Confusion Matrix and ROC-II

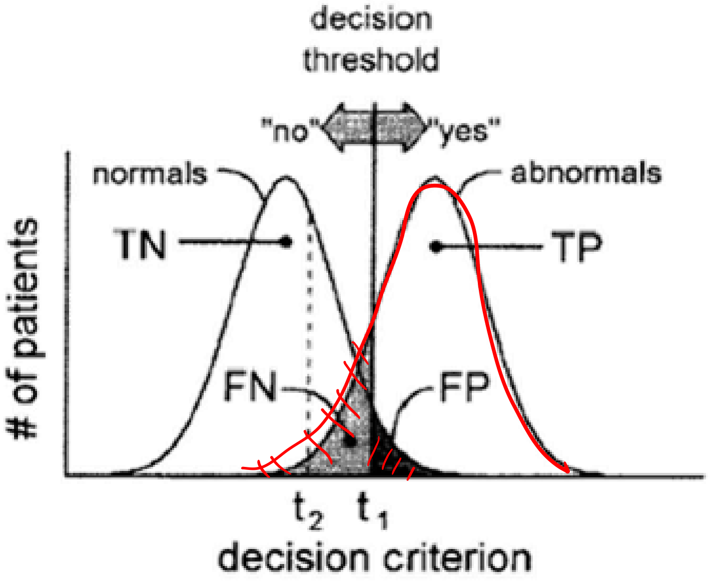
Dr A. RAMESH
DEPARTMENT OF MANAGEMENT STUDIES

Agenda

- Receiver operating characteristics curve
- Optimum threshold value

ROC analysis

- True Positive Fraction
  - TPF = TP / (TP+FN)
  - also called *sensitivity*
  - true abnormals called abnormal by the observer
- False Positive Fraction
  - FPF = FP / (FP+TN)
- *Specificity* = TN / (TN+FP)
  - True normals called normal by the observer
  - FPF = 1 - specificity

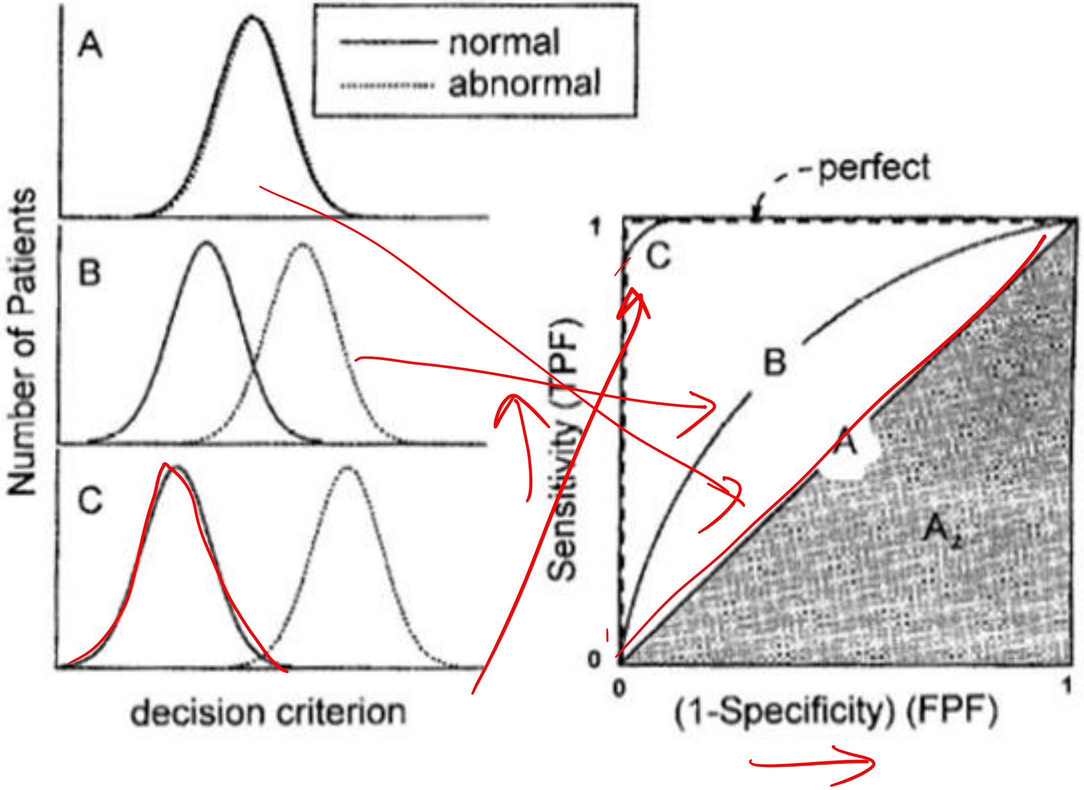


Evaluating classifiers (via their ROC curves)

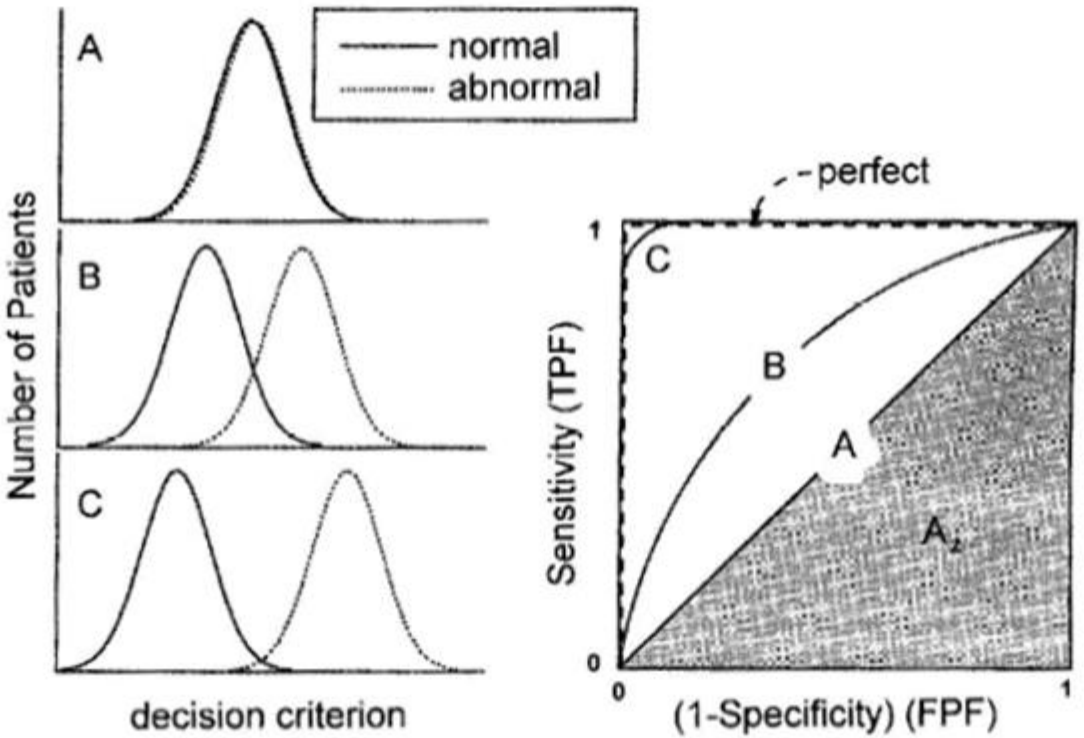
Classifier A can’t distinguish between normal and abnormal.

B is better but makes some mistakes.

C makes very few mistakes.



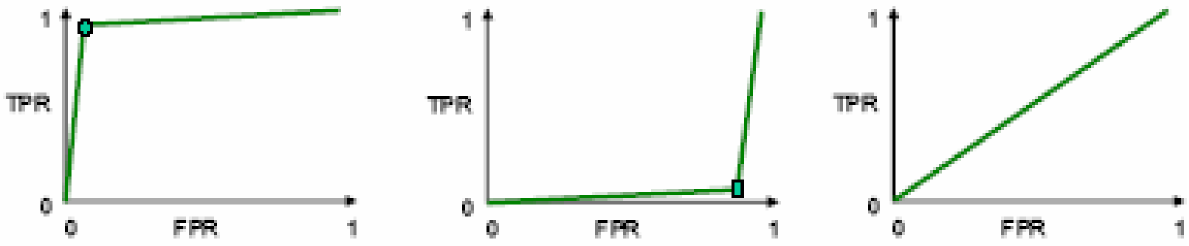
“Perfect” means no false positives and no false negatives.



ROC analysis

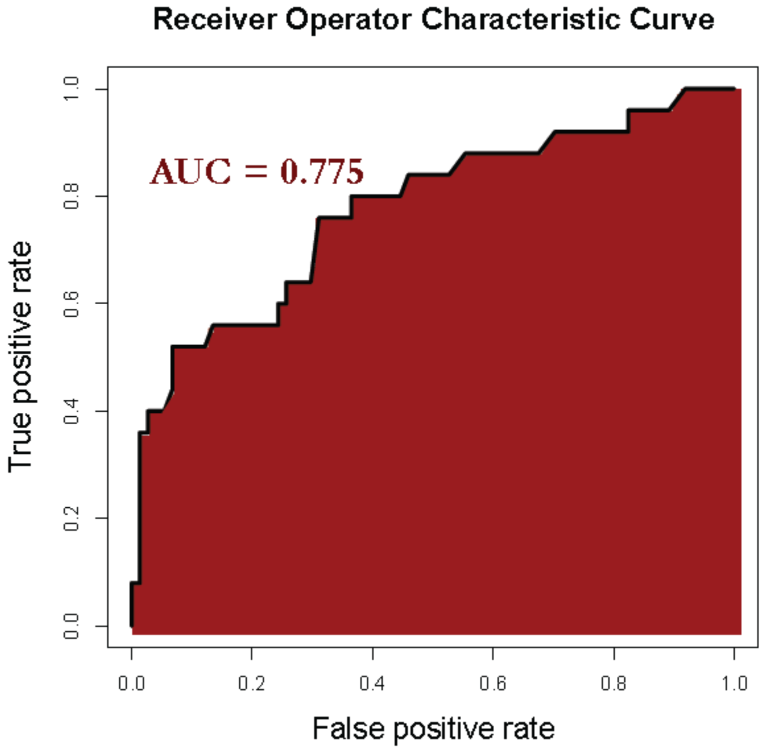
- ROC = receiver operator/operating characteristic/curve

- ROC space: good and bad classifiers.



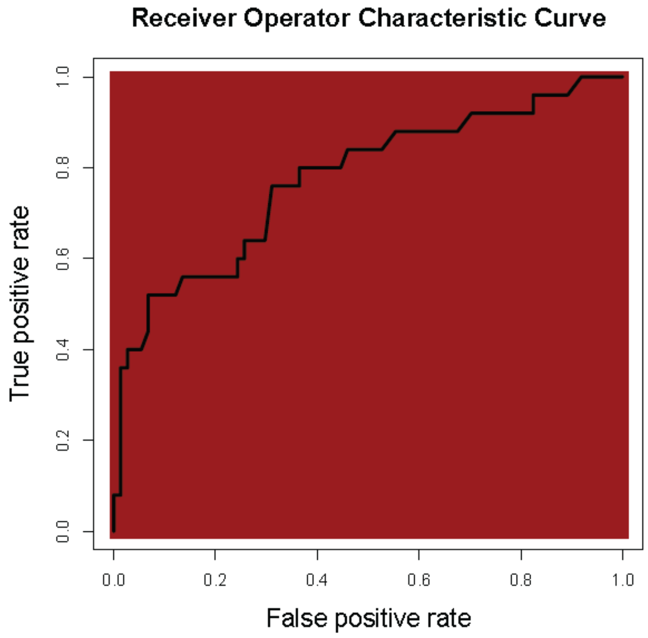
- Good classifier.
  - High TPR.
  - Low FPR.
- Bad classifier.
  - Low TPR.
  - High FPR.
- Bad classifier (real picture).

Area Under the ROC Curve (AUC)



Area Under the ROC Curve (AUC)

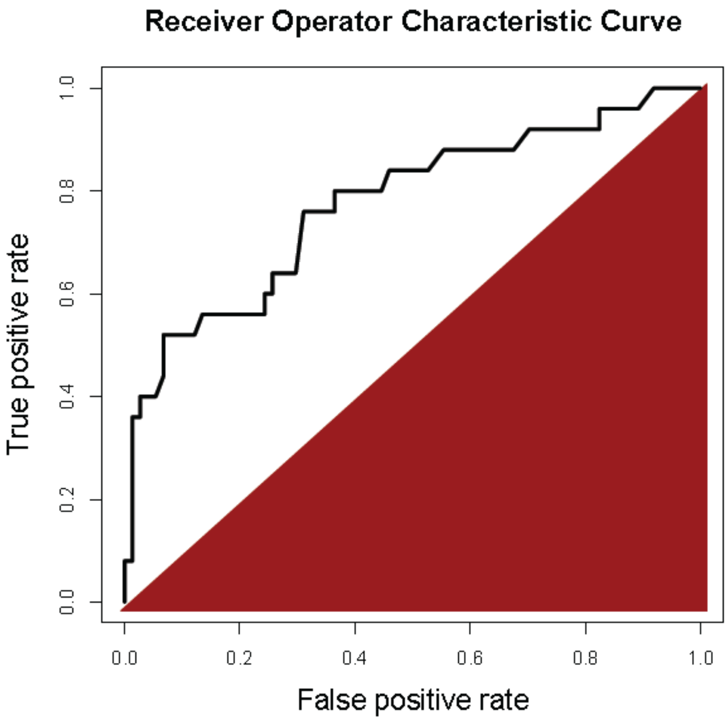
- What is a good AUC?
  - Maximum of 1 (perfect prediction)



Area Under the ROC Curve (AUC)

- What is a good AUC?
- Maximum of 1 (perfect prediction)
- Minimum of 0.5

  (just guessing)

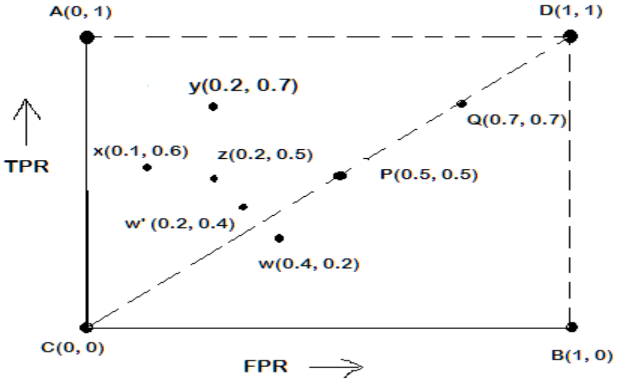


Selecting a Threshold using ROC

- Choose best threshold for best trade off
  - cost of failing to detect positives
  - costs of raising false alarms

ROC Plot

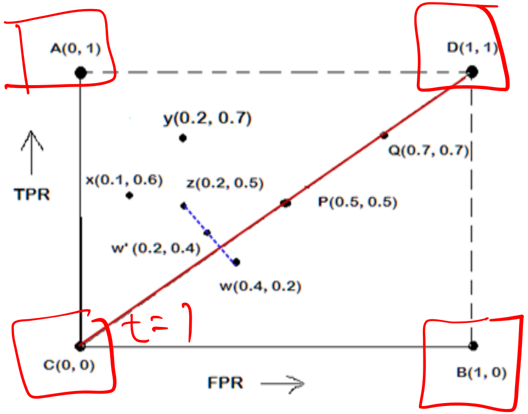
- A typical look of ROC plot with few points in it is shown in the following figure.



- Note the four cornered points are the four extreme cases of classifiers

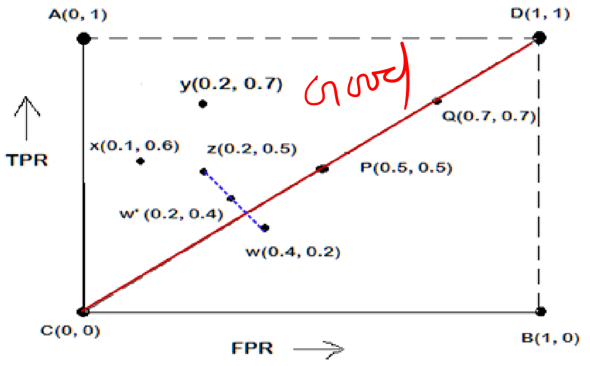
Interpretation of Different Points in ROC Plot

- The four points (A, B, C, and D)
- A: TPR = 1, FPR = 0, the ideal model, i.e., the perfect classifier, no false results
- B: TPR = 0, FPR = 1, the worst classifier, not able to predict a single instance
- C: TPR = 0, FPR = 0, the model predicts every instance to be a Negative class, i.e., it is an ultra-conservative classifier
- D: TPR = 1, FPR = 1, the model predicts every instance to be a Positive class, i.e., it is an ultra-liberal classifier



Interpretation of Different Points in ROC Plot

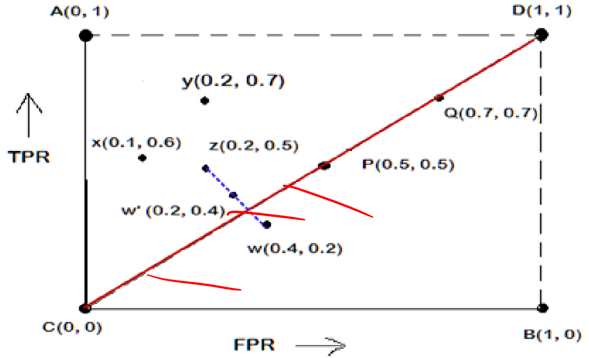
- Let us interpret the different points in the ROC plot.
- **The points on the upper diagonal region**
- All points, which reside on upper-diagonal region are corresponding to classifiers "good" as their TPR is as good as FPR (i.e., FPRs are lower than TPRs)
- Here, X is better than Z as X has higher TPR and lower FPR than Z.
- If we compare X and Y, neither classifier is superior to the other



Interpretation of Different Points in ROC Plot

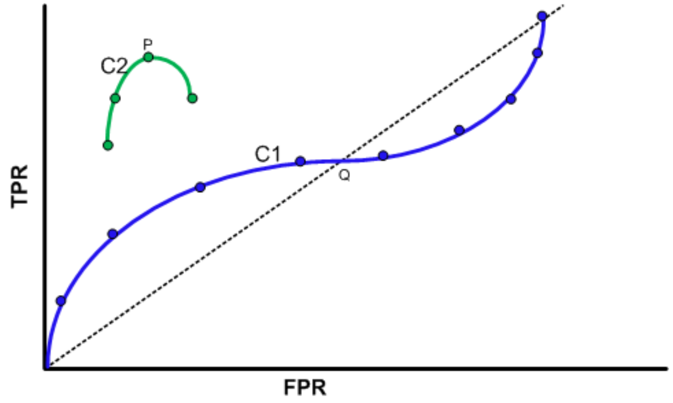
- Let us interpret the different points in the ROC plot.
- The points on the lower diagonal region
  - The Lower-diagonal triangle corresponds to the classifiers that are worst than random classifiers
  - A classifier that is worser than random guessing, simply by reversing its prediction, we can get good results.

W'(0.2, 0.4) is the better version than W(0.4, 0.2), W' is a mirror reflection of W



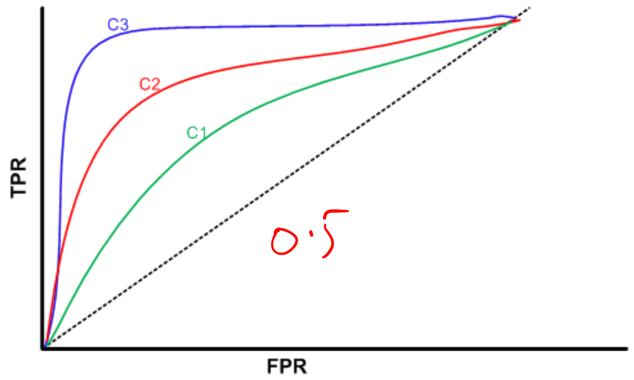
Tuning a Classifier through ROC Plot

- Using ROC plot, we can compare two or more classifiers by their TPR and FPR values and this plot also depicts the trade-off between TPR and FPR of a classifier.
- Examining ROC curves can give insights into the best way of tuning parameters of classifier.
- For example, in the curve C2, the result is degraded after the point P.
- Similarly for the observation C1, beyond Q the settings are not acceptable.

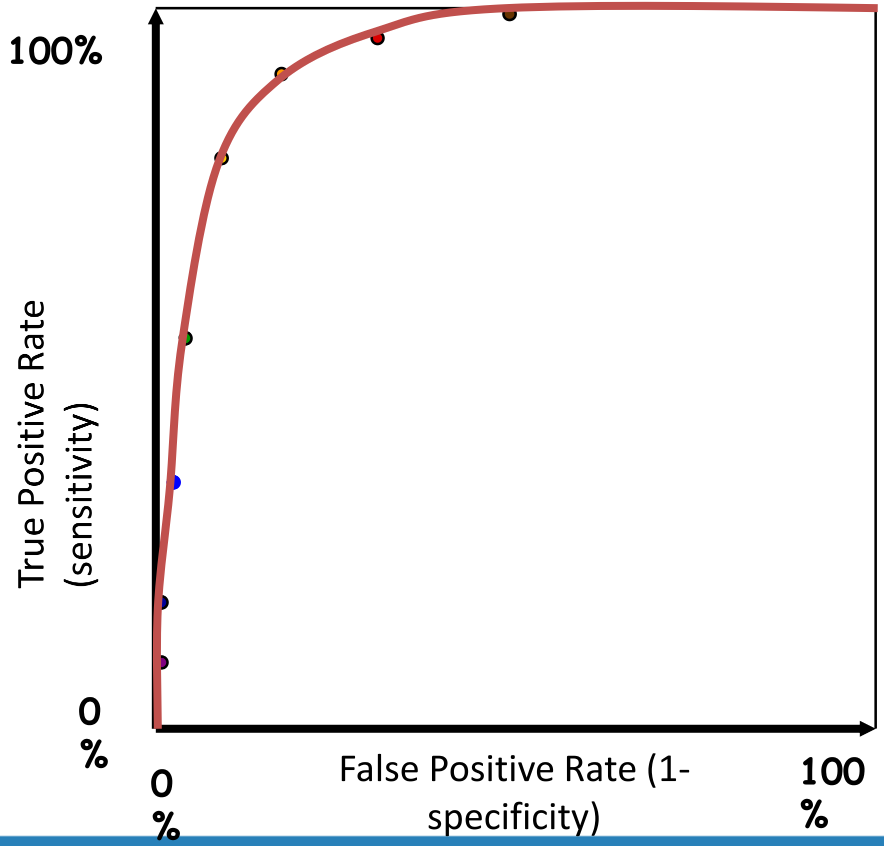


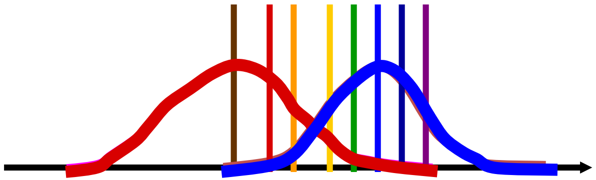
Comparing Classifiers trough ROC Plot

- We can use the concept of "area under curve" (AUC) as a better method to compare two or more classifiers.
- If a model is perfect, then its AUC = 1.
- If a model simply performs random guessing, then its AUC = 0.5
- A model that is strictly better than other, would have a larger value of AUC than the other.
- Here, C3 is best, and C2 is better than C1 as AUC(C3)>AUC(C2)>AUC(C1).



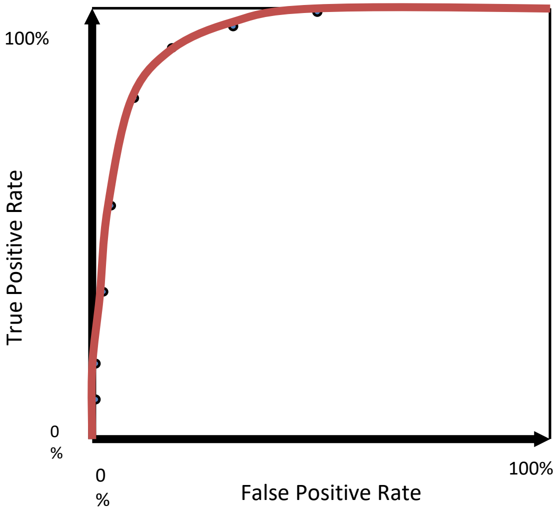
ROC curve


100%
True Positive Rate
(sensitivity)
0
%
0
%
False Positive Rate (1-
specificity)
100
%

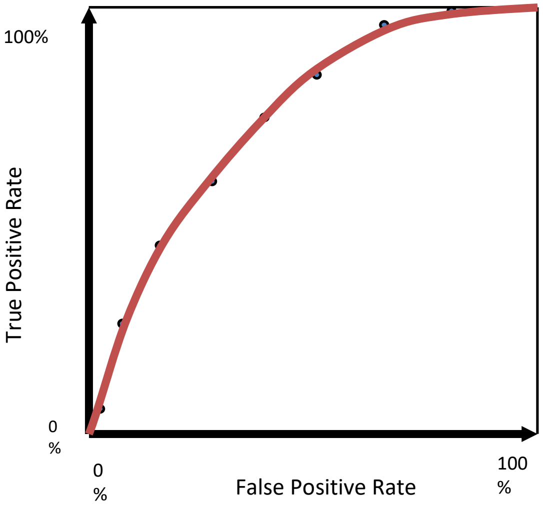


ROC curve comparison

A good test:

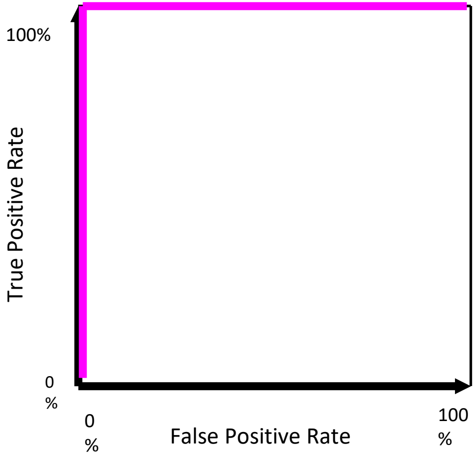


A poor test:

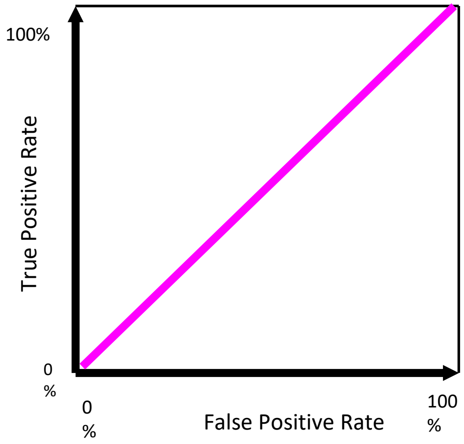


ROC curve extremes

Best Test:



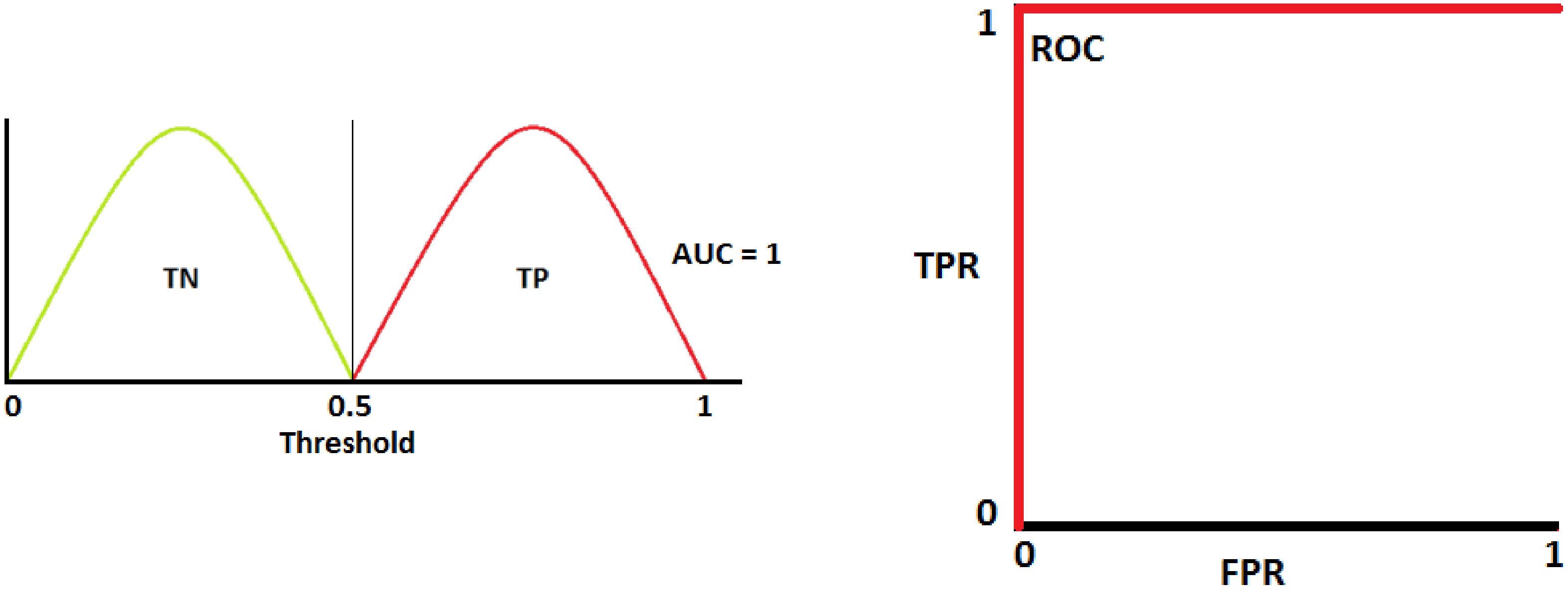
Worst test:



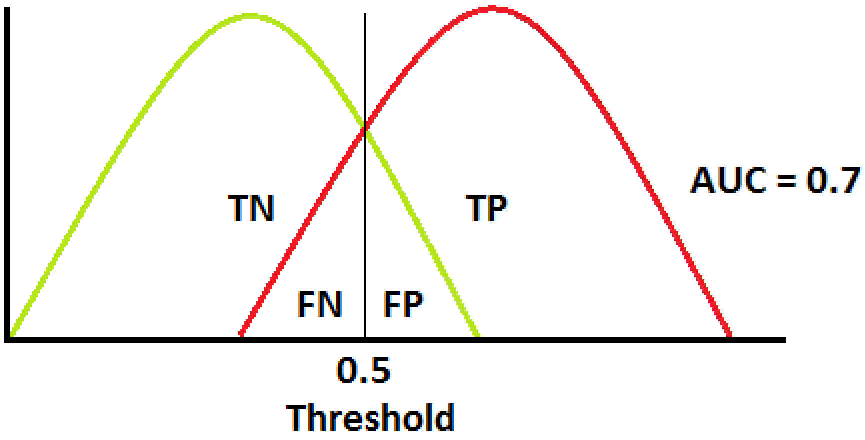
The distributions don’t overlap at all

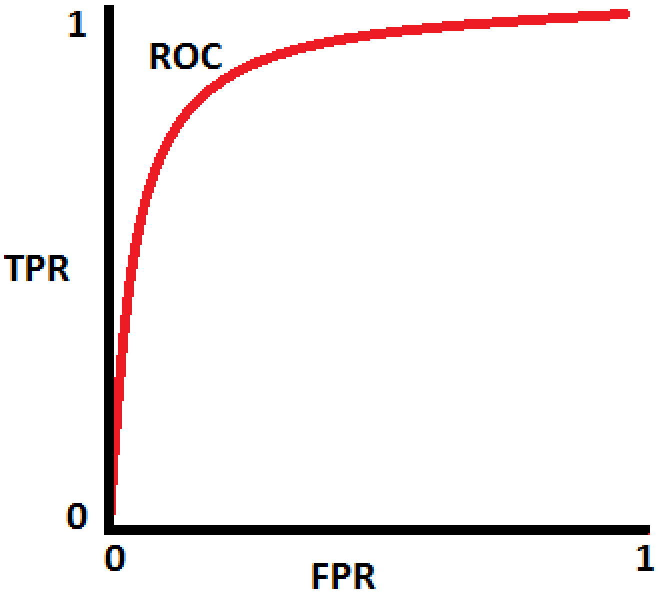
The distributions overlap completely

ROC curve extremes

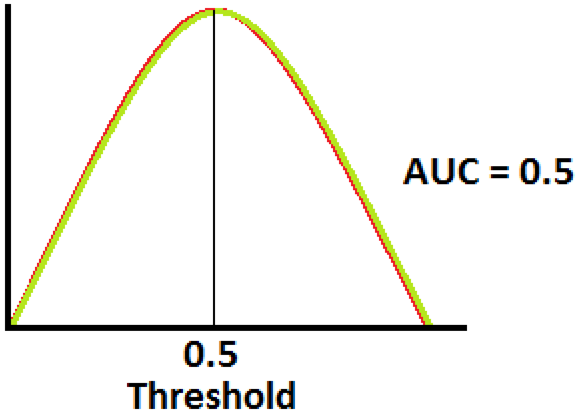


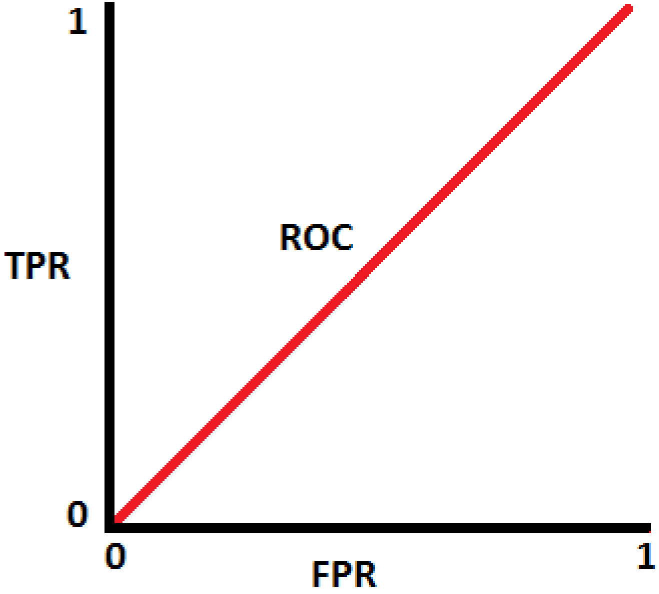
Typical ROC





ROC curve extremes





Example

- Let us consider an application of logistic regression involving a direct mail promotion being used by Simmons Stores.
- Simmons owns and operates a national chain of women's apparel stores.
- Five thousand copies of an expensive four-color sales catalog have been printed, and each catalog includes a coupon that provides a $50 discount on purchases of $200 or more.
- The catalogs are expensive and Simmons would like to send them to only those customers who have the highest probability of using the coupon.

Sources: Statistics for Business and Economics,11th Edition by David R. Anderson (Author), Dennis J. Sweeney (Author), Thomas A. Williams (Author)

Variables

- Management thinks that annual spending at Simmons Stores and whether a customer has a Simmons credit card are two variables that might be helpful in predicting whether a customer who receives the catalog will use the coupon.
- Simmons conducted a pilot study using a random sample of 50 Simmons credit card customers and 50 other customers who do not have a Simmons credit card.
- Simmons sent the catalog to each of the 100 customers selected.
- At the end of a test period, Simmons noted whether the customer used the coupon or not?

Data (10 customer out of 100)

Customer	Spending	Card	Coupon
1	2.291	1	0
2	3.215	1	0
3	2.135	1	0
4	3.924	0	0
5	2.528	1	0
6	2.473	0	1
7	2.384	0	0
8	7.076	0	0
9	1.182	1	1
10	3.345	0	0

Explanation of Variables

- The amount each customer spent last year at Simmons is shown in thousands of dollars and the credit card information has been coded as 1 if the customer has a Simmons credit card and 0 if not.
- In the Coupon column, a 1 is recorded if the sampled customer used the coupon and 0 if not.

Loading data file and get some statistical detail

In [1]:
```
import pandas as pd
import matplotlib.pyplot as plt
```

In [2]:
```
data = pd.read_excel('Simmons.xls')
data.head()
```

Out[2]:

	Customer	Spending	Card	Coupon
0	1	2.291	1	0
1	2	3.215	1	0
2	3	2.135	1	0
3	4	3.924	0	0
4	5	2.528	1	0

In [3]:
```
data.describe() #it is used to get some statistical detail
```

Out[3]:

	Customer	Spending	Card	Coupon
count	100.000000	100.000000	100.000000	100.000000
mean	50.500000	3.333790	0.500000	0.400000
std	29.011492	1.741298	0.502519	0.492366
min	1.000000	1.058000	0.000000	0.000000
25%	25.750000	2.059000	0.000000	0.000000
50%	50.500000	2.805500	0.500000	0.000000
75%	75.250000	4.468250	1.000000	1.000000
max	100.000000	7.076000	1.000000	1.000000

Method's description

- Dataframe.describe(): This method is used to get basic statistical details such as central tendency, dispersion and shape of dataset's distribution.
- Numpy.unique(): This method gives unique values in particular column.
- Series.value_counts(): Returns object containing counts of unique values.
- ravel(): It will return one dimensional array with all the input array elements.

Split dataset into training and testing sets

```
In [4]:  data['Coupon'].unique() # It gives unique value in perticular column

Out[4]: array([0, 1], dtype=int64)

In [5]:  data['Coupon'].value_counts()

Out[5]: 0     60
        1     40
        Name: Coupon, dtype: int64

In [7]:  from sklearn import linear_model
         from sklearn.model_selection import train_test_split
         from sklearn.linear_model import LogisticRegression

In [8]:  x = data[['Card','Spending']]
         y = data['Coupon'].values.reshape(-1,1)
         x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.25, random_state = 42)

In [9]:  len(x_train), len(y_train), len(x_test), len(y_test)

Out[9]: (75, 75, 25, 25)
```

Building the model and predicting values

In [10]:
```
Lreg = LogisticRegression(solver='lbfgs')
Lreg.fit(x_train, y_train.ravel()) #ravel() will return 1D array with all the input-array elements
```

Out[10]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
          intercept_scaling=1, max_iter=100, multi_class='warn',
          n_jobs=None, penalty='l2', random_state=None, solver='lbfgs',
          tol=0.0001, verbose=0, warm_start=False)
```

In [11]:
```
y_predict = Lreg.predict(x_test)
y_predict
```

Out[11]:
```
array([1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1,
       1, 0, 0], dtype=int64)
```

In [12]:
```
y_predict_train = Lreg.predict(x_train)
y_predict_train
```

Out[12]:
```
array([0, 0, 0, 0, 0, 0, 0, 1, 0, 1, 0, 0, 1, 0, 0, 1, 0, 1, 0, 0, 0, 1,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0,
       1, 1, 0, 0, 0, 0, 1, 1, 0], dtype=int64)
```

Calculate probability of predicting data values

In [13]:
```python
y_prob_train = Lreg.predict_proba(x_train)[:,1]
y_prob_train.reshape(1,-1)
```

Out[13]:
```
array([[0.49622117, 0.32880793, 0.44329114, 0.33320924, 0.41456465,
        0.32890329, 0.3975043 , 0.66921229, 0.25844531, 0.63672372,
        0.29274386, 0.28466974, 0.5159296 , 0.41992276, 0.24342356,
        0.528514  , 0.47965107, 0.52805789, 0.33191449, 0.27457435,
        0.49179296, 0.63261616, 0.24690181, 0.47089452, 0.27842076,
        0.41663875, 0.36155602, 0.49970327, 0.23621636, 0.37860052,
        0.48809323, 0.28877877, 0.28563859, 0.37231882, 0.65309742,
        0.43807264, 0.33638478, 0.40406607, 0.23431177, 0.37282384,
        0.49970327, 0.39768396, 0.32880793, 0.25782472, 0.47393834,
        0.42878861, 0.26520939, 0.33320924, 0.54682499, 0.45446086,
        0.44326597, 0.4965167 , 0.60065954, 0.38989654, 0.49149447,
        0.27414424, 0.27785686, 0.67464141, 0.28195004, 0.48593427,
        0.38633222, 0.31373499, 0.42810085, 0.27418723, 0.44371771,
        0.41629601, 0.642004  , 0.6571001 , 0.44068025, 0.28195004,
        0.40217015, 0.43807264, 0.50977653, 0.57944626, 0.2904233 ]])
```

75 /

In [14]:
```python
y_prob = Lreg.predict_proba(x_test)[:,1]
y_prob.reshape(1,-1)
y_prob
```

Out[14]:
```
array([0.52802946, 0.49516653, 0.45703306, 0.27712052, 0.34390047,
       0.26825171, 0.27712052, 0.607686  , 0.42836534, 0.43637155,
       0.31387455, 0.23676248, 0.45703306, 0.43602768, 0.37596116,
       0.44900317, 0.46952365, 0.68521935, 0.25167254, 0.47073304,
       0.42361093, 0.56580644, 0.52792177, 0.40302605, 0.27457435])
```

25

Summary for logistic model

In [15]:

```
x = data[['Spending', 'Card']]
y = data['Coupon']

import statsmodels.api as sm
x1 = sm.add_constant(x)
logit_model=sm.Logit(y,x1)
result=logit_model.fit()
print(result.summary())
```

```
Optimization terminated successfully.
         Current function value: 0.604869
         Iterations 5
                           Logit Regression Results
==============================================================================
Dep. Variable:                 Coupon   No. Observations:                  100
Model:                          Logit   Df Residuals:                       97
Method:                           MLE   Df Model:                            2
Date:                Mon, 16 Sep 2019   Pseudo R-squ.:                  0.1012
Time:                        10:15:13   Log-Likelihood:                -60.487
converged:                       True   LL-Null:                       -67.301
                                        LLR p-value:                  0.001098
==============================================================================
                 coef    std err          z      P>|z|      [0.025      0.975]
------------------------------------------------------------------------------
const         -2.1464      0.577     -3.718      0.000      -3.278      -1.015
Spending       0.3416      0.129      2.655      0.008       0.089       0.594
Card           1.0987      0.445      2.471      0.013       0.227       1.970
==============================================================================
```

Accuracy Checking

- By using accuracy_score function.
- By using confusion matrix

	Predicted (0)	Predicted (1)
Actual (0)	True Negative(tn)	False Positive(fp)
Actual (1)	False Negative(fn)	True Positive(tp)

Calculating Accuracy Score using Confusion Matrix

In [16]:
```
from sklearn.metrics import accuracy_score
score =accuracy_score(y_test,y_predict)
score
```

Out[16]:
```
0.76
```

In [17]:
```
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_predict)
```

Out[17]:
```
array([[15,  1],
       [ 5,  4]], dtype=int64)
```

In [18]:
```
tn, fp, fn, tp = confusion_matrix(y_test, y_predict).ravel()
print("True Negatives: ",tn)
print("False Positives: ",fp)
print("False Negatives: ",fn)
print("True Positives: ",tp)
```

```
True Negatives:  15
False Positives:  1
False Negatives:  5
True Positives:  4
```

Generating Classification Report

In [19]:
```
from sklearn.metrics import classification_report
print(classification_report(y_test, y_predict))
```

	precision	recall	f1-score	support
0	0.75	0.94	0.83	16
1	0.80	0.44	0.57	9
micro avg	0.76	0.76	0.76	25
macro avg	0.78	0.69	0.70	25
weighted avg	0.77	0.76	0.74	25

- Recall gives us an idea about when it's actually yes, how often does it predict yes.
- Precision tells us about when it predicts yes, how often is it correct

Interpreting Classification Report

- Precision = tp / (tp + fp)
- Accuracy = (tp + tn) / (tp + tn + fp + fn)
- Recall= tp / (tp + fn)

	Predicted (0)	Predicted (1)
Actual (0)	tn	fp
Actual (1)	fn	tp

Thank You