# EE2211 Introduction to Machine Learning

T14 & T22, Chua Dingjuan elechuad@nus.edu.sg
Materials @ tiny.cc/ee2211tut

# Tutorial 10

TVT, Confusion Matrices, ROC/AUC

# Training, Validation and Test

| | | |
|---|---|---|
| Training set | Validation set | Test set |
| Training set | Test set | Validation set |
| Validation set | Test set | Training set |
| Test set | Validation set | Training set |

Four-fold cross-validation test

Other common partitioning:

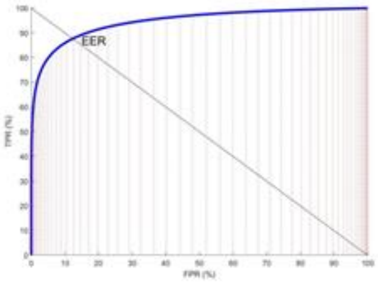
| Training set | Validation set | Test set | |
|---|---|---|---|
| 80% | 10% | 10% | 10-fold CV |
| 60% | 20% | 20% | 5-fold CV |
| 40% | 10% | 50% | 2-fold CV |

# Confusion Matrix

### Confusion Matrix for Binary Classification

| | $\hat{\text{P}}$ (predicted) | $\hat{\text{N}}$ (predicted) | |
|---|---|---|---|
| P (actual) | TP | FN | Recall TP/(TP+FN) |
| N (actual) | FP | TN | |
| | Precision TP/(TP+FP) | Accuracy (TP+TN)/(TP+TN+FP+FN) | |

# ROC / AUC




Receiver Operating Characteristic (ROC) Curve

**Area Under the Curve (ROC curve)**

Consider also a Heaviside step function given by

$$u(e) = \begin{cases} 1, & \text{if } e > 0 \\ 0.5 & \text{if } e = 0 \\ 0 & \text{if } e < 0 \end{cases}$$

The Area Under the ROC Curve (AUC) can be Expressed as

$$\text{AUC} = \frac{1}{m^+ m^-} \sum_{i=1}^{m^+} \sum_{j=1}^{m-} u(e_{ij})$$

The AUC=1 when all the samples are ranked correctly.

# Discussion of Solutions

10

Q5,6,7, 1-4(poll)

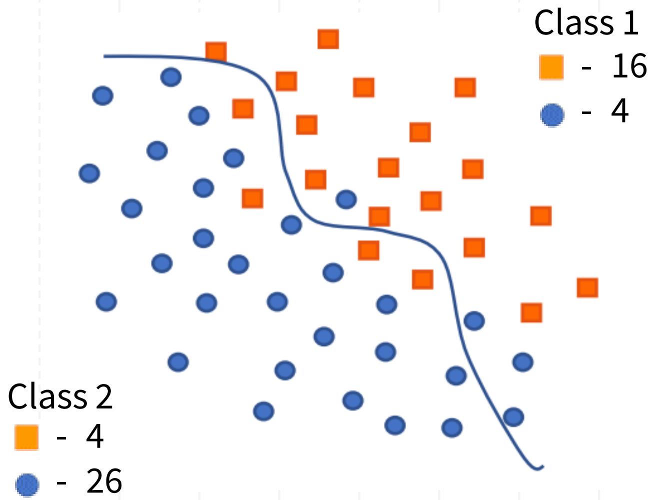
# Question5

Tabulate the confusion matrices for the following classification problems.

(a) Binary problem (the class-1 and class-2 data points are respectively indicated by squares and circles)

■ - Class 1

● - Class 2




(a) SOLUTION

■ - Class 1 - POSITIVE

● - Class 2 - NEGATIVE

| | $\widehat{P_1}$ | $\widehat{P_2}$ |
|---|---|---|
| $P_1$ | 16 TP | 4 FN |
| $P_2$ | 4 FP | 26 TN |

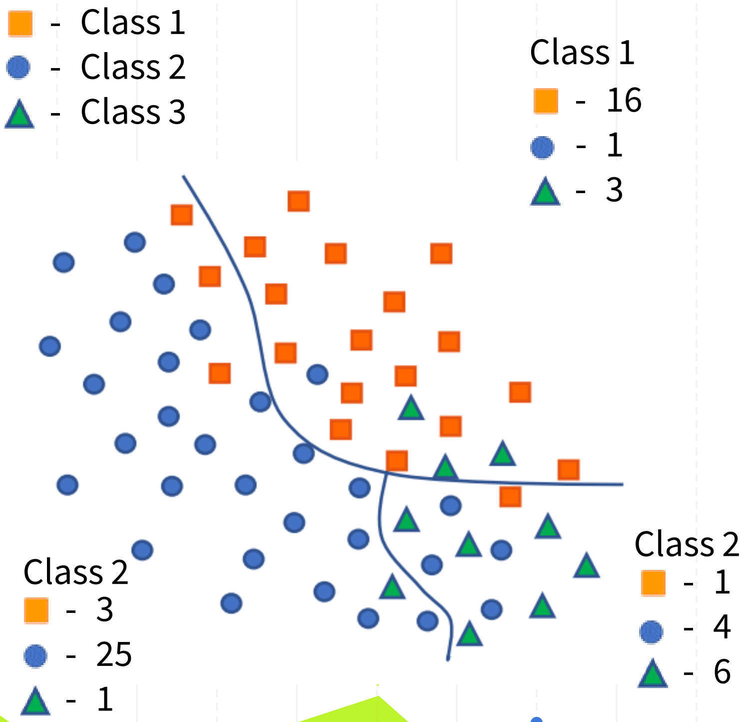
**Confusion Matrix for Binary Classification**

| | $\hat{P}$ (predicted) | $\hat{N}$ (predicted) | |
|---|---|---|---|
| **P (actual)** | TP | FN | Recall TP/(TP+FN) |
| **N (actual)** | FP | TN | |
| | Precision TP/(TP+FP) | | Accuracy (TP+TN)/(TP+TN+FP+FN) |

# Question5

Tabulate the confusion matrices for the following classification problems.

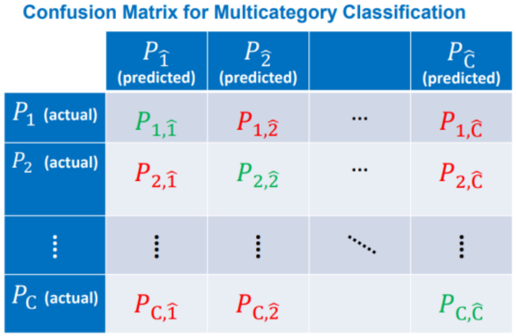
(b) Three-category problem (the class-1, class-2 and class-3 data points are respectively indicated by squares, circles and triangles).




(b) SOLUTION

| | $\hat{P}_1$ | $\hat{P}_2$ | $\hat{P}_3$ |
|---|---|---|---|
| $P_1$ | 16 | 3 | 1 |
| $P_2$ | 1 | 25 | 4 |
| $P_3$ | 3 | 1 | 6 |

- Class 1
- Class 2
- Class 3


Confusion Matrix for Multicategory Classification

| | $P_{\hat{1}}$ (predicted) | $P_{\hat{2}}$ (predicted) | | $P_{\hat{C}}$ (predicted) |
|---|---|---|---|---|
| $P_1$ (actual) | $P_{1,\hat{1}}$ | $P_{1,\hat{2}}$ | ⋯ | $P_{1,\hat{C}}$ |
| $P_2$ (actual) | $P_{2,\hat{1}}$ | $P_{2,\hat{2}}$ | ⋯ | $P_{2,\hat{C}}$ |
| ⋮ | ⋮ | ⋮ | ⋱ | ⋮ |
| $P_C$ (actual) | $P_{C,\hat{1}}$ | $P_{C,\hat{2}}$ | | $P_{C,\hat{C}}$ |

# Question 6

Get the data set “from sklearn.datasets import load_iris”. Perform a 5-fold Cross-validation to observe the best polynomial order (among orders 1 to 10 and without regularization) for validation prediction. Note that, you will have to partition the whole dataset for training/validation/test parts, where the size of validation set is the same as that of test. Provide a plot of the average 5-fold training and validation error rates over the polynomial orders. The randomly partitioned data sets of the 5-fold shall be maintained for reuse in evaluation of future algorithms.

Recall Tutorial 7!

## Question2

This question further explores linear regression and ridge regression.

(a) Use the polynomial model from orders 1 to 6 to train and test the data without regularization.

Plot the Mean Squared Errors (MSE) over orders from 1 to 6 for both the training and the test sets.

Which model order provides the best MSE in the training and test sets? Why?

(b) Use Regularization λ=1

Training Data

$\{x=-10\} \rightarrow \{y=4.18\}$

$\{x=-8\} \rightarrow \{y=2.42\}$

$\{x=-3\} \rightarrow \{y=0.22\}$

$\{x=-1\} \rightarrow \{y=0.12\}$

$\{x=2\} \rightarrow \{y=0.25\}$

$\{x=7\} \rightarrow \{y=3.09\}$

6

SOLUTIONS – Check your answers.

NO REGULARIZATION

| Order | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Training MSE | 2.3071 | 8.4408e-03 | 8.3026e-03 | 1.7348e-03 | 3.8606e-25 | 2.3656e-17 |
| Test MSE | 3.0006 | 0.0296 | 0.0301 | 0.0854 | 1.0548 | 10.7674 |

REGULARIZATION

| Order | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Training MSE | 2.3586 | 8.4565e-03 | 8.3560e-03 | 1.8080e-03 | 7.2650e-04 | 1.9348e-04 |
| Test MSE | 3.2756 | 0.0302 | 0.0314 | 0.0939 | 0.4369 | 6.0202 |

## Question2 (b)

(b) Use Regularization (ridge regression) λ=1 for all orders and repeat the same analyses. Compare the plots of (a) and (b). What do you see?

(b) SOLUTION

- With regularization, we can simply use the primal solution even for order 6
- Only one small difference in code:

```
#step 3, solving for coefficients of regression polynomial
w = np.linalg.inv(P.T @ P + np.identity(len(P.T))) @ P.T @ y
```

NO REGULARIZATION

| Order | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Training MSE | 2.3071 | 8.4408e-03 | 8.3026e-03 | 1.7348e-03 | 3.8606e-25 | 2.3656e-17 |
| Test MSE | 3.0006 | 0.0296 | 0.0301 | 0.0854 | 1.0548 | 10.7674 |

REGULARIZATION

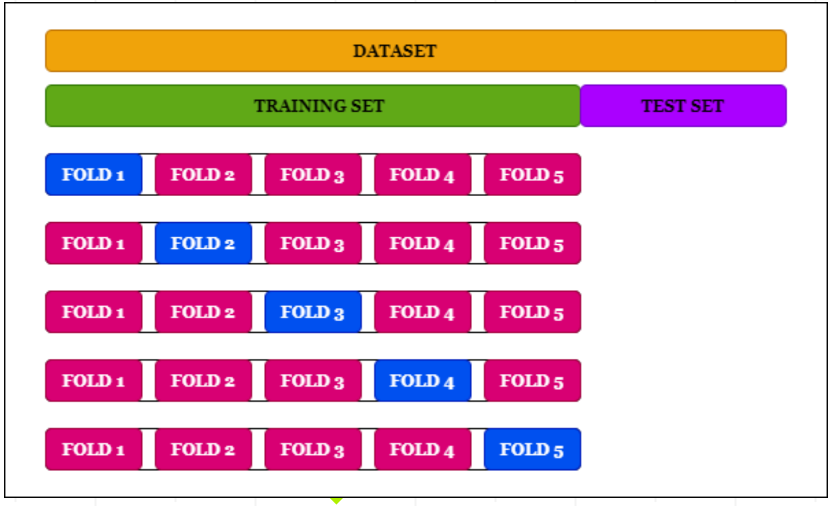
| Order | 1 | 2 | 3 | 4 | 5 | 6 |
|---|---|---|---|---|---|---|
| Training MSE | 2.3586 | 8.4565e-03 | 8.3560e-03 | 1.8080e-03 | 7.2650e-04 | 1.9348e-04 |
| Test MSE | 3.2756 | 0.0302 | 0.0314 | 0.0939 | 0.4369 | 6.0202 |

- None of the curves pass through training data exactly → training error increased slightly
- Test MSE for orders 5 & 6 reduced → Overfitting reduced.
- But Test MSE for orders 1 to 4 increased!! → Overly strong regularization.
- Regularization did not help in identifying best polynomial order → cross-validation to be learnt in future!

# Question 6

Get the data set “from sklearn.datasets import load_iris”. Perform a 5-fold Cross-validation to observe the best polynomial order (among orders 1 to 10 and without regularization) for validation prediction.

Provide a plot of the average 5-fold training and test error rates over the polynomial orders. The randomly partitioned data sets of the 5-fold shall be maintained for reuse in evaluation of future algorithms.




```
MAIN STEPS in Cross-Validation::
```

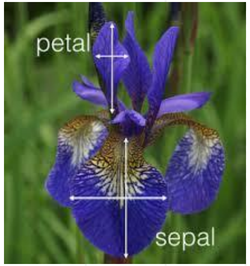
- Set aside test data

```
For each order from 1 to 10
  For each fold from 1 to 5
```

- Split data into training and validation data
- Using skpolynom, train polynomial on x_train ➔ solve for w
  - Calculate y_train_pred ➔ calculate training error
- Using x_test and the polynomial model (w)
  - Calculate y_val_pred ➔ calculate validation error
- Save errors for plotting

# Question 6

# Preparation...




```
5.1,3.5,1.4,0.2,Iris-setosa
4.9,3.0,1.4,0.2,Iris-setosa
4.7,3.2,1.3,0.2,Iris-setosa
```

```
##--- load data from scikit ---##
import numpy as np
import pandas as pd
print("pandas version: {}".format(pd.__version__))
import sklearn
print("scikit-learn version: {}".format(sklearn.__versi
from sklearn.datasets import load_iris
iris_dataset = load_iris()
X = np.array(iris_dataset['data'])
y = np.array(iris_dataset['target'])
## one-hot encoding
Y = list()
for i in y:
 letter = [0, 0, 0]
 letter[i] = 1
 Y.append(letter)
Y = np.array(Y)
```

Randomly sorts an array every time the function is called. Here it sorts from 0 to 149.

```
test_Idx = np.random.RandomState(seed=2).permutation(Y.shape[0])
X_test = X[test_Idx[:25]]
Y_test = Y[test_Idx[:25]]
X = X[test_Idx[25:]]
Y = Y[test_Idx[25:]]
```

```
>>> y
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

```
>>> Y
array([[1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0],
       [1, 0, 0]
```

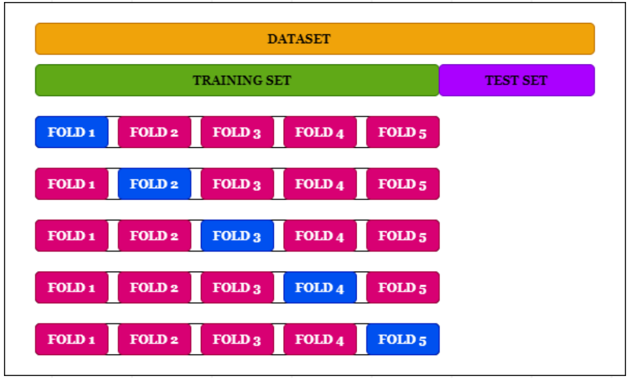
```
>>> test_Idx
array([  6,   3, 113,  12,  24, 129,  25, 108, 128,  45,  48,  42,  35,
         5,  85,  54,  41,  96, 144,  89,  77,  74, 115,  94,  87,  29,
         2, 127,  44, 125, 126,  23,  64, 117,  84,  14, 132,  91,  53,
       141,  78,  97, 143,  93,  11, 134,  28,  65,   0,  27,  36, 106,
       148, 131,  20, 140, 136, 105, 119,  13,  30, 101,  99,  59, 135,
       123,  71,  21,  55,  16, 114,  92,  98,  18,  81,  61,  86, 122,
```

# pandas.DataFrame.astype

`DataFrame.astype(dtype, copy=True, errors='raise')`

Cast a pandas object to a specified dtype `dtype`.

Given data may not be evenly distributed for each class thus a random partitioning (based on the randomly permuted index) is used.




```
from sklearn.preprocessing import PolynomialFeatures
error_rate_train_array = []
error_rate_val_array = []
##--- Loop for Polynomial orders 1 to 10 ---##
for order in range(1,11):
 error_rate_train_array_fold = []
 error_rate_val_array_fold = []
 # Random permutation of data
 Idx = np.random.RandomState(seed=8).permutation(Y.shape[0])
 # Loop 5 times for 5-fold
 for k in range(0,5):
  ##--- Prepare training, validation, and test data for the 5-fold ---#
  # Prepare indexing for each fold
  X_val = X[Idx[k*25:(k+1)*25]]
  Y_val = Y[Idx[k*25:(k+1)*25]]
  Idxtrn = np.setdiff1d(Idx, Idx[k*25:(k+1)*25])
  X_train = X[Idxtrn]
  Y_train = Y[Idxtrn]
```

| | k=0 | k=1 … | k=4 |
|---|---|---|---|
| X_val | Idx[0:25] | Idx[25:50] | Idx[100:125] |
| X_train | Idx[25:125] | Idx[0:25] and Idx[50:120] | Idx[0:100] |

## numpy.setdiff1d

numpy.setdiff1d(*ar1*, *ar2*, *assume_unique=False*)

Find the set difference of two arrays.

Return the unique values in *ar1* that are not in *ar2*.

https://scikit-learn.org/stable/modules/cross_validation.html

```python
##--- Polynomial Classification ---##
poly = PolynomialFeatures(order)
P = poly.fit_transform(X_train)
Pval = poly.fit_transform(X_val)
if P.shape[0] > P.shape[1]: # over-/under-determined cases
 reg_L = 0.00*np.identity(P.shape[1])
 inv_PTP = np.linalg.inv(P.transpose().dot(P)+reg_L)
 pinv_L = inv_PTP.dot(P.transpose())
 wp = pinv_L.dot(Y_train)
else:
 reg_R = 0.00*np.identity(P.shape[0])
 inv_PPT = np.linalg.inv(P.dot(P.transpose())+reg_R)
 pinv_R = P.transpose().dot(inv_PPT)
 wp = pinv_R.dot(Y_train)
##--- trained output ---##
y_est_p = P.dot(wp);
y_cls_p = [[1 if y == max(x) else 0 for y in x] for x in y_est_p ]
m1tr = np.matrix(Y_train)
m2tr = np.matrix(y_cls_p)
# training classification error count and rate computation
difference = np.abs(m1tr - m2tr)
error_train = np.where(difference.any(axis=1))[0]
error_rate_train = len(error_train)/len(difference)
error_rate_train_array_fold += [error_rate_train]
##--- validation output ---##
yval_est_p = Pval.dot(wp);
yval_cls_p = [[1 if y == max(x) else 0 for y in x] for x in yval_est_p ]
m1 = np.matrix(Y_val)
m2 = np.matrix(yval_cls_p)
# validation classification error count and rate computation
difference = np.abs(m1 - m2)
error_val = np.where(difference.any(axis=1))[0]
error_rate_val = len(error_val)/len(difference)
error_rate_val_array_fold += [error_rate_val]
error_rate_train_array += [np.mean(error_rate_train_array_fold)]
error_rate_val_array += [np.mean(error_rate_val_array_fold)]
```

```python
##--- plotting ---##
import matplotlib.pyplot as plt
order=[x for x in range(1,11)]
plt.plot(order, error_rate_train_array, color='blue', marker='o', linewidth=3,
label='Training')
plt.plot(order, error_rate_val_array, color='orange', marker='x', linewidth=3,
label='Validation')
plt.xlabel('Order')
plt.ylabel('Error Rates')
plt.title('Training and Validation Error Rates')
plt.legend()
plt.show()
```

# Question 7

Download the spambase data set from the UCI Machine Learning repository https://archive.ics.uci.edu/ml/machine-learning-databases/spambase/ and use the following function to pack the data: … function given to clean data

Randomly split the dataset into two parts, 80% for training and 20% for testing. Compute the test Classification Error Rate and the AUC based on the optimal linear regression model without regularization. In other words, use the training set to train a linear model and use the test set to check the classification performance in terms of Classification Error Rate, ROC and AUC.

**Hint:** to plot the ROC curve, the population of output predictions (say, values stored in vector `y_predict`) needs to be separated according to the two known output classes (0 or 1 values in the target vector `y_test`).

Let `y_predict_for_PosSamples` (when `y_test==1`) and `y_predict_for_NegSamples` (when `y_test==0`) denote these two populations of prediction.

Then compute the TPR (=1-FNR) and the FPR at various threshold/operating points in order to plot the ROC curve. To obtain the highest possible resolution for the ROC plot, you can set the decision threshold according to each element of the lower population of the two predicted classes of data (`y_predict_for_PosSamples` , `y_predict_for_NegSamples`).

# Question 7

## Error Rate , ROC,  AUC

STEPS ::

### Classification Error Rate Calculation

- Split data into training and test data (20% - 80%)
- Using x_train, train linear model with no regularization ➔ solve for w
  (training error not required in this qn)
- Using x_test and the linear model (w)
  - Calculate y_test_pred (also called predicted score, in decimal values)
    ➔ convert to 0 & 1 class outputs using sgn & threshold → calculate test error

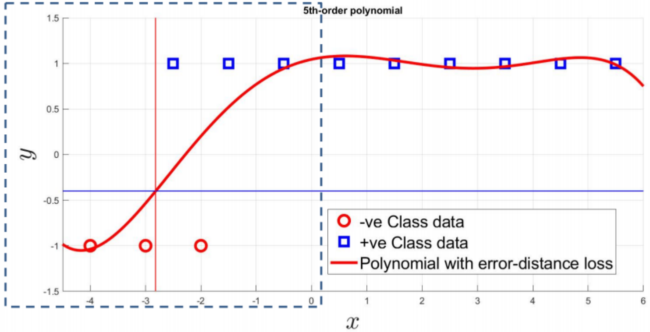
### ROC & AUC Calculation

- ROC :
  - TPR (FNR = 1-TPR)
  - FPR
- AUC = Sum of values of ROC Curve

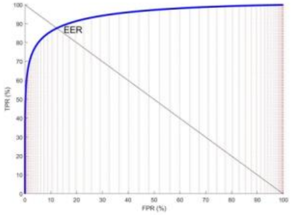
## Evaluation Metrics

**Classification**

| | N1 | N2 | P1 | N3 | P2 | P3 |
|---|---|---|---|---|---|---|
| y | -1.1 | -0.5 | -0.1 | 0.2 | 0.6 | 0.9 |




| | $\hat{P}$ (predicted) | $\hat{N}$ (predicted) |
|---|---|---|
| P (actual) | $TP = 2$ | $FN = 1$ |
| N (actual) | $FP = 1$ | $TN = 2$ |

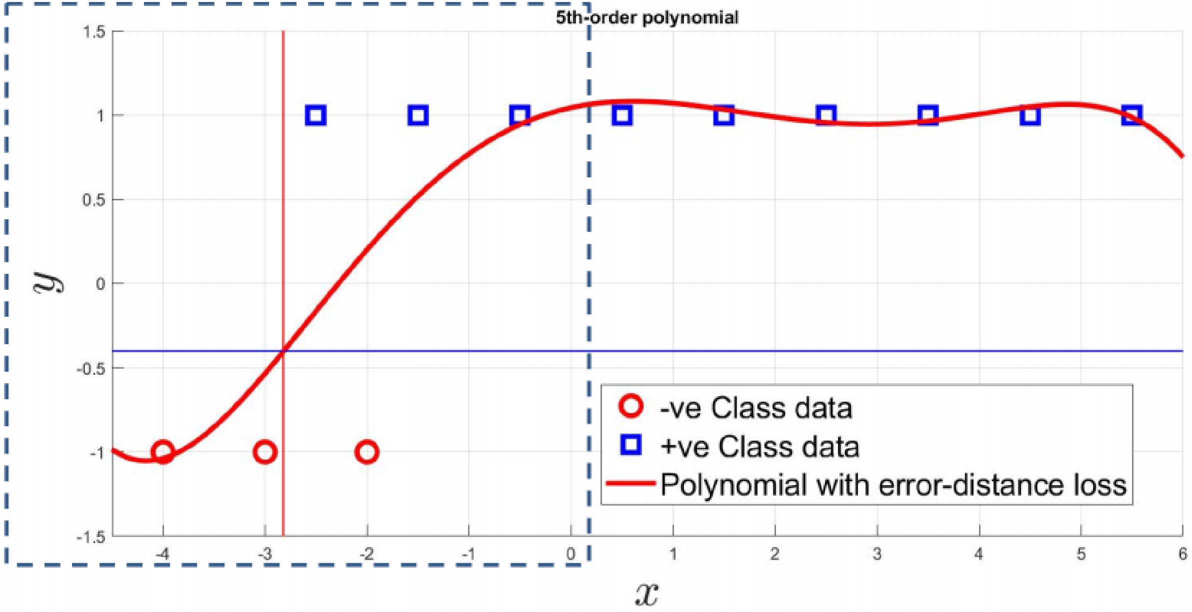



Receiver Operating Characteristic (ROC) Curve

# Evaluation Metrics

## Classification

| | N1 | N2 | P1 | N3 | P2 | P3 |
|---|---|---|---|---|---|---|
| y | -1.1 | -0.5 | -0.1 | 0.2 | 0.6 | 0.9 |

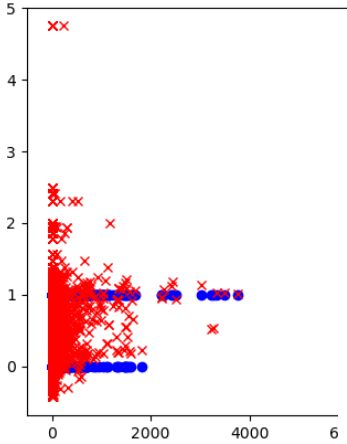


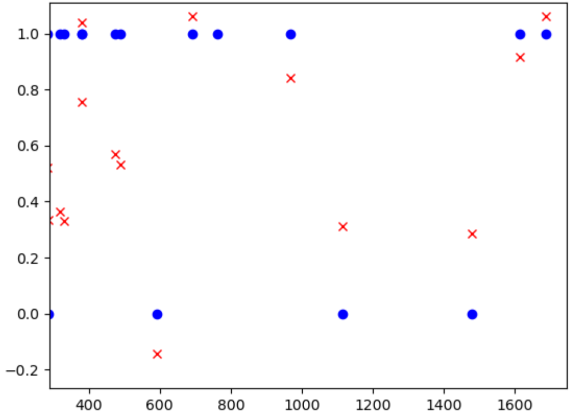

| | $\hat{P}$ (predicted) | $\hat{N}$ (predicted) |
|---|---|---|
| P (actual) | $TP = 2$ | $FN = 1$ |
| N (actual) | $FP = 1$ | $TN = 2$ |

# Preparation...

```
import numpy as np
from numpy.random import RandomState
from sklearn.model_selection import train_test_split
import matplotlib.pyplot as plt

## load data set
def load_data(Train=False):
    import csv
    data = []
    ## Read the training data
    f = open('spambase.data')
    reader = csv.reader(f)
    next(reader, None)
    for row in reader:
        data.append(row)
    f.close()
    ## x[:-1]: omit the last element of each x row
    X = np.array([x[:-1] for x in data]).astype(np.float)
    ## x[-1]: the first element from the right instead of from the left
    y = np.array([x[-1] for x in data]).astype(np.float)
    del data # free up the memory
    if Train:
        # returns X_train, X_test, y_train, y_test
        return train_test_split(X, y, test_size=0.2, random_state=8)
    else:
        return X, y
```



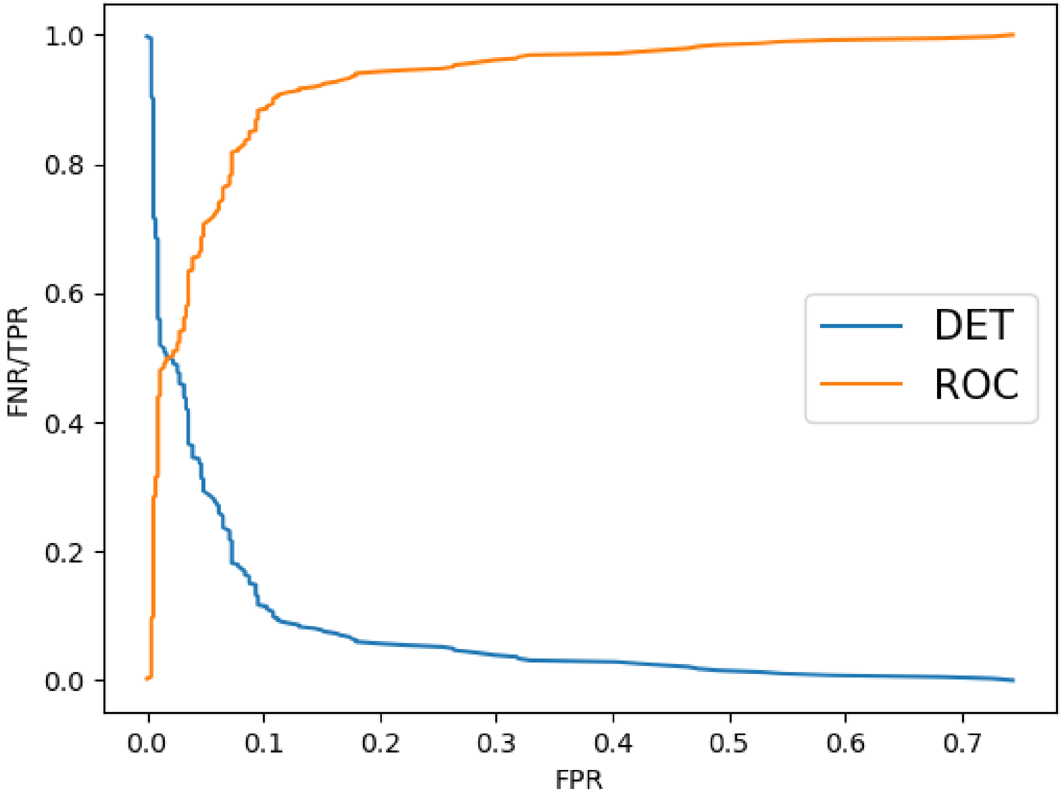




$$\hat{\mathbf{w}} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}$$

```
## Get training and test sets
X_train, X_test, y_train, y_test = load_data(Train=True)
## Linear regression
inv_XTX = np.linalg.inv(X_train.transpose().dot(X_train))
pinv = inv_XTX.dot(X_train.transpose())
W = pinv.dot(y_train)
## Prediction
y_predict = X_test.dot(W)
## Calculate classification error rate
yp_cls = [1 if yout >= 0 else 0 for yout in y_predict]
difference = np.abs(y_test - yp_cls)
test_error_count = (difference == 1).sum()
test_error_rate = test_error_count/len(y_test)
print(test_error_rate)
```

```
>>> print(y_predict)
[ 5.21970743e-01 -9.47271581e-02  2.85388006e-01  3.64771245e-02
  5.62578224e-01  8.39877181e-01  5.03194524e-01  2.31738718e-01
  5.68480382e-01 -1.43373161e-01  3.33307382e-01  5.58615423e-02
  1.84713858e-01  1.07125763e+00  2.07053489e-01  5.36811940e-01
  3.29518779e-01  7.86071178e-02  1.94773420e-01  1.17444646e+00
  3.39828857e-02  5.32183397e-01  1.29671636e-02  7.54377034e-01
  3.90815497e-02  3.58021208e-02  4.80841982e-01  1.29671636e-02
  2.12738543e-01  3.14619143e-01 -4.22198960e-01  2.74140860e-01
  1.06026202e+00 -2.92638858e-03  6.37023981e-01  9.15957178e-01
```

```python
##--- Compute FPR and FNR at different thresholds ---##
## separate the two classes of predicted data based on the ground truth y_test
pos_idx = np.where(y_test == 1) # identify the indexing of positive-class in the test set
neg_idx = np.where(y_test == 0) # identify the indexing of negative-class in the test set
y_predict_for_PosSamples = y_predict[pos_idx] # prediction of the positive-class data
y_predict_for_NegSamples = y_predict[neg_idx] # prediction of the negative-class data
## use the shorter among the two arrays as threshold
if ( len(y_predict_for_PosSamples) <= len(y_predict_for_NegSamples) ):
    sorted = np.sort(y_predict_for_PosSamples) # sort in ascending order to be used as threshold
else:
    sorted = np.sort(y_predict_for_NegSamples) # sort in ascending order to be used as threshold
FNR = []
FPR = []
TPR = []
## Compute FNR, FPR, and TPR for each threshold
for k in range(len(sorted)):
    yp_cls_pos = np.abs([1 if yout >= sorted[k] else 0 for yout in y_predict_for_PosSamples])
    yp_cls_neg = np.abs([1 if yout >= sorted[k] else 0 for yout in y_predict_for_NegSamples])
    FNR += [(yp_cls_pos == 0).sum()/len(y_predict_for_PosSamples)]
    FPR += [(yp_cls_neg == 1).sum()/len(y_predict_for_NegSamples)]
    TPR += [1-(yp_cls_pos == 0).sum()/len(y_predict_for_PosSamples)]
##--- Plot ROC and DET curves ---##
plt.plot(FPR, FNR, '-', label = 'DET')
plt.plot(FPR, TPR, '-', label = 'ROC')
plt.xlabel('FPR')
plt.ylabel('FNR/TPR')
plt.legend(fontsize=15)
```


1.0
0.8
0.6
0.4
0.2
0.0
FNR/TPR
0.0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
FPR
DET
ROC

```python
##--- Comput AUC ---##
ypos_array = [[1 if y_predict_for_PosSamples[j] >= y_predict_for_NegSamples[k] else
               0 for j in range(len(y_predict_for_PosSamples))] for k in range(len(y_predict_for_NegSamples))]
AUC = np.sum(ypos_array)/(len(y_predict_for_PosSamples)*len(y_predict_for_NegSamples))
print(AUC)
```

```python
ypos_array_1=[[0 for i in range(len(y_predict_for_PosSamples))] for j in range(len(y_predict_for_NegSamples))]

for j in range(len(y_predict_for_PosSamples)) :
    for k in range(len(y_predict_for_NegSamples)) :
        if y_predict_for_PosSamples[j] >= y_predict_for_NegSamples[k] :
            ypos_array_1[k][j]= 1
        else :
            ypos_array_1[k][j]= 0
```

$\{\vec{x}_j^+, \vec{x}_k^-\}$. The AUC of a model on a given dataset can be expressed as the probability that for such a random observation pair, the score of the minority class observation is greater than that of the majority class observation (Bamber, 1975). If the model is linear, with the coefficients of the predictor variables given by vector $\vec{\beta}$, then ignoring ties,

$$AUC(\vec{\beta}) = \Pr(\vec{\beta}.\vec{X}^+ > \vec{\beta}.\vec{X}^-).$$

https://icml.cc/Conferences/2004/proceedings/papers/132.pdf

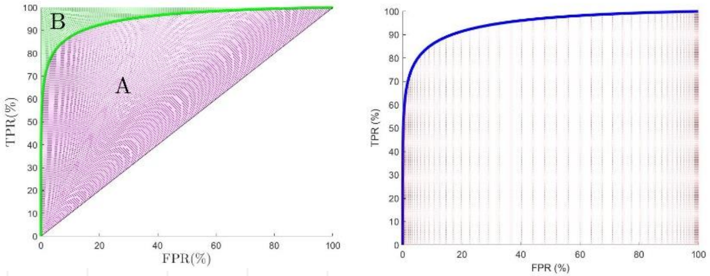
**Question 1:**

We have two classifiers showing the same accuracy in a leave-one-out evaluation. The more complex model (such as a $9^{th}$-order polynomial model) is preferred over the simpler one (such as a $2^{nd}$-order polynomial model).

a) True
b) False

**Question 2:** According to the plots below, the Gini Coefficient is equal to Two times the Area Under the ROC minus One.

a) True
b) False
Answer: a).
Reason: Since the area (A+B) = ½ (half the square box of area 1), Gini-coefficient = A/(A+B) = 2 A.
AUC = A+1/2 => A = AUC – 0.5. Substitute A into the Gini above: Gini-coefficient = 2(AUC – 0.5).




The **AUC (Area Under Curve)** is the area enclosed by the ROC curve. A perfect classifier has AUC = 1 and a completely random classifier has AUC = 0.5. Usually, your model will score somewhere in between.

The Gini Coefficient is 2*AUC – 1, and its purpose is to normalize the AUC so that a random classifier scores 0, and a perfect classifier scores 1.

**Question 3:**

Suppose the binary classification problem, which you are dealing with, has highly imbalanced classes. The majority class has 99 hundred samples and the minority class has 1 hundred samples. Which of the following metric(s) would you choose for assessing the classification performance? (Select all relevant metric(s) to get full credit)

a) Classification Accuracy

b) Cost sensitive accuracy

c) Precision and recall

e) None of these

Answer: (b, c)

**Question 4:**

Given below is a scenario for Training error rate Tr, and Validation error rate Va for a machine learning algorithm. You want to choose a hyperparameter (P) based on Tr and Va.

Which value of P will you choose based on the above table?

a) 10

b) 9

c) 8

d) 7

e) 6

Answer: e).

| P | Tr | Va |
|---|---|---|
| 10 | 0.10 | 0.25 |
| 9 | 0.30 | 0.35 |
| 8 | 0.22 | 0.15 |
| 7 | 0.15 | 0.25 |
| 6 | 0.18 | 0.15 |