Lesson 13 Confusion Matrix and ROC Last Update April 18

Confusion Matrix

A Confusion matrix is an N x N matrix used for evaluating the performance of a classification model, where N is the number of target classes. A confusion matrix is a summary of prediction results on a classification problem. The matrix compares the actual target values with those predicted by the machine learning model. The number of correct and incorrect predictions are summarized with count values and broken down by each class. A class is a output category, label or target value **that we want** to predict based on the values of some other attributes, In a binary classifier the output classes are 0's and 1's representing **false** and **true** or **positive** and **negative.** In other classifiers the classes may be home sale prices, or average, medium and/or high prices.

For a binary classification problem, a confusion matrix would be a 2 x 2 matrix as shown below with 4 values:

- The target has two classes: **Positive (True)**or **Negative (False)**
- The **columns** represent the **actual values** of the target variable
- The **rows** represent the **predicted values** of the target variable

Confusion Matrix

Number Predictions	**Positive Predicted**	**Negative Predicted**
True Actual	True Positive (TP)	False Negative (FN)
False Actual	False Positive (FP)	True Negative (TN)

True Positive (TP)

- The predicted value matches the actual value (P,P)
- The actual value was positive and the model predicted a positive value

True Negative (TN)

- The predicted value matches the actual value (N,N)
- The actual value was negative and the model predicted a negative value

False Positive (FP) – Type 1 error

- The predicted value was falsely predicted (N,P)
- The actual value was negative but the model predicted a positive value
- Also known as the **Type 1 error**

False Negative (FN) – Type 2 error

- The predicted value was falsely predicted (P,N)
- The actual value was positive but the model predicted a negative value
- Also known as the **Type 2 error**

We can chart the confusion matrix values for your understanding:

	actual	**Predicted**	**Description**
TP	**P**	**P**	**P** expected got **P**
TN	**N**	**N**	**N** expected got **N**
FP	**N**	**P**	**N** expected but got **P** instead
FN	**P**	**N**	**P** expected but got **N** instead

Uses of confusion matrix

A confusion matrix can be used to predict how many people are infected with a contagious disease in times before they show the symptoms, and isolate them from the healthy population. The two values for our target variable would be: Sick and Not Sick.

Confusion matrix metrics:

A confusion matrix has many available metric calculations used to measure classifier performance.

The Metric rates that are computed from a confusion matrix are:

Accuracy

Accuracy tells us how often the classifier, is correct.

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN}$$

Precision

Precision tells us how many of the correctly predicted cases actually turned out to be positive. This would determine whether our model is reliable or not. Precision is a useful metric in cases where **False Positive (FP)** is a higher concern than **False Negatives (FN).**

$$\text{Precision} = \frac{TP}{TP + FP}$$

Recall

Recall tells us how many of the actual positive cases we were able to predict correctly with our model. Recall is a useful metric in cases where **False Negative (FN)** trumps **False Positive(FP)**.

$$\text{Recall} = \frac{TP}{TP + FN}$$

F1-Score

F1-score is a harmonic mean of **Precision** and **Recall**, and it gives a combined idea about these two metrics. It is maximum when Precision is equal to Recall.

$$\text{F1 - Score} = \frac{2 \times \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

Confusion Matrix Calculation Example

Here is a table that has 10 predictions for actual values with calculated outcomes TP, TN, FP and FN

Index	**Actual**	**Predicted**	**Outcome**	**Comment**
1	1	1	TP	both actual and predicted true
2	0	0	TN	both actual and predicted false
3	0	0	TN	both actual and predicted false
4	1	1	TP	both actual and predicted true
5	0	0	TN	both actual and predicted false
6	0	0	TN	both actual and predicted false
7	1	0	FP	actual true but predicted false
8	0	1	FN	actual false but predicted true
9	0	0	TN	both actual and predicted false
10	1	0	FP	actual true but predicted false

The total outcome values are:

TP = 2, TN = 5, FP = 2, FN = 1

The Confusion Matrix would be:

N = 10	**Positive Predicted**	**Negative Predicted**
True Actual	TP = 2	FN = 1
False Actual	FP = 2	TN = 5

$$\text{Accuracy} = \frac{TP + TN}{TP + FP + TN + FN} = \frac{2 + 5}{2 + 2 + 5 + 1} = \frac{7}{10} = .7\ (70\%)$$

CONFUSION MATRIX ROC HOMEWORK 1

N = 10	**Positive Predicted**	**Negative Predicted**
True Actual	TP = 2	FN = 1
False Actual	FP = 2	TN = 5

Using the above Confusion matrix, in a python program calculate and print out the values for Precision, Recall and F1-score. Put your program in a python file called confusion_roc_homework1.py

Use the following formulas to calculate your answers:

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

$$\text{F1 - Score} = \frac{2 \times \text{Precision} * \text{Recall}}{\text{Precision} + \text{Recall}}$$

You should get something like this:

Confusion Matrix ROC Homework 1

```
Precision:  0.5
Recall:  0.6666666666666666
F1Score:  0.5714285714285715
```

Calculating a Confusion Matrix using Sklearn

Sklearn has the **confusion_matrix** function for calculating a confusion matrix. The confusion matrix returns a 2D array of TP, FN, FP, TN. Sklearn also has metrics functions for calculating accuracy, recall, precision and f1-score. We can plot a confusion matrix, by putting the 2d confusion matrix into a data frame and then plot with a seaborn heatmap all at the same time like this:

sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')

Here is the complete sklearn program:

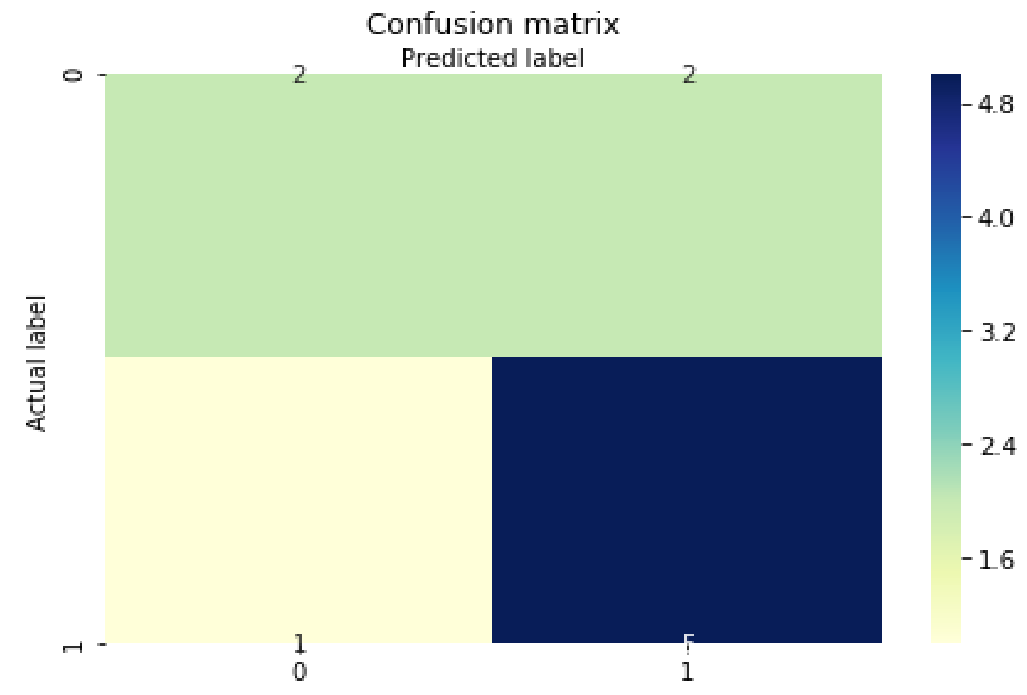
```
"""
confusionmatrix.py
confusion matrix
"""
# confusion matrix in sklearn
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

# actual values
actual = [1,0,0,1,0,0,1,0,0,1]

# predicted values
predicted = [1,0,0,1,0,0,0,1,0,0]
```

```
# calculate and print confusion matrix
cnf_matrix = confusion_matrix(actual,predicted, labels=[1,0])
print('Confusion matrix : \n',cnf_matrix)

# print outcome values reshape 2d array to a 1d array
tp, fn, fp, tn = confusion_matrix(actual,predicted,labels=[1,0]).reshape(-1)
print("Outcome values:")
print("TP: ", tp, " FN: ", fn, " FP: ",fp, " TN: ", tn)

# plot confusion matrix as a heat map using seaborn
class_names=[0,1]
fig, ax = plt.subplots()
tick_marks = np.arange(len(class_names))
plt.xticks(tick_marks, class_names)
plt.yticks(tick_marks, class_names)

# put confusion matrix in a datafram and call seaborn heatmap function
sns.heatmap(pd.DataFrame(cnf_matrix), annot=True, cmap="YlGnBu" ,fmt='g')
ax.xaxis.set_label_position("top")
plt.tight_layout()
plt.title('Confusion matrix', y=1.1)
plt.ylabel('Actual label')
plt.xlabel('Predicted label')
plt.show()

# print out statistics
print("Accuracy:",accuracy_score(actual, predicted))
print("Precision:",precision_score(actual, predicted))
print("Recall:",recall_score(actual, predicted))
print("F1-Score:",f1_score(actual, predicted))
```

Here is the program Output

```
Confusion matrix :
 [[2 2]
 [1 5]]
Outcome values:
TP:  2  FN:  2  FP:  1  TN:  5
Accuracy: 0.7
Precision: 0.6666666666666666
Recall: 0.5
F1-Score: 0.5714285714285715
```

Here if the Confusion Matrix Heatmap Plot:



Sklearn confusion matrix classification report

The Sklearn confusion matrix classification report is a tabular report showing the main classification metrics. Here is the code to make a confusion matrix classification report using sklearn.

```
# sklearn confusion matric classification report
from sklearn.metrics import classification_report

# actual values
actual = [1,0,0,1,0,0,1,0,0,1]

# predicted values
predicted = [1,0,0,1,0,0,0,1,0,0]

# class targtest 0 and 1
target_names = ['class 0', 'class 1']

print(classification_report(actual, predicted, target_names=target_names))
```

Here is the classification report using the above actual and predicted values:

```
              precision    recall  f1-score   support

     class 0       0.71      0.83      0.77         6
     class 1       0.67      0.50      0.57         4

   micro avg       0.70      0.70      0.70        10
   macro avg       0.69      0.67      0.67        10
weighted avg       0.70      0.70      0.69        10
```

The **precision** is the ratio tp / (tp + fp) where tp is the number of true positives and fp the number of false positives. The precision is intuitively the ability of the classifier not to label as positive a sample that is negative.

The **recall** is the ratio tp / (tp + fn) where tp is the number of true positives and fn the number of false negatives. The recall is intuitively the ability of the classifier to find all the positive samples.

The **F-beta** score can be interpreted as a weighted harmonic mean of the precision and recall, where an F-beta score reaches its best value at 1 and worst score at 0.

The F-beta score weights recall more than precision by a factor of beta.
beta == 1.0 means recall and precision are equally important.

The support is the number of occurrences of each class in y_true.

The reported averages include macro average (averaging the unweighted mean per label), weighted average (averaging the support-weighted mean per label), and sample average (only for multilabel classification). Micro average (averaging the total true positives, false negatives and false positives) is only shown for multi-label or multi-class with a subset of classes, because it corresponds to accuracy otherwise.

Precision Recall Curve

The precision-recall curve shows the tradeoff between **precision** and **recall** for different threshold. A high area under the curve represents both **high recall** and **high precision**. **High precision** relates to a low false positive rate, and **high recall** relates to a low false negative rate. High scores for both, show that the classifier is returning accurate results (high precision), as well as returning a majority of all positive results (high recall).

A system with high recall, but low precision returns many results, but most of its predicted labels are incorrect when compared to the training labels. A system with high precision but low recall is just the opposite, returning very few results, but most of its predicted labels are correct when compared to the training labels. An ideal system with high precision and high recall will return many results, with all results labeled correctly.

Recall is a performance measure of the whole positive part of a dataset.

Precision is a performance measure of positive predictions.

Plotting Precision-Recall Curve from test results

To plot a **Precision-Recall** curve we use the following actual values resulting from the following predicted probability score values.

```
# actual values
actual = [1,1,1,1,0,1,0,1,0,0]

# predicted probability score values
scores = [1.0,.95,.9,.89,.77,.69,54,.34,.24,.12];
```

We first make a chart where we calculate TP FP and FN from the Yes and No's.

TP is the Cumulative sum of the Yes where as FP is the cumulative sums of the Nos. FN is the cumulative sum in reverse starting from the totals of the No’s.

We calculate Precision and Recall from the TP, FP and FN formulas.

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

						Precision	**Recall**
Index	**Probability scores**	**Actual (Sick)**	**Yes (sum) TP**	**No (sum) FP**	**-No (sum) FN**	**TP / TP + FP**	**TP / TP + FN**
1	1.0	1	1	0	4	1/1	1/5
2	.95	1	2	0	4	2/2	2/6
3	.9	1	3	0	4	3/3	3/7
4	.89	1	4	0	4	4/4	4/8
5	.77	0	4	1	3	4/5	4/7
6	.69	1	5	1	3	5/6	5/8
7	.54	0	5	2	2	5/7	5/7
8	.34	1	6	2	2	6/7	6/8
9	.24	0	6	3	1	6/8	6/7
10	.12	0	6	4	0	6/10	6/6
		Total:	6	4			

From the chart Precision and Recall columns we can plot the **Precision-Recall** curve **.**

We write a program to calculate **Precision-Recall** and plot the above chart using the actual values:

actual = [1,1,1,1,0,1,0,1,0,0]

and the predicted probability score values:

scores = [1.0,.95,.9,.89,.77,.69,54,.34,.24,.12];

Here is the program:

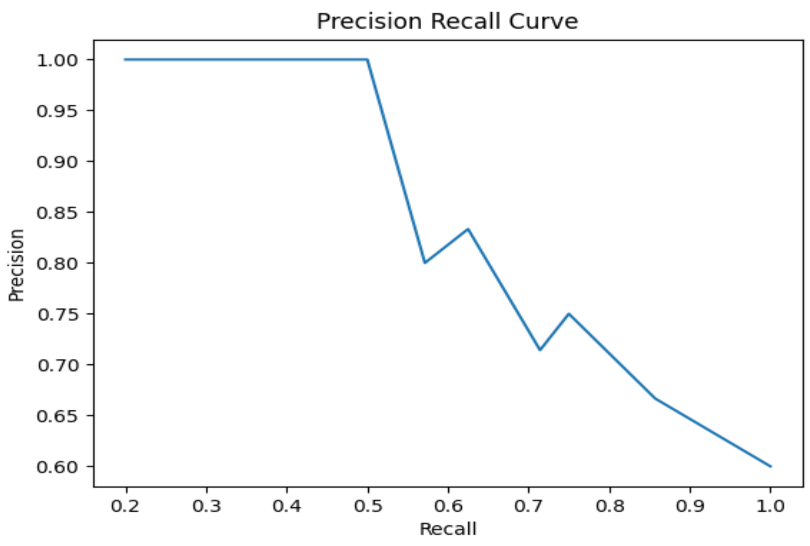
```
#  plot Precision and Recall Curve
import matplotlib.pyplot as plt
import numpy as np

print("Calculate Precision-Recall Curve from Actual Data")

# precision
precision = []
# recall
recall = []

# actual values
actual = [1,1,1,1,0,1,0,1,0,0]

TP = 0
FP = 0
FN = len(actual) - sum(actual)

for i in range(len(actual)):

   # cummulate tp
   if actual[i] == 1:
      TP = TP + 1

   # commulate fp
   if actual[i] == 0:
      FP = FP + 1
      FN = FN - 1
```

```
    # append precision and recall
    precision.append(TP/(TP+FP))
    recall.append(TP/(TP+FN))

  # print results
  print("Precision: ")
  print(precision)
  print("Recall: ")
  print(recall)

  # plot results
  plt.plot(recall, precision)
  plt.title('Precision Recall Curve')
  plt.xlabel('Recall')
  plt.ylabel('Precision')
  plt.show()
```

Here is the Program output:

```
Calculate Precision-Recall Curve from Actual Data

Precision :
[1.0, 1.0, 1.0, 1.0, 0.8, 0.8333333333333334, 0.7142857142857143, 0.75, 0.6666666666666666, 0.6]

Recall:
[0.2, 0.3333333333333333, 0.42857142857142855, 0.5, 0.5714285714285714, 0.625, 0.7142857142857143, 0.75,
0.8571428571428571, 1.0]
```

Here is the Plot:

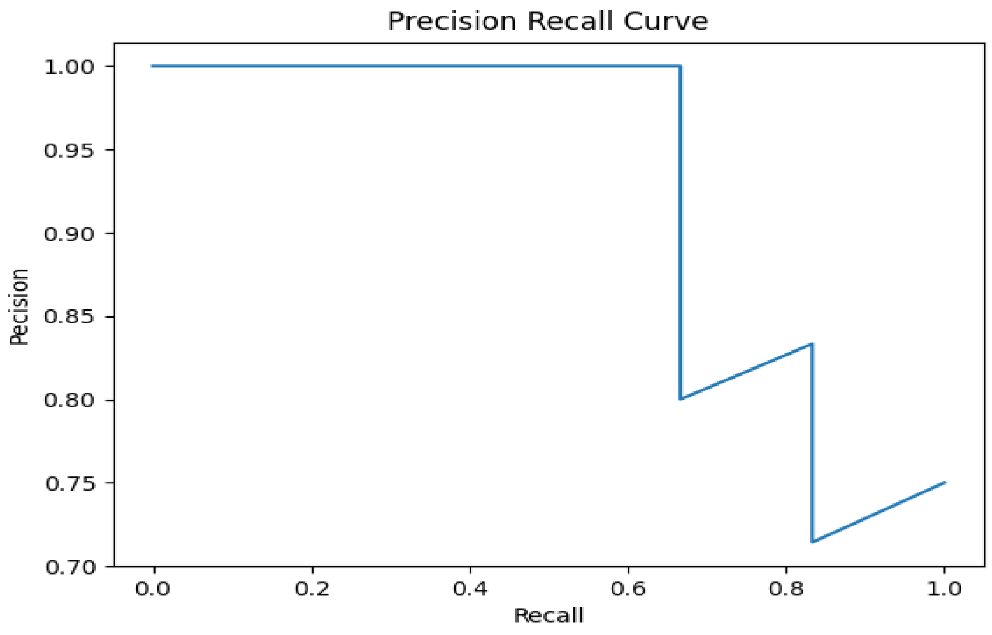


Calculating Precision – Recall Curve using sklearn

Sklearn has the **precision_recall_curve** function located in from sklearn.metrics precision_recall_curve. The **precision_recall_**curve takes the actual value and the probability scores**.** We will use the same actual and scores as we used in plotting the previous ROC curve.

```
# calculating prescion recall curve using sklearn
from sklearn.metrics import precision_recall_curve
import numpy as np
import matplotlib.pyplot as plt

print("Precision Recall Curve")

# true values
actual = np.array([1,1,1,1,0,1,0,1,0,0])

# probability scores
scores = np.array([1.0,.95,.9,.89,.77,.69,54,.34,.24,.12]);

# calculate precision, recall and thresholds
precision, recall, thresholds = precision_recall_curve(actual,  scores)

# print results
print("precision:",precision)
print("recall:",recall)
print("thresholds:",thresholds)

# plot precision recall and print AUC in title
plt.plot(recall, precision)
plt.title('Sklearn Precision Recall Curve')
plt.ylabel('Precision')
plt.xlabel('Recall')
plt.show()
```

Here is the plot:



CONFUSION MATRIX HOMEWORK QUESTION 2

Calculate and plot a precision_recall_curve using the following actual and predicted values from homework Question 1.

```
# actual values
actual = [1,0,0,1,0,0,1,0,0,1]

# predicted values
predicted = [1,0,0,1,0,0,0,1,0,0]
```

Step 0: put the above actual and predicted values in your program

```
# actual values
actual = [1,0,0,1,0,0,1,0,0,1]

# predicted values
predicted = [1,0,0,1,0,0,0,1,0,0]
```

Step1: estimate probabilities for actual and predicted values

We do not have the probabilities for each value, but you have 4 options to calculate them:

(1) calculate the probabilities from the confusion matrix
(2) estimate probabilities from calculated recall and precision values
(3) hard code some values
(4) make some random values like this:

```
test_probs = np.random.rand(10)
```

Note: Each output value will have its own accuracy probability between 0 and 1. Usually the classifier provides these probabilities for you. We have only the actual and predicted output values.

Step 2: create calculate_precision_recall function

Make a function that will calculate recall and precision from the input actual values and the predicted values and return them.

Count the TP's TN's and FN's using the following chart

	Actual	**Predicted**	**Description**
TP	**P**	**P**	**P** expected got **P**
TN	**N**	**N**	**N** expected got **N**
FP	**N**	**P**	**N** expected but got **P** instead
FN	**P**	**N**	**P** expected but got **N** instead

Use these formulas for calculating Precision and Recall.

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$\text{Recall} = \frac{TP}{TP + FN}$$

Step 3: calculate precision and recall using the calculate_precision_recall function

Using the calculate_recall_precision function calculate precision recall for the actual and predicted sample values.

Print out the calculated values for precision and recall.

You should get the following results:

```
precision:  0.6666666666666666
recall:  0.5
```

Step 4: generate test thresholds

Make a table of 100 probability test thresholds between 0 and 1 using linspace

```
thresholds = np.linspace(0, 1, num=100)
```

Step 5: make empty lists recalls and precisions

Make a list to store recalls
Make a list to store precisions

```
precisions = []
recalls = []
```

Step 6: calculate lists of precisions and recalls

```
In a outer Loop go through each threshold
    make a list called test_preds (test predictions)
    In a Inner Loop go through each actual value
        if the probably for each actual value is greater than the
        threshold then append 1 to test_preds list
```

if the probability for each actual value is less or equal than the threshold then append 0 to test_preds list

At the end of inner loop iterations calculate precision and recall and append to the precisions and recalls list

Step 7: plot precisions and recall curve

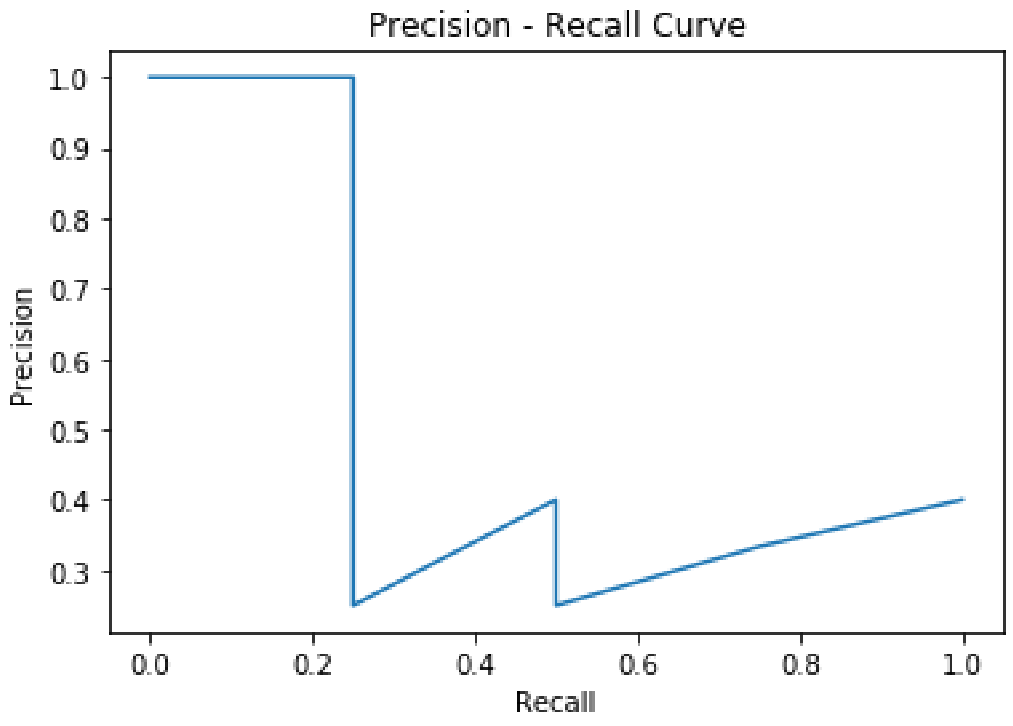
After all iterations have completed, plot the precision recall curve stored in the precision and recalls list.

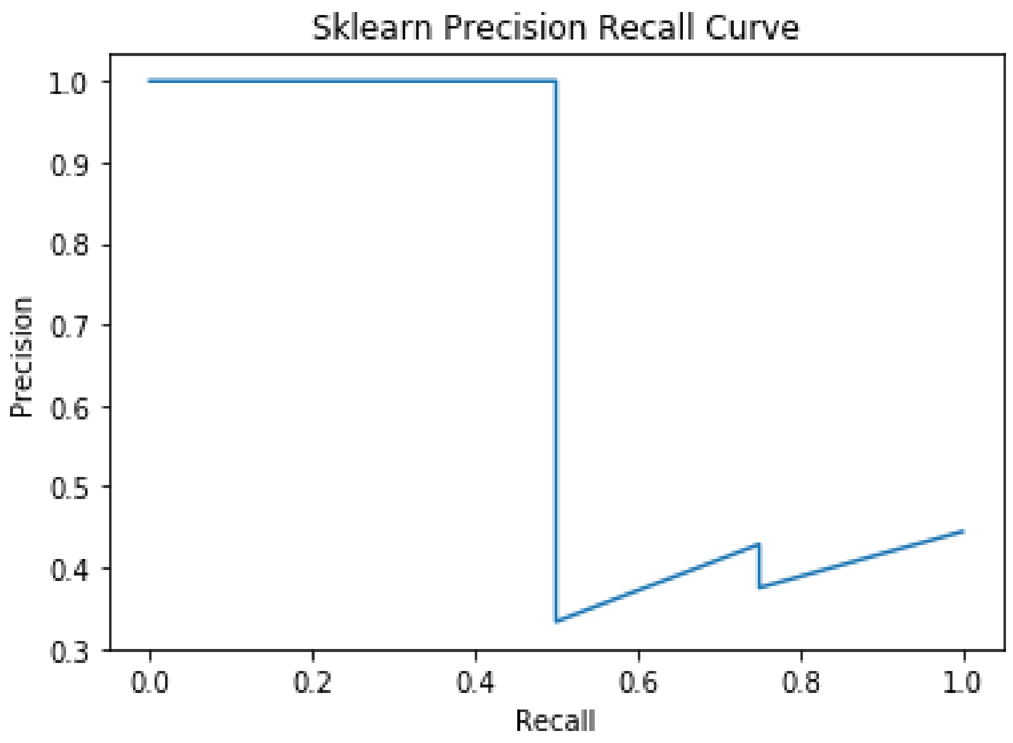
Step 8:

Use **sklearn** to plot the same data using the actual values and using your estimated probabilities.

Put your answer in a py file called confusion_roc_homework2.py

You should get something like this:





Receiver Operator Characteristic (ROC)

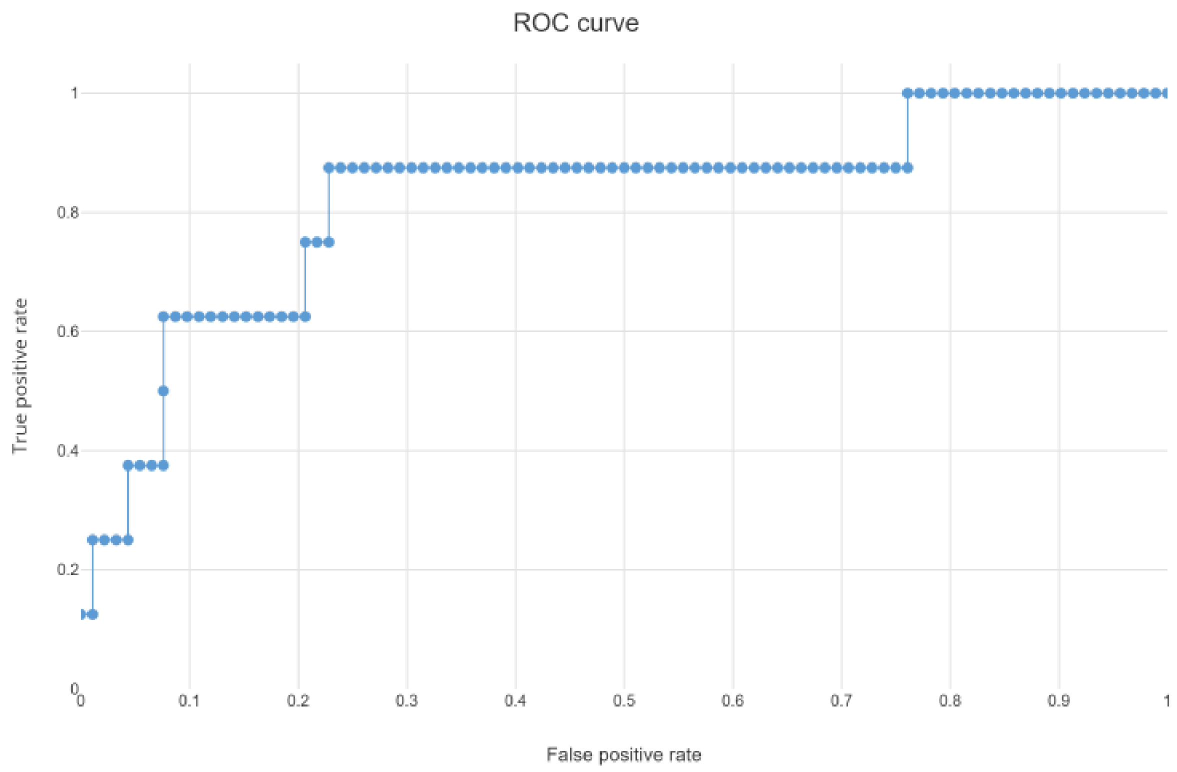
A **Receiver Operator Characteristic (ROC)** curve is a graphical plot used to show the diagnostic ability of binary classifiers. The ROC curve shows the trade-off between **sensitivity** True Positive Rate (or TPR) and **specificity** 1-False Positive Rate(1 – FPR).

The **True Positive Rate** (TPR) is the proportion of observations that were correctly predicted to be positive out of all positive observations (TP/(TP + FN)).

The **False Positive Rate** (FPR) is the proportion of observations that are incorrectly predicted to be positive out of all negative observations (FP/(TN + FP)).

For example, in medical testing, the true positive rate is the rate in which people are correctly identified to test positive for the disease in question.

Example ROC Curve



Creating an ROC Curve

A ROC curve is constructed by plotting the **True Positive Rate** (TPR) against the **False Positive Rate** (FPR).

The **True Positive Rate** is the proportion of observations that were correctly predicted to be positive out of all positive observations (TP/(TP + FN)).

The **False Positive Rate** is the proportion of observations that are incorrectly predicted to be positive out of all negative observations (FP/(TN + FP)).

A discrete classifier that returns only the predicted class gives a single point on the ROC space. But for probabilistic classifiers, which give a probability or score that reflects the degree to which an instance belongs to one class rather than another, we can create a curve by **varying** the threshold for the score.

The ROC curve shows the trade-off between sensitivity (or TPR) and specificity (1 – FPR). Classifiers that give curves closer to the top-left corner indicate a better performance. As a baseline, a random classifier is expected to give points lying along the diagonal (FPR = TPR). The closer the curve comes to the 45-degree diagonal of the ROC space, the less accurate the test.

Note that the ROC does not depend on the class distribution. This makes it useful for evaluating classifiers predicting rare events such as diseases or disasters. In contrast, evaluating performance using accuracy (TP +TN)/(TP + TN + FN + FP) would favor classifiers that always predict a negative outcome for rare events.

Sensitivity ("positivity in disease") True Positive Rate (TPR) refers to the proportion of subjects who have the target condition (reference standard positive) and give positive test results.

A simple example would be to determine what proportion of the actual sick people were correctly detected by the model.

$$\text{Sensitivity (TPR)} = \frac{TP}{(TP + FN)}$$

Specificity ("negativity in health") True Negative Rate (TNR) is the proportion of subjects without the target condition and give negative test results.

Specificity (TNR) tells us what proportion of the negative class got correctly classified. Specificity would mean determining the proportion of healthy people who were correctly identified by the model

$$\text{Specificity (TNR)} = \frac{TN}{(TN + FP)}$$

Note: Specificity is also known as (1-FPR)

False Positive Rate (FPR)

FPR tells us what proportion of the negative class got incorrectly classified by the classifier.

A higher TNR and a lower FPR is desirable since we want to correctly classify the negative class.

$$FPR = \frac{FP}{(TP + FN)}$$

False Negative Rate (FNR)

False Negative Rate (FNR) tells us what proportion of the positive class got incorrectly classified by the classifier.

A higher TPR and a lower FNR is desirable since we want to correctly classify the positive class.

$$FNR = \frac{FN}{(TP + FN)}$$

Positive Predictive Value (PPV)

Positive predictive value is the proportion of positive results that are true positives (i.e. have the target condition)

Positive predictive value (PPV) = TP / (TP + FP)

Negative Predictive Value (NPV)

Negative predictive value is the proportion of negative results that are true negatives (i.e. do not have the target condition).

Negative predictive value (NPV) = TN / (TN + FN)

Positive Likelihood Ratio (LR+)

Positive Likelihood Ratio (LR+) is the ratio of the proportion of patients who have the target condition and test positive to the proportion of patients without the target condition who also test positive

Positive likelihood ratio (LR+) = sensitivity / (1 – specificity)

Negative Likelihood Ratio (LR-)

Negative Likelihood Ratio (LR-) is the ratio of the proportion of patients who have the target condition who test negative to the proportion of patients without the target condition who also test negative

Negative likelihood ratio (LR−) = (1 – sensitivity) / specificity

We can put our findings in the following chart

	Disease Present +	Disease Absent -	Total
Index Test Positive +	True Positive (TP)	False Positive (FP)	TP + FP
Index Test Negative -	False Negative (FN)	True Ngative (TN)	TN+FN
Total	**TP+FN**	**TN+FP**	

Probability of Predictions

A machine learning classification model can be used to predict the actual class of the data point directly or predict its probability of belonging to different classes. Setting different thresholds for classifying positive class for data points will inadvertently change the **Sensitivity** and **Specificity** of the model. And one of these thresholds will probably give a better result than the others, depending on whether we are aiming to lower the number of False Negatives or False Positives.

The **Receiver Operator Characteristic (ROC)** curve is an evaluation metric for binary classification problems. It is a probability curve that plots the **TPR** against **FPR** at various threshold values and essentially separates the 'signal' from the 'noise'.

The **Area Under the Curve (AUC)** is the measure of the ability of a classifier to distinguish between classes and is used as a summary of the ROC curve.

When AUC = 1, then the classifier is able to perfectly distinguish between all the Positive and the Negative class points correctly. If the AUC had been 0, then the classifier would be predicting all Negatives as Positives, and all Positives as Negatives.

When 0.5<AUC<1, there is a high chance that the classifier will be able to distinguish the positive class values from the negative class values. This is so because the classifier is able to detect more numbers of True positives and True negatives than False negatives and False positives.

So, the higher the AUC value for a classifier, the better its ability to distinguish between positive and negative classes.

In a ROC curve, a higher X-axis value indicates a higher number of False positives than True negatives. While a higher Y-axis value indicates a higher number of True positives than False negatives. So, the choice of the threshold depends on the ability to balance between False positives and False negatives.

The Confusion matrix corresponds to a single point on your ROC Curve: You would need many Confusion matrixes to make 1 ROC curve. You cannot make a ROC curve from a single Confusion Matrix.

A better way is the post the results for each threshold, The threshold may be concentration of some prescription dosage, that results that the patient is not sick and not sick.

We now calculate ROC curve

We first make a table with different probability scores where the actual sick people are known We then cumulative the yes's and no's.

Since we know the probability scores we can calculate sensitivity TPR as

$$\text{Sensitivity (TPR)} = \frac{\text{Cumulative sum Yes}}{\text{total Yes (TP+FN)}}$$

```
                          Cumulative sum No
Specifity  (TNR)  =     --------------------------
                           total No (FP + TN)
```

Cumulative sum means you simply add the yes and no counts per index

```
# actual values
actual = [1,1,1,1,0,1,0,1,0,0]

# predicted probability score values
scores = [1.0,.95,.9,.89,.77,.69,54,.34,.24,.12];
```

Index	Probability scores	Actual (sick)	Yes (cumsum)	No (cumsum)	TPR sensitivity	FPR (1-specify)
1	1.0	1	1	0	1/6	0/2
2	.95	1	2	0	2/6	0/2
3	.9	1	3	0	3/6	0/2
4	.89	1	4	0	4/6	0/2
5	.77	0	4	1	4/6	1/2
6	.69	1	5	0	5/6	0/2
7	.54	0	5	2	5/6	2/2
8	.34	1	6	2	6/6	2/2
9	.24	0	6	0	6/6	2/2
10	.12	0	6	0	6/6	2/2
		Total:	6	2		

Each index in the table is an entry to the ROC curve

We calculate the ROC curve from the table as follows

We first total the number of sick peoples (yes). and the total number of not sick people (NO)

Total yes = 6
Total no = 2

We then calculate each entry per index

We calculate TPR from the cumulative yes per index

$$\text{Sensitivity (TPR)} = \frac{\text{Cumulative yes}}{\text{total Yes (TP + FN)}} = \frac{1}{6} = .16666$$

$$\text{Specifity (TNR)} = \frac{\text{Cumulative No}}{\text{total No (FP + TN)}} = \frac{0}{6} = 0$$

The first entry(TNR,TPR) in the ROC table lower left bottom corner will be (0,.16666)

We can make a small program to calculate Sensitivity (TPR) and Sensitivity (TPR) from our table.

Here is the code:

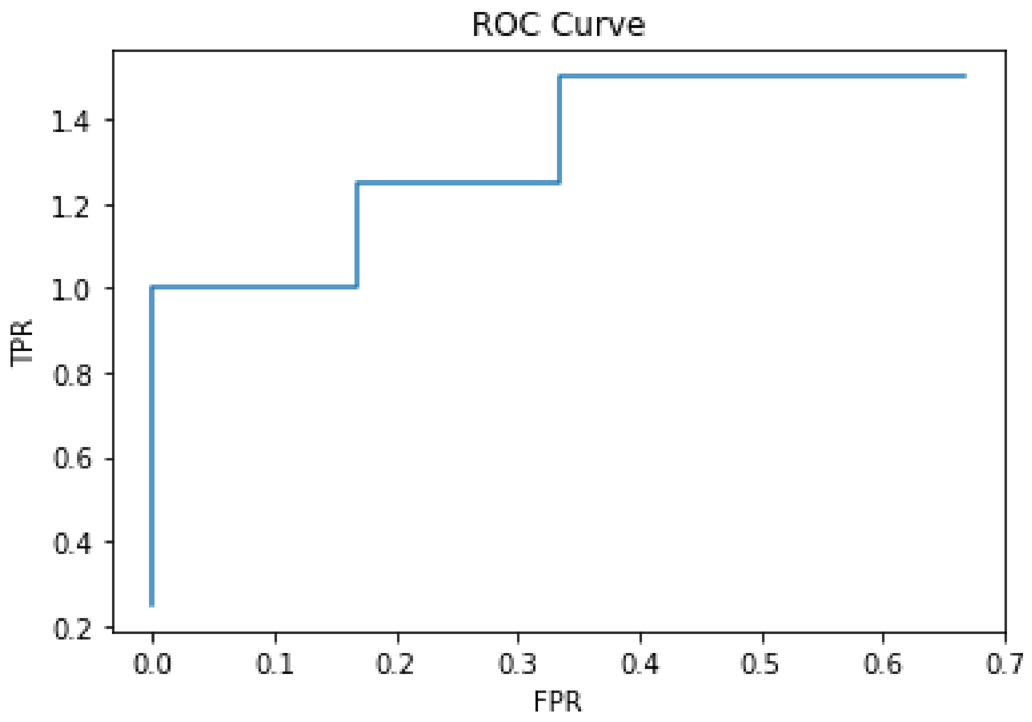
```
# calculate ROC curve
import matplotlib.pyplot as plt
import numpy as np

# actual values (sick)
actual= [1,1,1,1,0,1,0,1,0,0]

# predicted probability score values
scores = [1.0,.95,.9,.89,.77,.69,54,.34,.24,.12];

print("Calculate ROC")

# false positive rate
fpr = []
# true positive rate
tpr = []

TP = 0
FP = 0

# calculate total true
P = sum(actual)

# calculate total false
N = len(actual) - P
```

```
for i in range(len(scores)):

   # cummulate tp
   if actual[i] == 1:
      TP = TP + 1

   # commulate fp
   if actual[i] == 0:
      FP = FP + 1

   # append fpr,tpr
   fpr.append(FP/float(N))
   tpr.append(TP/float(P))

# print results
print("TPR: ", tpr)
print("FPR: ", fpr)

# plot results
plt.plot(fpr, tpr)
plt.title('ROC Curve')
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.show()
```

Here is the output:

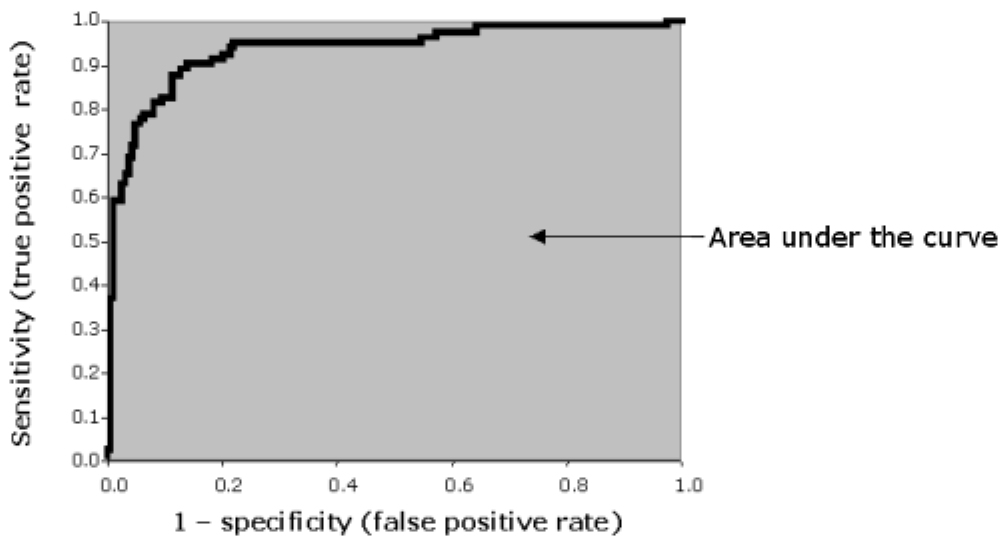
```
Calculate ROC

TPR:  [0.25, 0.5, 0.75, 1.0, 1.0, 1.25, 1.25, 1.5, 1.5, 1.5]

FPR:  [0.0, 0.0, 0.0, 0.0, 0.16666666666666666, 0.16666666666666666,
0.3333333333333333, 0.3333333333333333, 0.5, 0.6666666666666666]
```

Here is the plot

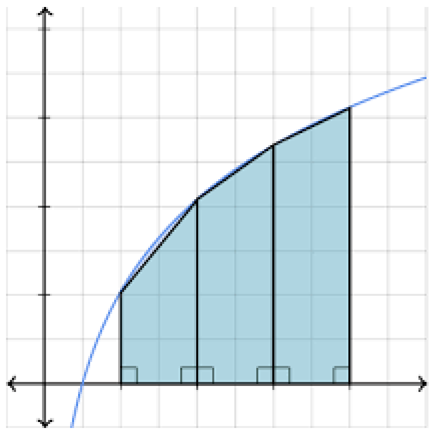


AREA UNDER CURVE (AUC)

For binary classifiers ROC give a probability or score that reflects the degree to which an instance belongs to one class rather than another. To compare different classifiers, it can be useful to summarize the performance of each classifier into a single measure. The area under the ROC curve, which is abbreviated to AUC, is equivalent to the probability that a randomly chosen positive instance is ranked higher than a randomly chosen negative instance.



We can use the trapezoid algorithm the calculate the area of the curve,. The trapezoid algorithm sums the rectangles under the curve.



area under roc curve (auc)= ∑(Xi - Xi-1) * (Yi + Yi-1) / 2

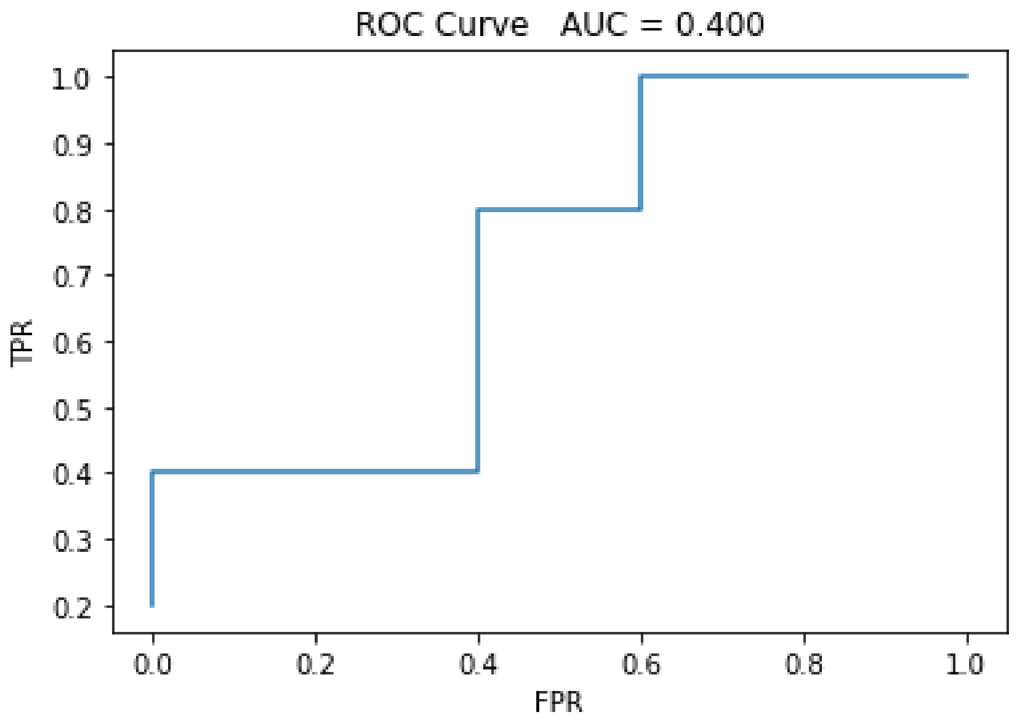
auc = sum (i to n)(((i - (i-1)) * (tpr[i] + tpr[i-1]))/2.0)

CONFUSION MATRIX AND ROC HOMEWORK QUESTION 3

Calculate the area under the ROC curve (AUC) for the previous ROC CURVE.

```
auc = 0
for i in range(1,len(tpr)):
   v = (i - (i-1)) * abs(tpr[i] - tpr[i-1])/2.0
   auc += v
```

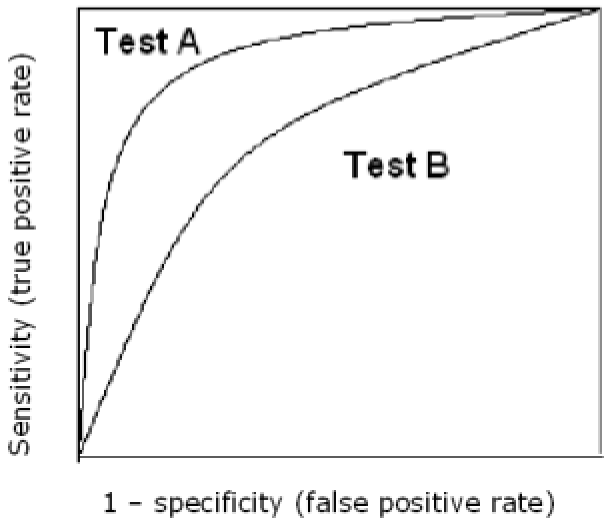
Display the AUC in the title of the plot . You should get something like this:



Put your answer in a py file called confusion_roc_homework3.py

AUC for different tests

Each test will give you different ROC curve with different AUC values



Higher AUC means better classification results

AUROC	Category
0.9-1.0	Very good
0.8-0.9	Good
0.7-0.8	Fair
0.6-0.7	Poor
0.5-0.6	Fail

Calculating ROC and AUC using SkLearn

Sklearn has the **roc_curve function located in metrics module for calculating tpr, fpr and** Decreasing thresholds on the decision function used to compute fpr and tpr. Sklearn also has the **roc_auc_score function located in the metrics module for calculating the area under the curve (AUC)**

We use the actual and score to calculate the predicted values

actual = [1,1,1,1,0,1,0,1,0,0]

scores = [1.0,.95,.9,.89,.77,.69,54,.34,.24,.12]

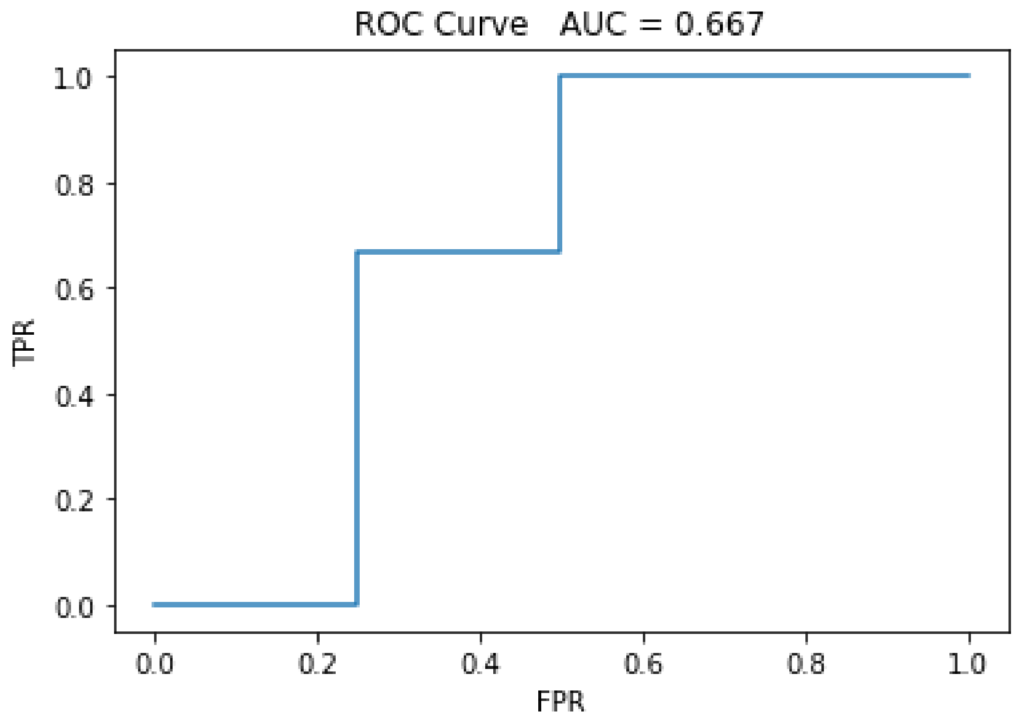
Here is the Program:

```
# plotting ROC curve using sklearn
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
import matplotlib.pyplot as plt

print("ROC Curve Program Output")

# sick values
actual = [1,1,1,1,0,1,0,1,0,0]

# scores
scores = [1.0,.95,.9,.89,.77,.69,54,.34,.24,.12];

# calculate fpr, tpr and thresholds
fpr, tpr, thresholds = roc_curve(actual,  scores)

# calculate area under curve (auc)
auc = roc_auc_score(actual, scores)

# print results
print("fpr:",fpr)
print("tpr:",tpr)
print("thresholds:",thresholds)
print("auc:",auc)

# plot ROC and print AUC in title
plt.plot(fpr,tpr)
plt.title('ROC Curve   AUC = ' + "%.3f" % (auc))
plt.ylabel('TPR')
plt.xlabel('FPR')
plt.show()
```

Here is the program output

```
ROC Curve Program Output
fpr: [0.   0.25 0.25 0.5  0.5  1.  ]
tpr: [0.         0.         0.66666667 0.66666667 1.         1.        ]
thresholds: [55.   54.    0.89  0.77  0.34  0.12]
```

Here is the plot:



ConfusionMatrixAndROC Homework Question 4

Calculate and plot a ROC curve using the following actual and predicted values from homework Question 1.

```
# actual values
actual = [1,0,0,1,0,0,1,0,0,1]

# predicted values
predicted = [1,0,0,1,0,0,0,1,0,0]
```

Step 0: put the above actual and predicted values in your program

```
# actual values
actual = [1,0,0,1,0,0,1,0,0,1]

# predicted values
predicted = [1,0,0,1,0,0,0,1,0,0]
```

Step1 estimate probabilities for predicted values

We do not have the probabilities for each value, but you have 4 options to calculate them:

(1) calculate the probabilities from the confusion matrix
(2) estimate probabilities from calculated recall and precision values
(3) hard code some values
(4) make some random values like this:

```
test_probs = np.random.rand(10)
```

Note: Each output value will have its own accuracy probability between 0 and .1. Usually the classifier provides these probabilities for you. We have only the actual and predicted output values.

Step 2: create calculate_tpr_fpr function

Make a function that will calculate tpr and fpr from the input actual values and the predicted values and return them.

Count the TP's TN's and FN's using the following chart

	Actual	**Predicted**	**Description**
TP	**P**	**P**	**P** expected got **P**
TN	**N**	**N**	**N** expected got **N**
FP	**N**	**P**	**N** expected but got **P** instead
FN	**P**	**N**	**P** expected but got **N** instead

Use these formulas for calculating TPR (true prediction rate) and FPR (false prediction rate)

$$TPR = \frac{TP}{(TP + FN)}$$

$$FPR = \frac{FP}{(TN + FP)}$$

Step 3: calculate TPR and FPR using the calculate_tpr_fpr function

Using the calculate_tpr_fpr function calculate tpr and fpr for the given actual and predicted sample values.

Print out the calculated values for tpr and fpr.
You should get:

```
tpr:  0.5
fpr:  0.16666666666666666
```

Step 4: generate test thresholds

Make a table of about 100 probability test thresholds between 0 and 1 using linspace

```
thresholds = np.linspace(0, 1, num=100)
```

Step 5: make empty lists tprs and fpr

```
# make a list to store tprs
tprs = []


# make a list to store fprs
fprs = []
```

Step 6: calculate lists of tprs and fprs

```
In a outer Loop go through each threshold

    make a empty list called test_preds (test predictions)

    In a Inner Loop go through each actual value

        if the probably for each actual value is greater than the
        threshold then append 1 to test_preds list

        if the probability for each actual value is  less or equal than the
        threshold then append 0 to test_preds list


    At the end of inner loop iterations calculate tpr and fpr and append to
    the tprs and fprs list
```

Step 7: plot ROC curve

After all iterations have completed, plot the RC curve stored in the tprs and fprs lists.

Setp 8: calculate AUC

From the tpr and fpr calculate the area under curve (auc)

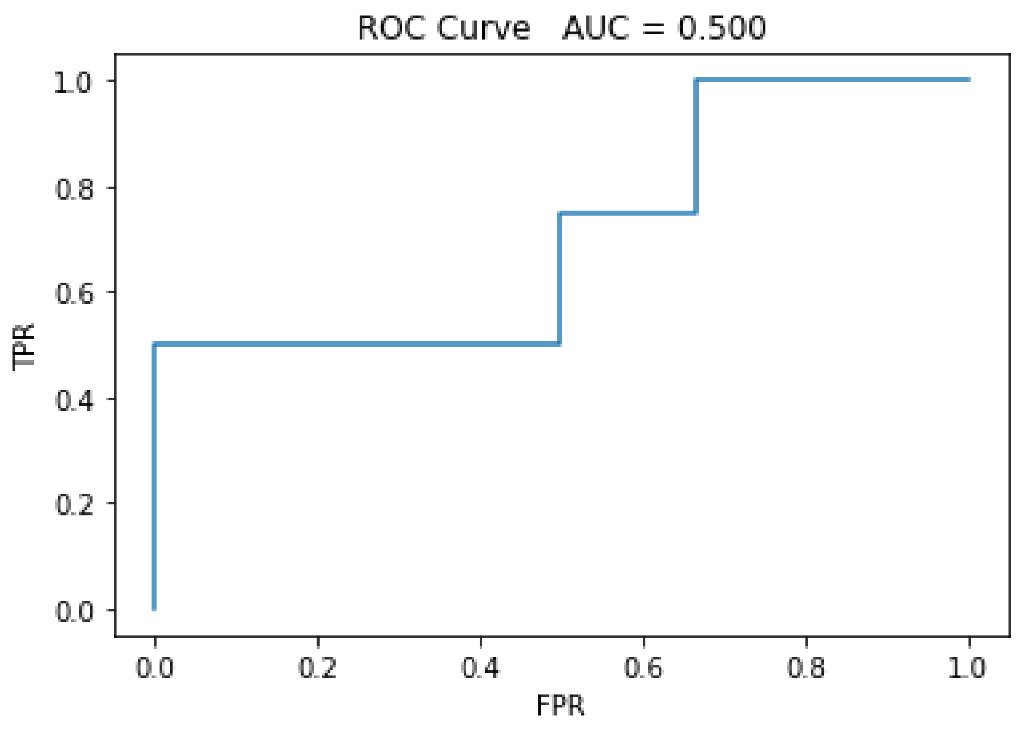
You can use this formula:

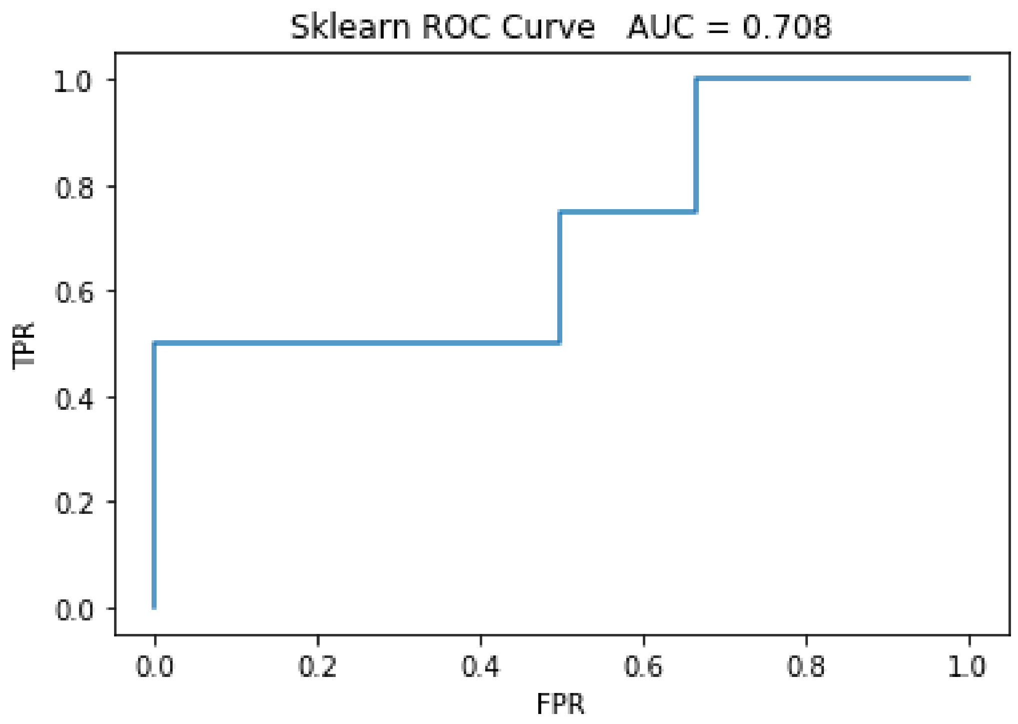
```
auc = 0
for i in range(1,len(tpr)):
    v = (i - (i-1)) * abs(tpr[i] - tpr[i-1])/2.0
   auc += v
```

Step 9:

Use **sklearn** to plot the ROC curve using the same actual values and using your estimated probabilities. Compare your finding to the sklearn results.

You should get something like this:





Put your answer in a py file called confusion_roc_homework4.py

END