COMP4220: Machine Learning, Spring 2021, Assignment 4

Due: Wednsday, March 24, 11pm

Please submit one pdf file for all questions.

You can type your answer for the first two questions in the below cell of each question using "Markdown" option!

**When turning in assignments after the due date, please clearly specify the number of late hours used.

1. What are precision and recall?

Precision descrives the ratio between true positives and the total positives. It can be calculated by deviding the number of true positives by total number of positives (true positives plus false positives). This differs from recall which aims to measure the model's correctness in identifying true positives. Recall can be calculated by dividing the number of true positives by the sum of the true positives and false negatives.

2. Explain the use of ROC curves and the AUC of an ROC Curve.

The Receiver Operator Characteristic (ROC) curve is used an evaluation metric for binary classification problems. It is a probability curve that shows the ability of the binary classifications system as the system's discrimination threshold changes. The area under the curve (AUC)

summarizes the ROC curve and measures the ability of the binary classification system to discern between classes.

3. Programming assignment(Logistic Regression)

This example uses two feature of the Social_Network_Ads dataset that is uploaded along with this Jupyter notebook.

In [1]:
```
# Importing the libraries
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
```

In [2]:
```
# Importing the dataset

dataset = pd.read_csv('Social_Network_Ads.csv')
X = dataset.iloc[:, [2, 3]].values
y = dataset.iloc[:, 4].values
```

P3.1: Split the dataset into the Training set(%75) and Test set(%25) using "train_test_split" from sklearn.model_selection.

In [25]:
```
from sklearn.model_selection import train_test_split

X_training_set, X_testing_set, y_training_set, y_testing_set = train_te
st_split(X, y, test_size = 0.25, random_state = 0)
```

p3.2: Apply Feature Scaling method for X_train and X_test with "StandardScaler" from "sklearn.preprocessing"

Hint: use StandardScaler.fit_transform(X_train) for "X_train" and use StandardScaler.transform(X_test) for "X_test"

In [26]:
```
from sklearn.preprocessing import StandardScaler

standard_scaler = StandardScaler()
X_training_set = standard_scaler.fit_transform(X_training_set)
X_testing_set = standard_scaler.transform(X_testing_set)
```

p3.3: Train the logistic Regression model on the training set

In [27]:
```
from sklearn.linear_model import LogisticRegression

LogReg = LogisticRegression(random_state = 0)
LogReg.fit(X_training_set, y_training_set)
```

Out[27]:
```
LogisticRegression(random_state=0)
```

p3.4: Predict and show the Test set results

In [28]:
```
y_pred = LogReg.predict(X_testing_set)
```

p3.5: Make and show the Confusion Matrix and show the result

In [29]:
```
from sklearn.metrics import confusion_matrix

ConfusionMatrix = confusion_matrix(y_testing_set, y_pred)
print(ConfusionMatrix)
```

```
[[65  3]
 [ 8 24]]
```

How do you decide which threshold to use?

First, use the cross_val_predict()function to get the scores of all instances in the training set, but this time specify that you want to return decision scores instead of predictions:

In [30]:
```
from sklearn.model_selection import cross_val_predict

y_scores = cross_val_predict(LogReg, X_training_set, y_training_set, cv
=3,method="decision_function")
print(y_scores)
```

```
[-0.48692607 -1.04986532 -1.51516084 -0.67015423  0.79967339  3.1185230
2
 -1.49989183 -1.23670604 -1.95072403 -1.62164108 -2.80135744 -2.1685056
2
 -0.06181078  0.41955945  2.52665714  1.2601478   2.05975022 -2.0523831
7
  1.03714225  0.47018759 -5.01453253  4.59236605  0.05431167  0.3633045
 -0.56809224 -1.94067897  0.43482846  2.78903723  0.07962575 -0.9944160
7
 -5.01453253  3.07352168 -5.34883943  1.86205875  1.39153927 -3.4651501
4
 -0.1329319  -2.70572795 -0.30490645 -0.59822742 -2.12310143 -4.6251792
3
 -2.43209426  0.11538773 -1.61159602 -0.09797561 -0.54800212  3.7461508
9
  0.96120004 -1.53565381 -1.48984677 -1.31787221 -1.0241484  -1.2367060
4
  4.74425048 -4.62517923  3.23022722 -0.26432336 -3.3691178   1.9231348
```

```
 -1.89005082 -4.76179465 -3.84486123  0.07480464 -2.16328167 -2.8981954
8
 -1.49989183 -1.84504948 -1.31827506 -3.40970089 -1.6369101  -1.2266609
8
 -2.44696043 -0.23860644  0.28695943  2.89029352 -3.21723336 -3.6122134
7
  3.7614199   0.63251994 -1.9153649  -1.00928223 -1.81451145 -0.4860845
1
 -0.10239387  2.13920322 -3.84968234 -2.50803648  2.44770476  3.6524067
4
 -6.02267718  0.49601234  0.6091968  -0.82442804 -4.30573849 -0.3708036
1
  0.45491858  2.85574008  0.83462967 -0.84244081 -3.46515014 -0.9023991
4
  6.4443794   1.14218538  0.575591    2.63428436  5.16440038 -2.7507293
  4.48085774 -0.62601921 -3.97183444 -1.32945651 -0.26432336 -2.8322983
2
 -1.28733418 -3.26786151 -4.74770755 -1.24920581 -5.11589309 -0.4027420
4
 -0.48111084 -1.28590338 -5.63275079 -1.12110035  4.01440874 -5.8460806
3
 -0.59241341 -1.32260095 -5.85602796 -4.52443038  1.37917368 -2.2175915
2
  2.34567749 -0.65774311 -0.56754507 -1.10738921 -7.17270454 -3.0174104
 -0.96745451 -4.79435245 -0.60115087 -1.10927111  0.39895874 -1.6341942
5
  2.34944129 -0.95441541 -0.82254614 -4.75079932 -2.78915956 -2.3208286
6
 -1.03278421 -1.02592865 -0.18067474 -0.66083488 -0.0158717  -0.1160170
7
 -0.65586121 -7.00601961 -0.87416471 -1.92091922 -1.18763991  0.7877109
7
 -0.65276944  0.82938221  4.36269961 -3.96590144 -1.03587598 -1.2392584
8
 -6.28268763 -2.56090873  1.22243608  1.23050151  2.20574279 -0.3989782
4
  1.9129007  -2.36747356  3.27745016 -4.38758745  0.79080274 -4.6892334
2
 -0.44253137 -3.89438821  2.07543835 -6.65772873 -2.93594983  2.6143896
9
```

```
 -5.64767179 -1.37112775 -2.34260523  2.83695381  2.21697634 -2.4259476
9
  1.01240589 -4.44417968  0.82215966 -3.07091087 -3.35266217 -0.9331168
3
 -5.07610375 -7.25604701  1.43576591  0.75703849 -1.59252302 -0.8225461
4
 -6.0227129  -0.35421524 -2.45578969  0.71706725 -2.8009888  -1.0825208
8
 -2.99406058 -2.08077257 -0.57724376 -0.43111903  2.76618482  0.3037489
4
  3.96766608 -2.61356812 -1.68130854 -4.35040358 -2.74381046 -0.2593138
3
 -2.17563041  3.57896716  2.54311273 -0.28649186 -2.41511542 -0.9444124
7
 -0.50391612  1.97290458 -0.94750165 -4.48117646 -0.89005641 -0.6126282
5
  1.57291002 -2.29563819  2.06776243 -0.83825898  1.96213948 -0.3085526
1
 -3.68480704 -0.80031584  0.32218997 -1.35208296 -2.82843374 -2.6186854
  2.94566594 -3.6381269   1.08317488 -3.10277271 -2.41767406 -2.2110149
1
 -0.99312071 -0.25931383  1.08317488 -0.89058695 -5.20677688 -3.9155550
5
 -2.13512863 -0.57468512 -0.0690676   0.17041741 -1.40079121 -0.3736737
8
  2.28827587 -4.35861004 -3.43711556 -1.23516437 -1.60489172 -4.8344908
8
  3.17950313 -0.83825898  0.89037001 -5.18780531 -0.62083472 -0.6449235
7
 -3.81813856 -1.09671556  1.42113746 -1.45823645  0.21400837 -3.7714584
2
 -4.08171243 -0.41161692 -2.58383144 -2.78228414 -2.13256999  1.2416563
4
  1.71700664  1.54573198  3.744594   -1.47720802 -0.57724376 -2.2463994
 -1.48797313 -0.20186858 -3.10277271 -1.65924779 -3.14636367 -1.5782442
3]
```

Logistic Regression Classification metrics

p3.6: Compute the accuracy, precision, recall and f1_score and show all results

In [31]:
```
from sklearn.metrics import precision_score
from sklearn.metrics import recall_score
from sklearn.metrics import f1_score
from sklearn.metrics import accuracy_score

AccuracyScore = accuracy_score(y_testing_set, y_pred)
PrecisionScore = precision_score(y_testing_set, y_pred)
RecallScore = recall_score(y_testing_set, y_pred)
F1Score = f1_score(y_testing_set, y_pred)

print ("Accuracy Score is", AccuracyScore)
print ("Precision Score is", PrecisionScore)
print ("Recall Score is", RecallScore)
print ("F1 Score is", F1Score)
```

```
Accuracy Score is 0.89
Precision Score is 0.8888888888888888
Recall Score is 0.75
F1 Score is 0.8135593220338982
```

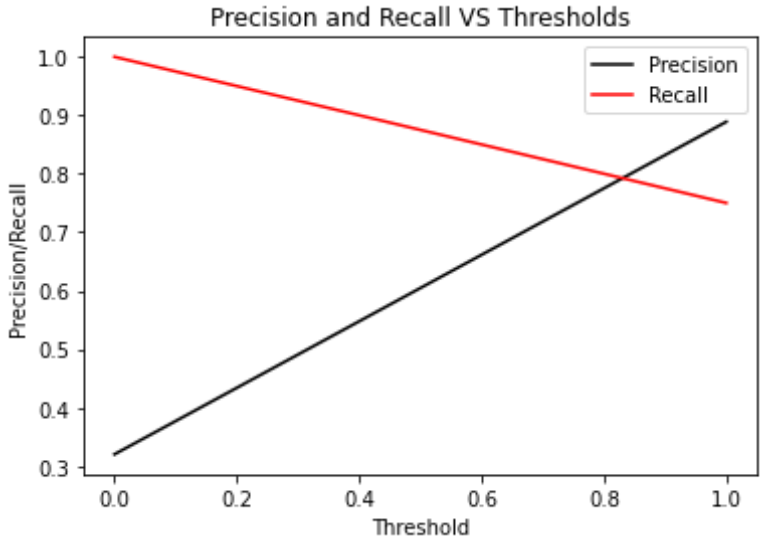
p3.7: use the precision_recall_curve() function to compute precision and recall for all possible thresholds

In [40]:
```
from sklearn.metrics import precision_recall_curve

precision, recall, thresholds = precision_recall_curve(y_testing_set, y
_pred)
```

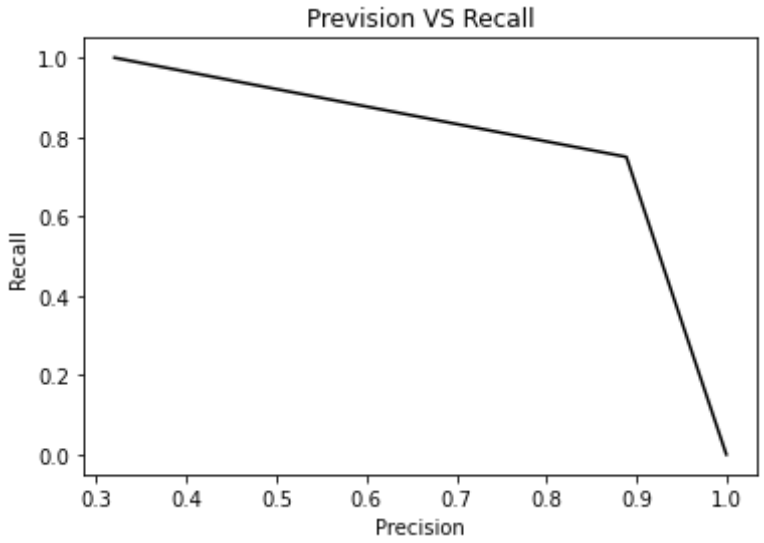
p3.8:Use Matplotlib to plot precision and recall as functions of the threshold value

In [47]:
```
plt.plot(thresholds, precision[:-1], color = 'black', label = 'Precisio
n')
plt.plot(thresholds, recall[:-1], color = 'red', label = 'Recall')
plt.xlabel('Threshold')
plt.ylabel('Precision/Recall')
plt.legend(loc = 'upper right')
plt.title("Precision and Recall VS Thresholds")
plt.show()
```



p3.9: Plot precision_vs_recall plot

In [46]:
```
plt.plot(precision, recall, color = 'black')
plt.xlabel('Precision')
plt.ylabel('Recall')
plt.title("Prevision VS Recall")
plt.show()
```

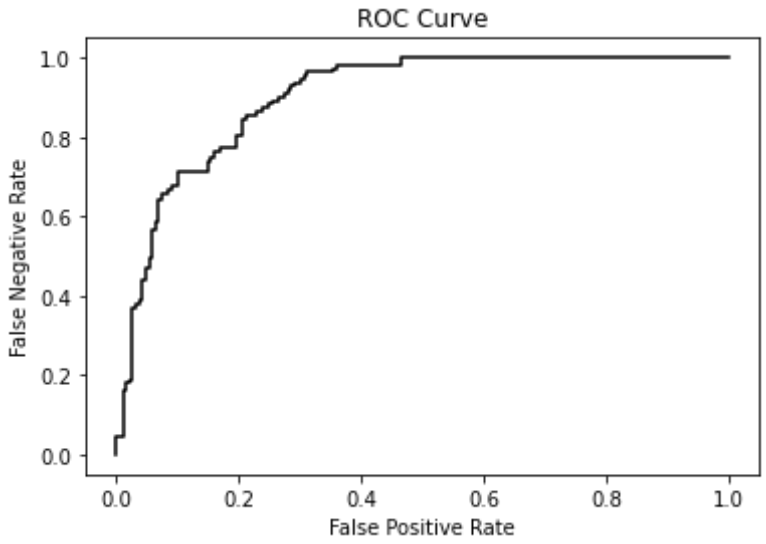


p3.10:plot the ROC curve

you should first use the roc_curve() function to compute the TPR and FPR for various threshold values:

In [45]:
```
from sklearn import metrics

FPR, TPR, threshold = metrics.roc_curve(y_training_set, y_scores)
plt.plot(FPR, TPR, color = 'black')
plt.xlabel("False Positive Rate")
plt.ylabel("False Negative Rate")
plt.title("ROC Curve")
plt.show()
```



p3.11: Compute AUC score and show the result

In [54]:
```
AUCScore = metrics.auc(FPR, TPR)
print (AUCScore)
```

```
0.9016159016159017
```

In [ ]:
```

```