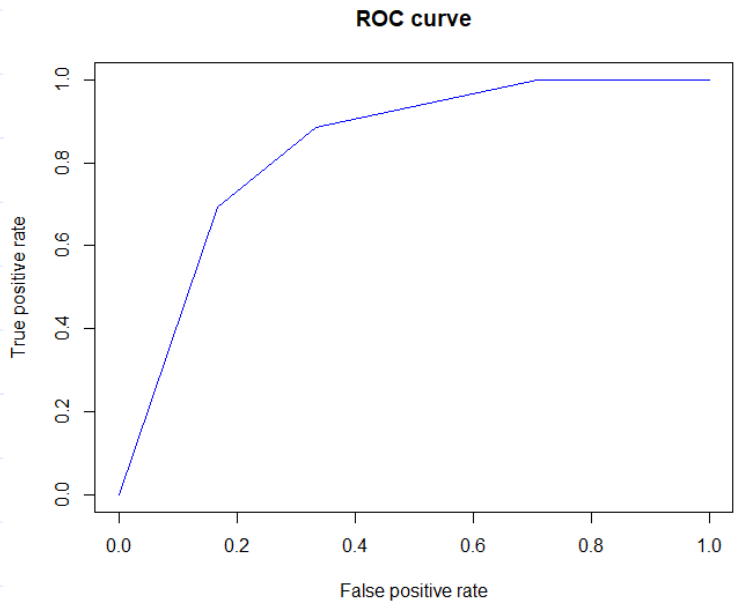
accuracy = 0.632 * acc_test + 0.368 * acc_tain

Evaluating Classifier Accuracy: Bootstrap

- **Bootstrap**
  - A probabilistic analysis shows that, on average, about 63.2% of the original data end up in the training set (which is also referred to as **bootstrap**), and the remaining 36.8% form the test set
  - Model is built from the training dataset and it is tested on both training dataset and test dataset
  - Accuracy of the model is computed by combining two accuracies

ROC Curves

- **ROC** (Receiver Operating Characteristics) curves: for visual comparison of classification models.
- Shows the trade-off between the true positive rate (TPR = TP / (TP + FN)=TP/P) and the false positive rate (FPR = FP / (FP + TN)=FP/N).
- An ROC curve can be constructed for classification methods which generate probabilities for class labels.



Worked Example

- Rank the test tuples in decreasing order of predicted probability: the one that is most likely to belong to the positive class appears at the top of the list.

tuple_id	Actual class	Probability
1	P	0.992
2	P	0.964
3	N	0.953
4	P	0.931
5	P	0.893
6	N	0.875
7	P	0.82
8	N	0.793
9	N	0.778
10	P	0.742

For each row:

- Assume all tuples above, including itself, are classified as positive and all tuples below are classified as negative.
- Calculate TPR (= TP/P) and FPR = (FP/N) and plot

Worked Example

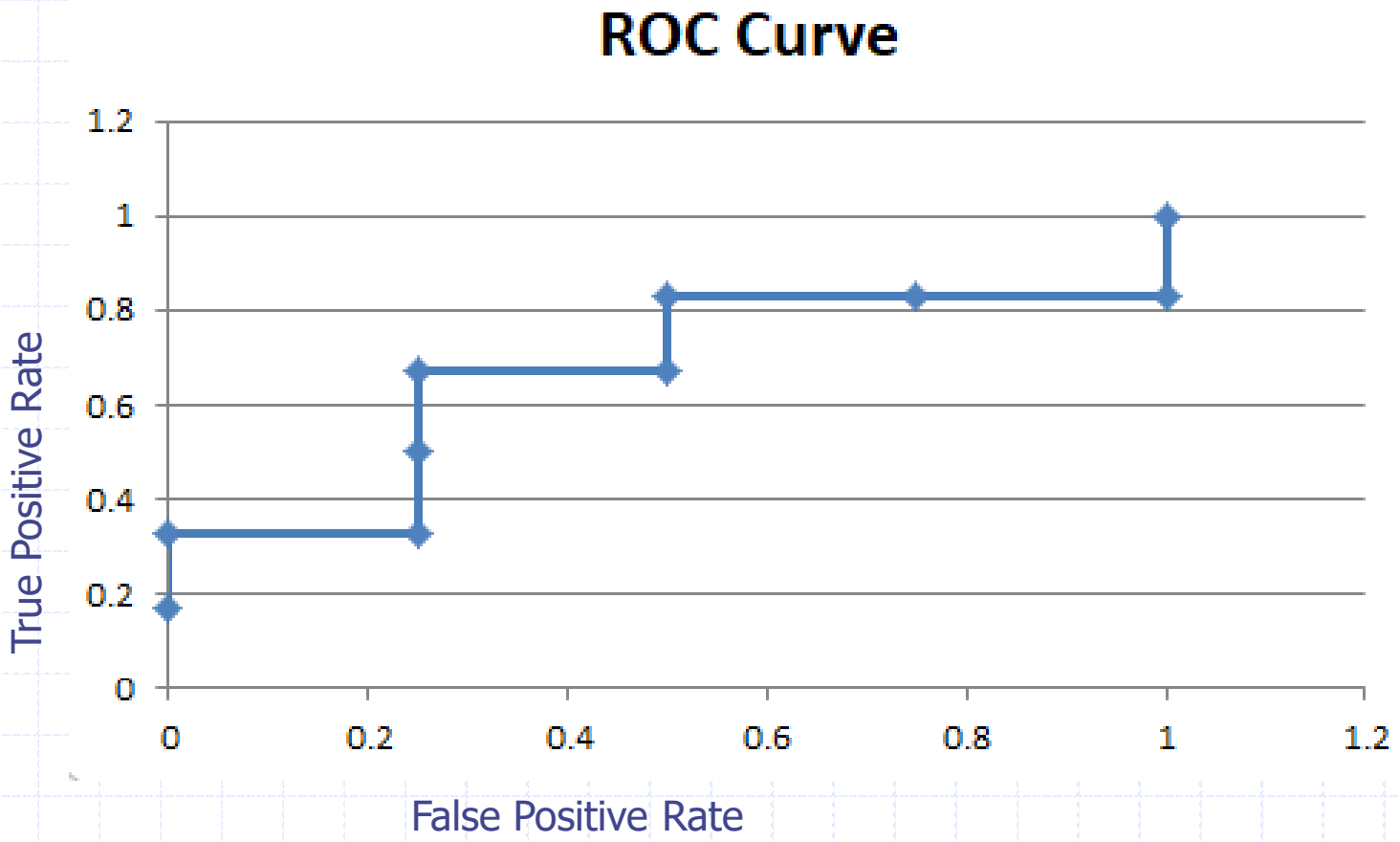
tuple_id	Actual class	Probability	Classified as	TP	FP	TN	FN	TPR	FPR
1	P	0.992	P	1	0	4	5	0.17	0
2	P	0.964	P	2	0	4	4	0.33	0
3	N	0.953	P	2	1	3	4	0.33	0.25
4	P	0.931	P	3	1	3	3	0.5	0.25
5	P	0.893	N	4	1	3	2	0.67	0.25
6	N	0.875	N	4	2	2	2	0.67	0.5
7	P	0.82	N	5	2	2	1	0.83	0.5
8	N	0.793	N	5	3	1	1	0.83	0.75
9	N	0.778	N	5	4	0	1	0.83	1.0
10	P	0.742	N	6	4	0	0	1.0	1.0

P = 6, N = 4

Fourth row:

- Top four are classified as P and all others are classified as N.
- TP is 3, FP is 1, TN is 3, FN is 3
- TPR = 3 / 6 = 0.5 and FPR = 1 / 4 = 0.25

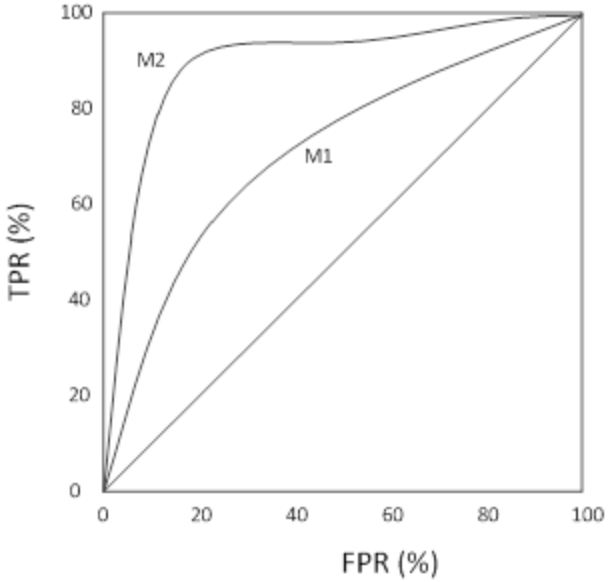
Worked Example



ROC Curves

Observations:

- Curve M2 illustrates a model in which even when the false positive rate is small (say, 20%) the true positive rate is high (90%). This model is better at detecting outcomes of interest than M1, which requires a high false positive rate in order to achieve a high true positive rate.
- The diagonal line indicates that the rate of true positives is always the same as the rate of false positives; a model of this kind is unable to distinguish between positive and negative and is for that reason useless.
- Better models can be detected by computing the area under the ROC curve (AUC)



Create ROC Curves in R

See the demo code.

```
library(rpart)
tree.model <- rpart(event1 ~ age+chol+sex, data = coronarydata)
tree.prob <- predict(tree.model, coronarydata, type='prob')

pred <- prediction(predictions = tree.prob[,2], labels = coronarydata$event1)

perf <- performance(pred, measure = "tpr", x.measure = "fpr")
plot(perf, main="ROC curve", col = "blue")

perf.auc <- performance(pred, measure = "auc")

#get area under the curve
unlist(perf.auc@y.values)
```