Lab 2 part 2: Practical application

Tanvi Mukesh PARMAL

Arnaud MATHEY

Imports:

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import statsmodels.api as sm
import statsmodels.formula.api as smf
import sklearn
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix,classification_report
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.discriminant_analysis import QuadraticDiscriminantAnaly
sis
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import roc_curve, auc
```

2.1 The dataset description

1) Read the dataset

In [2]:
```
auto=pd.read_csv('auto.txt', sep=";")
```

In [3]:
```
auto.describe()
```

Out[3]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	
count	300.000000	300.000000	300.000000	300.000000	300.000000	300.000000	300.0
mean	23.699333	5.433333	192.338333	104.050000	2945.846667	15.460000	75.9
std	7.963814	1.717183	105.453661	38.959127	849.694403	2.729389	3.6
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.0
25%	17.000000	4.000000	98.000000	75.000000	2212.500000	13.500000	73.0
50%	23.600000	4.000000	140.000000	91.500000	2730.000000	15.400000	76.0
75%	29.925000	8.000000	275.750000	129.000000	3526.250000	17.000000	79.0
max	46.600000	8.000000	455.000000	230.000000	4997.000000	23.500000	82.0

2) Create a binary variable, denoted mpg01, that is worth 1 if mpg contains a value above its median, and a 0 if mpg contains a value below its median. You can use the method median() and the function get_dummies(). Then, create a single data set containing both mpg01 and the other Auto variables

In [4]:
```
mpg_median = auto [ "mpg" ] . median ( )
mpg01= pd. get_dummies ( auto [ "mpg"]>mpg_median , drop_first=True
)
auto['mpg01 ']=mpg01
```

In [5]:
```
auto.describe()
```

Out[5]:

	mpg	cylinders	displacement	horsepower	weight	acceleration	
count	300.000000	300.000000	300.000000	300.000000	300.000000	300.000000	300.0
mean	23.699333	5.433333	192.338333	104.050000	2945.846667	15.460000	75.9
std	7.963814	1.717183	105.453661	38.959127	849.694403	2.729389	3.6
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.0
25%	17.000000	4.000000	98.000000	75.000000	2212.500000	13.500000	73.0
50%	23.600000	4.000000	140.000000	91.500000	2730.000000	15.400000	76.0
75%	29.925000	8.000000	275.750000	129.000000	3526.250000	17.000000	79.0
max	46.600000	8.000000	455.000000	230.000000	4997.000000	23.500000	82.0

In [6]:
```
mpg_median
```

Out[6]: 23.6

The median of mgp is 23.6

In [7]:
```
X = auto.iloc[:,1:8]
```

In [8]:
```
X
```

Out[8]:

	cylinders	displacement	horsepower	weight	acceleration	year	origin
0	8	307.0	130	3504	12.0	70	1
1	8	350.0	165	3693	11.5	70	1
2	8	318.0	150	3436	11.0	70	1
3	8	304.0	150	3433	12.0	70	1
4	8	302.0	140	3449	10.5	70	1
...	...	...	...	...	...	...	...
295	6	262.0	85	3015	17.0	82	1
296	4	144.0	96	2665	13.9	82	3
297	4	135.0	84	2370	13.0	82	1
298	4	151.0	90	2950	17.3	82	1
299	4	135.0	84	2295	11.6	82	1

300 rows × 7 columns

In [9]:
```
y = auto.iloc[:,-1]
```

In [10]:
```
y
```

Out[10]:
```
0      0
1      0
2      0
3      0
4      0
      ..
295    1
296    1
297    1
298    1
299    1
Name: mpg01 , Length: 300, dtype: uint8
```

3) You will fit the classifiers studied in the lecture using the train data and evaluate the quality of your models using the test data. That is why the whole data set was split into two sub- sets. To this end, you will find in moodle 4 files x_train.csv, x_test.csv, y_train.csv and y_test.csv. Import the four datasets.

In [11]:
```
#X_train, X_test, y_train, y_test = train_test_split( X, y, test_size=0.33, random_state=42)

X_train = pd.read_csv('x_train.csv',index_col=0)
X_test = pd.read_csv('x_test.csv',index_col=0)
y_train = pd.read_csv('y_train.csv',index_col=0)
y_test = pd.read_csv('y_test.csv',index_col=0)
```

In [12]:
```
X_train.shape
```

Out[12]:
```
(225, 4)
```

In [13]:
```
X_test.shape
```

Out[13]:
```
(75, 4)
```

In [14]:
```
y_train.shape
```

Out[14]:
```
(225, 1)
```

In [15]:
```
y_test.shape
```

Out[15]:
```
(75, 1)
```

As you can see there is 225 in the training set and 75 in the testing set

2.2 Logistic regression

1) Perform logistic regression with mpg01 as the response and cylinders, weight, displacement and horsepower as predictors. You will need to set the argument as follows fa- mily=sm.families.Binomial() to tell Python to run logistic regression. Previously, do not forget to merge the x_train and the y_train dataframes, to this end you can use the function concat() of the Pandas library as follows :

In [16]:
```
family=sm.families.Binomial()
```

In [17]:
```
auto_train=pd.concat([X_train,y_train],axis=1)
```

In [18]:
```
auto_train.shape
```

Out[18]:
```
(225, 5)
```

In [19]:
```
mpg01 = auto_train.iloc[:,-1]
mpg01
```

Out[19]:
```
259    1
37     1
97     1
191    1
135    0
      ..
251    1
192    0
117    0
47     0
172    0
Name: mpg01, Length: 225, dtype: int64
```

In [20]:
```
formula = 'mpg01 ~ cylinders + weight + displacement + horsepower'
model = smf . glm( formula = formula , data=auto_train , family=sm.f
amilies.Binomial( ) )
logreg = model.fit()
```

In [21]:
```
print(logreg.summary())
```

```
                 Generalized Linear Model Regression Results
==================================================================
============
Dep. Variable:                 mpg01   No. Observations:
225
Model:                           GLM   Df Residuals:
220
Model Family:               Binomial   Df Model:
4
Link Function:                 logit   Scale:
1.0000
Method:                         IRLS   Log-Likelihood:
-56.878
Date:               Sun, 10 Oct 2021   Deviance:
113.76
Time:                       20:16:04   Pearson chi2:
315.
No. Iterations:                    7
Covariance Type:           nonrobust
==================================================================
==============
                  coef    std err          z      P>|z|      [0.0
25      0.975]
------------------------------------------------------------------
--------------
Intercept      11.2381      2.214      5.075      0.000       6.8
98      15.578
cylinders       0.0629      0.472      0.133      0.894      -0.8
62       0.988
weight         -0.0019      0.001     -1.978      0.048      -0.0
04   -1.75e-05
displacement   -0.0174      0.011     -1.520      0.129      -0.0
40       0.005
horsepower     -0.0355      0.017     -2.036      0.042      -0.0
70      -0.001
==================================================================
==============
```

Let's start with the cylinder coefficient as you can see it's positive so we can say that the cylinder does indeed have an impact on mgp01. his value is also a lot higher than over coefficient which seems to indicate that it has a lot more impact on mgp01.

The other coefficient are negative whitch mean that they are inversely proportional to mgp01. weight as almost no impact compared to the others. horsepower is around twice as more influent than displacement but still twice less influent than cylinders.

The intercept coefficient has an intercept positive so the probability og gessing mgp is higher than 0.5. In fact I calculated that there is around a 0.99998632599 probability of finding the mgp01 based on these coefficient whitch is extremely high we do have all the necessery coefficient for acurate prediction

2) The command print(logreg.fittedvalues) allows to print the estimated probability P(mpg01 = 1|X) (where X is the set of explanatory variables). That means, the probability of high mileage, higher than the median given cylinders, weight, displacement and horsepower fixed

In [22]:
```
print(logreg.fittedvalues)
```

```
259    0.629961
37     0.965645
97     0.982545
191    0.987334
135    0.658731
         ...
251    0.982233
192    0.044046
117    0.140836
47     0.000058
172    0.066813
Length: 225, dtype: float64
```

In [23]:
```
logmodel=LogisticRegression()
logmodel.fit(X_train,y_train.values.ravel())
```

Out[23]:
```
LogisticRegression()
```

In [24]:
```
y_predict=logmodel.predict(X_test)
yhat = logmodel.predict(X_train)
confusion_matrixs = confusion_matrix(yhat,y_train)
```

In [25]:
```
print("Accuracy",(logmodel.score(X_train,yhat)))
```

```
Accuracy 1.0
```

In [26]:
```
print(confusion_matrixs)
```

```
[[ 96   7]
 [ 13 109]]
```

In [27]:
```
target_names = ['class 0', 'class 1']
print(classification_report( yhat , y_train , digits=3, target_names
=target_names))
```

```
              precision    recall  f1-score   support

     class 0      0.881     0.932     0.906       103
     class 1      0.940     0.893     0.916       122

    accuracy                          0.911       225
   macro avg      0.910     0.913     0.911       225
weighted avg      0.913     0.911     0.911       225
```

As you can see the Accuracy is around 1 which is normal for training data, For the confusion matrice the result seems pretty good the result of 96 real positive for only 13 false positive is decent. 7 for 109 is really good too. The classification report also show us similar data with better result for the identification of negative data

In [28]:
```
print("Accuracy",(logmodel.score(X_test,y_test)))
```

```
Accuracy 0.8533333333333334
```

In [29]:
```
y_test_predict = logmodel.predict(X_test)
confusion_matrixs = confusion_matrix(y_test_predict,y_test)
print(confusion_matrixs)
```

```
[[33  3]
 [ 8 31]]
```

In [30]:
```
target_names = ['class 0', 'class 1']
print(classification_report( y_test_predict,y_test , digits=3, targe
t_names=target_names))
```

```
              precision    recall  f1-score   support

     class 0      0.805     0.917     0.857        36
     class 1      0.912     0.795     0.849        39

    accuracy                          0.853        75
   macro avg      0.858     0.856     0.853        75
weighted avg      0.860     0.853     0.853        75
```

As you can see the Accuracy is around 85% which is good for test data, For the confusion matrice the result seems pretty good the result of 33 real positive for 8 false positive seems not amazing around 25% error on positive. 3 for 31 is pretty good however. The report is in line with the previous one.

2.3 K-Nearest Neighbors

1) Fit a KNN classifier for the following values of K ∈ {1,3,5,17,51,75,101} and calculate the test error. What value of K would you choose ? For this value of K calculate the performance indicators you calculated in the previous section.

In [31]:
```
test_scores = []
k_values = [3, 5, 17, 51, 75, 101]

for k in k_values:
    knn = KNeighborsClassifier(n_neighbors = k)
    knn.fit(X_train,  np.ravel(y_train))
    score = knn.score(X_test, y_test)
    test_scores.append(score)

best_k = test_scores.index(max(test_scores))
print("Best k={}".format(k_values[best_k]))
```

```
Best k=3
```

In [32]:
```
y_hat = knn.predict(X_test)
cm = confusion_matrix(y_test, y_hat)

print("CONFUSION MATRIX\n")
print(cm)

print("\nCLASSIFICATION REPORT")
print(classification_report(y_test, y_hat, digits=3))
```

```
CONFUSION MATRIX

[[33  8]
 [ 4 30]]

CLASSIFICATION REPORT
              precision    recall  f1-score   support

           0      0.892     0.805     0.846        41
           1      0.789     0.882     0.833        34

    accuracy                          0.840        75
   macro avg      0.841     0.844     0.840        75
weighted avg      0.845     0.840     0.840        75
```

ROC (Receiver operating characteristic) curve for KNN

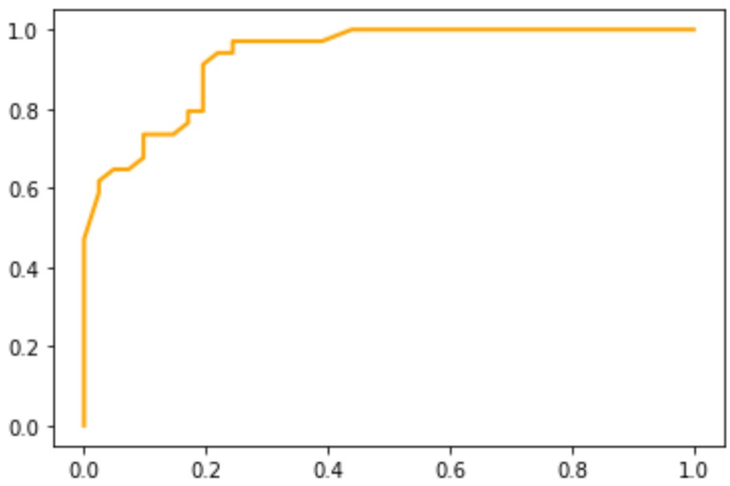
In [33]:
```
knn_scores = knn.predict_proba(X_test)[:,1]

fpr, tpr, thresholds = roc_curve(y_test, knn_scores)
auc_knn = auc(fpr, tpr)
print("AUC for KNN: {}".format(auc_knn))
```

```
AUC for KNN: 0.930416068866571
```

In [34]:
```
# Plot ROC curve for k-NN
plt.plot(fpr, tpr, c='orange', linewidth=2, label='k-NN')
```

Out[34]: [<matplotlib.lines.Line2D at 0x1d59f59da90>]



2.4 Discriminant Analysis

1) Interpret the prior probabilities as well as the group means

In [35]:
```
Lda = LinearDiscriminantAnalysis()
model_lda = Lda.fit( X_train , np.ravel(y_train ) )
yhat_lda = model_lda.predict(X_test)
print(model_lda.priors_)
```

```
[0.48444444 0.51555556]
```

In [36]:
```
print ( model_lda.means_ )
```

```
[[   6.75229358  272.16513761  129.33027523 3586.14678899]
 [   4.12931034  111.19396552   77.8362069  2294.68965517]]
```

The prior probabilities mean that the values have 48% chance to being positive of 51.5% of being negative according to the model.

In [37]:
```
confusion_matrixs = confusion_matrix(yhat_lda,y_test)
print(confusion_matrixs)
```

```
[[33  3]
 [ 8 31]]
```

In [38]:
```
print(model_lda.score(X_test,y_test))
```

```
0.8533333333333334
```

In [39]:
```
Qda = QuadraticDiscriminantAnalysis()
model_qda = Qda.fit( X_train , np.ravel(y_train ) )
yhat_qda = model_qda.predict(X_test)
```

In [40]:
```
print(model_qda.priors_)
```

```
[0.48444444 0.51555556]
```

In [41]:
```
print ( model_qda.means_ )
```

```
[[   6.75229358  272.16513761  129.33027523 3586.14678899]
 [   4.12931034  111.19396552   77.8362069  2294.68965517]]
```

In [42]:
```
print(model_qda.score(X_test,y_test))
```

```
0.8666666666666667
```

In [43]:
```
confusion_matrixs = confusion_matrix(yhat_qda,y_test)
print(confusion_matrixs)
```

```
[[34  3]
 [ 7 31]]
```

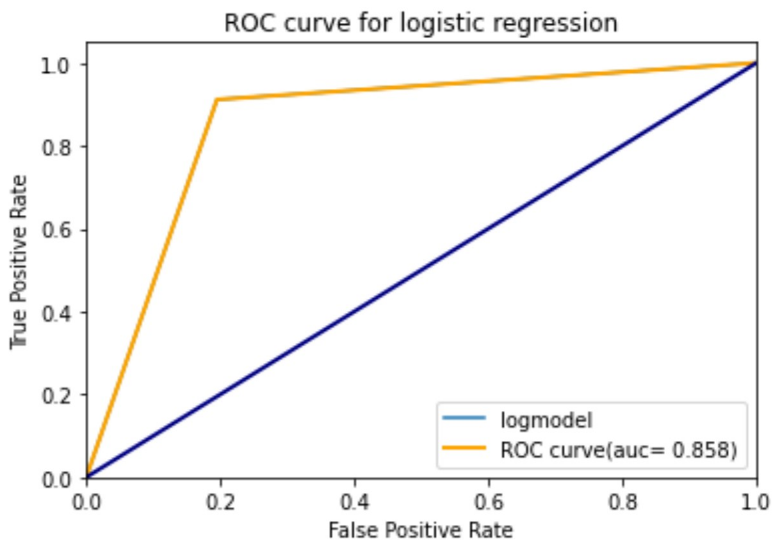
It seems that beside having similar coefficient qda is a little better with 1 less error on the test data. So the sensitivity is better on qda as well as the accuracy.

2.5 ROC (Receiver operating characteristic) curve

1) Interpret the outputs of the roc_curve() function, namely fpr and tpr.

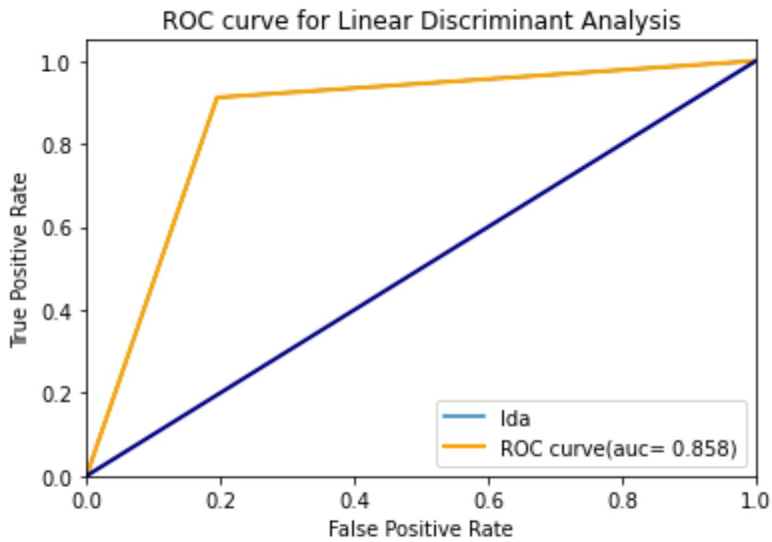
In [44]:
```
fpr,tpr,thresholds=roc_curve(y_test,logmodel.predict(X_test))
aire=auc(fpr,tpr)
```

In [45]:
```
plt.figure()
plt.plot(fpr,tpr,label="logmodel")
plt.plot(fpr,tpr,color='orange',lw=2,label='ROC curve(auc= %0.3f)' % aire)
plt.plot([0,1],[0,1],color='navy',lw=2)
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for logistic regression')
plt.legend(loc="lower right")
plt.show()
```



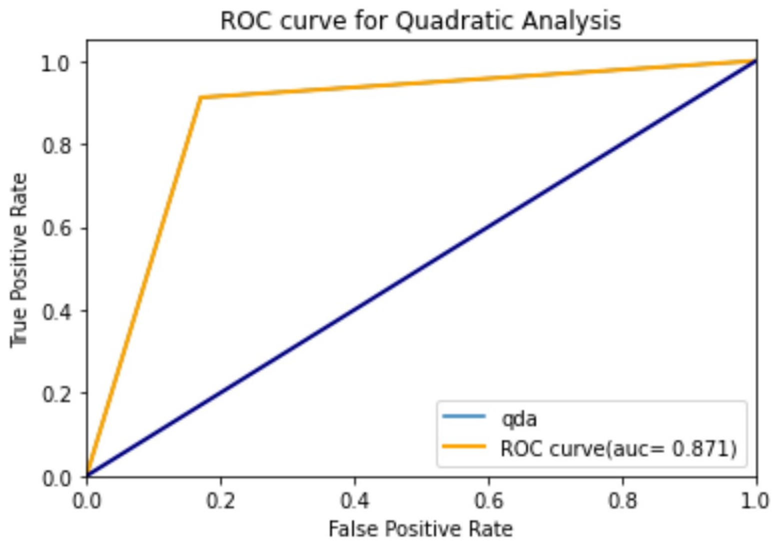
In [46]:
```
fpr,tpr,thresholds=roc_curve(y_test,model_lda.predict(X_test))
aire=auc(fpr,tpr)
```

In [47]:
```
plt.figure()
plt.plot(fpr,tpr,label="lda")
plt.plot(fpr,tpr,color='orange',lw=2,label='ROC curve(auc= %0.3f)' %
aire)
plt.plot([0,1],[0,1],color='navy',lw=2)
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for Linear Discriminant Analysis')
plt.legend(loc="lower right")
plt.show()
```



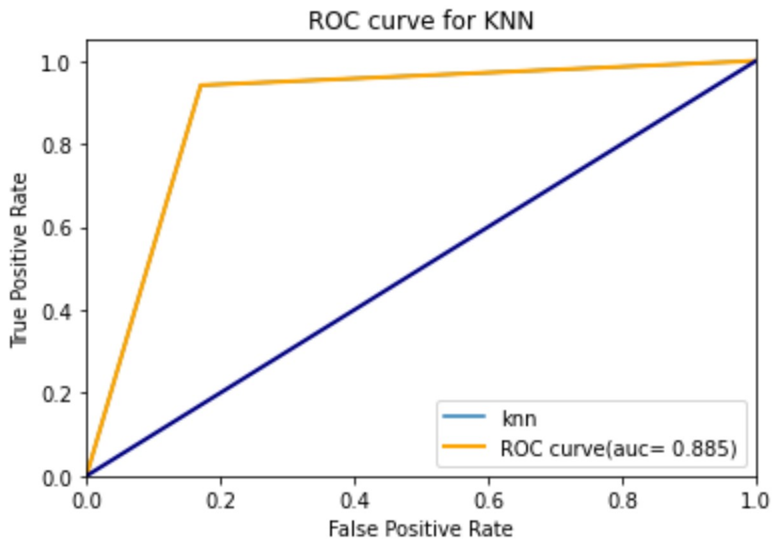
In [48]:
```
fpr,tpr,thresholds=roc_curve(y_test,model_qda.predict(X_test))
aire=auc(fpr,tpr)
```

In [49]:
```
plt.figure()
plt.plot(fpr,tpr,label="qda")
plt.plot(fpr,tpr,color='orange',lw=2,label='ROC curve(auc= %0.3f)' %
aire)
plt.plot([0,1],[0,1],color='navy',lw=2)
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for Quadratic Analysis')
plt.legend(loc="lower right")
plt.show()
```



In [50]:
```
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train,  np.ravel(y_train))
fpr,tpr,thresholds=roc_curve(y_test,knn.predict(X_test))
aire=auc(fpr,tpr)
```

In [51]:
```
plt.figure()
plt.plot(fpr,tpr,label="knn")
plt.plot(fpr,tpr,color='orange',lw=2,label='ROC curve(auc= %0.3f)' %
aire)
plt.plot([0,1],[0,1],color='navy',lw=2)
plt.xlim([0.0,1.0])
plt.ylim([0.0,1.05])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC curve for KNN')
plt.legend(loc="lower right")
plt.show()
```



As you can see the highest value of auc is 0.871 for the quadratic analysis. KNN was below everyone but not by far.