IIS – Logistic Regression Report

Bhavesh Sood(2019355)

First I'll show the snipped of the code and result of the best Accuracy percentage I could get (74.26%)

In [77]:
```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

In [78]:
```
df=pd.read_excel('./Data.xlsx',sheet_name='Sheet1')

from sklearn.metrics import classification_report
from sklearn.model_selection import train_test_split
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression
from sklearn import preprocessing
from sklearn.metrics import accuracy_score

model=LogisticRegression(solver='lbfgs',max_iter=100000)
```

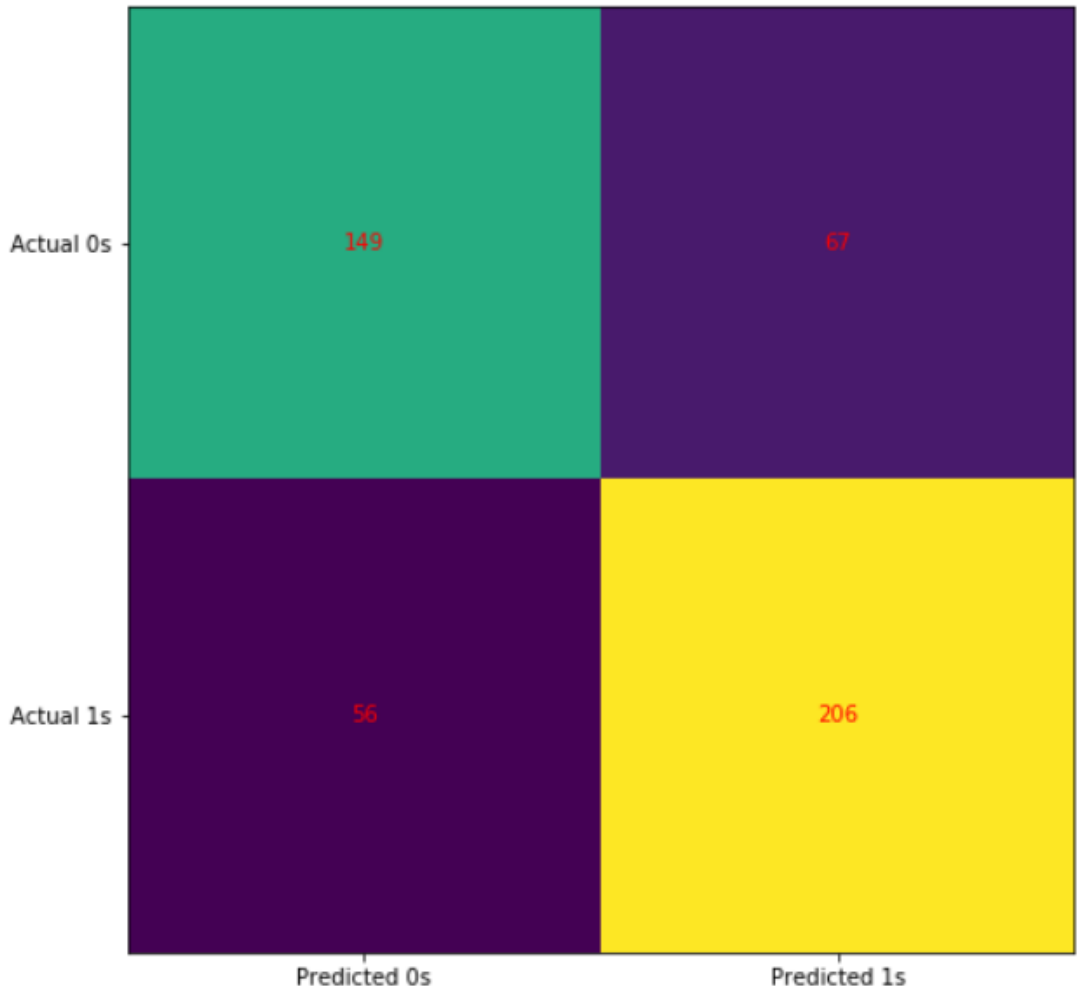
In [79]:
```
X=df[['conscientiousness','agreeableness','extraversion','nueroticism','openess_to_experience','12percentage','CollegeTier','collegeGPA','CollegeCityTier','English','Logical','Quant','ComputerProgramming','ElectronicsAndSemicon','Domain','ElectricalEngg','CivilEngg','MechanicalEngg']].values

X=preprocessing.scale(X)
y=df["High-Salary"].values
```

In [92]:
```
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=11,train_size=0.8806)
model.fit(X_train,y_train)
print("Model Score- " ,model.score(X_test,y_test))
y_predicted=model.predict(X_test)
print("Accuracy in (%) =",accuracy_score(y_test,y_predicted)*100,"%")
print()
print("Classification Report")
print()
labels=["0 - Low Salary","1- High Salary"]
print(classification_report(y_test, y_predicted,target_names=labels))
```

```
Model Score-  0.7426778242677824
Accuracy in (%) = 74.26778242677824 %

Classification Report

               precision    recall  f1-score   support

0 - Low Salary       0.73      0.69      0.71       216
1- High Salary       0.75      0.79      0.77       262

      accuracy                           0.74       478
     macro avg       0.74      0.74      0.74       478
  weighted avg       0.74      0.74      0.74       478
```

In [94]:
```
print("CONFUSION MATRIX\n")
cm =confusion_matrix(y_test, model.predict(X_test))

fig, ax = plt.subplots(figsize=(8, 8))
ax.imshow(cm)
ax.grid(False)
ax.xaxis.set(ticks=(0, 1), ticklabels=('Predicted 0s', 'Predicted 1s'))
ax.yaxis.set(ticks=(0, 1), ticklabels=('Actual 0s', 'Actual 1s'))
ax.set_ylim(1.5, -0.5)
for i in range(2):
    for j in range(2):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='red')
plt.show()
```

CONFUSION MATRIX



These are the basic steps I followed, the detailed code snippets are afterwards.

Steps:

1. First I figured out which columns from the data could be used to train the model, because there were plenty of columns which didn't contribute anything logical about whether a person will get High Salary or not. For example : I didn't use the ID column , the Gender coulumn.
2. Then I tried modifying the data to use them better.
3. Then I figured that every model has max_iters limit that refers to the number of iterations it uses for gradient descent to converge to the minimum cost. Earlier the limit was really less for such a large data, thus I made it to a higher number.
4. Then I tried shuffling the data(by using different random states) and found it to give different results.
5. Then I even tried different ratios for the train-test splits , and found that even they gave different results.T
6. I then figured out a way to find the best random state and training size ratio of the train_test_split function. I used a double for loop where for each random state from 1 to 20 I tried all ratios of training size from .40 to .90 .This was the main reason to achieve a high accuracy of 74.26%.(Code is below)

P.T.O

Data Modification:

I used the pandas library extensively to try modify data in many ways- to change the string inputs of fields like 10boards to different states 0- icse, 1- cbse , 2- state board (or some fields that contained 0 (meaning no info))

```
In [65]: for i in range(0,3998):
             c=df.loc[i,'10board']
             if(isinstance(c, int)):
                 df.loc[i,'10board']='2'
             elif(c[0]=='i'):
                 df.loc[i,'10board']='0'
             elif(c[0]=='c'):
                 df.loc[i,'10board']='1'
             else :
                 df.loc[i,'10board']='2'
         df['10board']

Out[65]: 0       2
         1       2
         2       2
         3       2
         4       2
                ..
         3993    2
         3994    2
         3995    2
         3996    2
         3997    2
         Name: 10board, Length: 3998, dtype: object
```

```
Name: 10board, Length: 3998, dtype: object

In [66]: for i in range(0,3998):
             c=df.loc[i,'12board']
             if(isinstance(c, int)):
                 df.loc[i,'12board']='2'
             elif(c[0]=='i'):
                 df.loc[i,'12board']='0'
             elif(c[0]=='c'):
                 df.loc[i,'12board']='1'
             else :
                 df.loc[i,'12board']='2'
         df['12board']

Out[66]: 0       2
         1       2
         2       2
         3       2
         4       2
                ..
         3993    2
         3994    2
         3995    2
         3996    2
         3997    2
         Name: 12board, Length: 3998, dtype: object
```

I even tried changing the fields like Domain which had values between (0,1) for most, but some rows had values -1 , which as per me meant that they had no domain field. However giving them negative valued seemed like bad choice so I tried changing that too.

```
for i in range(0,3998):
    c=df.loc[i,'Domain']
    if(c<0):
        df.loc[i,'Domain']='0'
df.Domain.head(19)
```

```
Out[30]: 0     0.635979
         1     0.960603
         2     0.450877
         3     0.974396
         4     0.124502
         5            0
         6     0.356536
         7     0.829585
         8     0.694479
         9     0.493596
         10    0.765674
         11    0.968237
         12    0.229482
         13    0.538387
         14           0
         15    0.308401
         16           0
         17    0.911395
         18    0.563268
         Name: Domain, dtype: object
```

Scaling- Also I used the preprocessing.scale() function to scale all my data to improve the model. This function changes every field such that their mean=0 and variance=1. (Normalisation)

Though these changes though seemed better at first , but after finalising the random state and training size , it showed better accuracy without them.

(with modification)

```
legeCityTier','English','Logical','Quant','ComputerProgramming','ElectronicsAndSemicon','10board','12board','Domain','Ele
```

```
In [89]:
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=11,train_size=0.8806)
model.fit(X_train,y_train)
print("Model Score- " ,model.score(X_test,y_test))
y_predicted=model.predict(X_test)
print("Accuracy in (%) =",accuracy_score(y_test,y_predicted)*100,"%")
print()
print("Classification Report")
print()
labels=["0 - Low Salary","1- High Salary"]
print(classification_report(y_test, y_predicted,target_names=labels))
```

```
Model Score-  0.7301255230125523
Accuracy in (%) = 73.01255230125523 %

Classification Report

              precision    recall  f1-score   support

0 - Low Salary       0.71      0.68      0.70       216
1- High Salary       0.75      0.77      0.76       262

      accuracy                           0.73       478
     macro avg       0.73      0.73      0.73       478
  weighted avg       0.73      0.73      0.73       478
```

(without modification/string fields)

```
','Logical','Quant','ComputerProgramming','ElectronicsAndSemicon','Domain','ElectricalEngg','Civi
```

```
93]:
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=11,train_size=0.8806)
model.fit(X_train,y_train)
print("Model Score- " ,model.score(X_test,y_test))
y_predicted=model.predict(X_test)
print("Accuracy in (%) =",accuracy_score(y_test,y_predicted)*100,"%")
print()
print("Classification Report")
print()
labels=["0 - Low Salary","1- High Salary"]
print(classification_report(y_test, y_predicted,target_names=labels))
```

```
Model Score-  0.7426778242677824
Accuracy in (%) = 74.26778242677824 %

Classification Report

              precision    recall  f1-score   support

0 - Low Salary       0.73      0.69      0.71       216
1- High Salary       0.75      0.79      0.77       262

      accuracy                           0.74       478
     macro avg       0.74      0.74      0.74       478
  weighted avg       0.74      0.74      0.74       478
```

Shuffling and finding best Training size:

Now to find the best random state and Training size I did this:

```
Maxscore=0
randstate=0
trainsize=0;
for i in range(1,20):
    for j in range(40,90):
        X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=i,train_size=j/100)
        model.fit(X_train,y_train)
        z=model.score(X_test,y_test)
        if(z>Maxscore):
            print("MaxScore",z,j)
            Maxscore=z
            randstate=i
            trainsize=j
```

As you can see I basically tried all combinations of random states and training sizes and stored the best one in some variable:

The result for them actually came out to be randstate=11 and training size=0.88 with max accuracy 74.2%

Now one can say that increasing the training size would always lead to better accuracy , but this is not true as increasing this training size to even 0.9 would actually give a lower accuracy. Plus one should also maintain the test size to be atleast 10%(As even with the current model about 500 samples are tested).

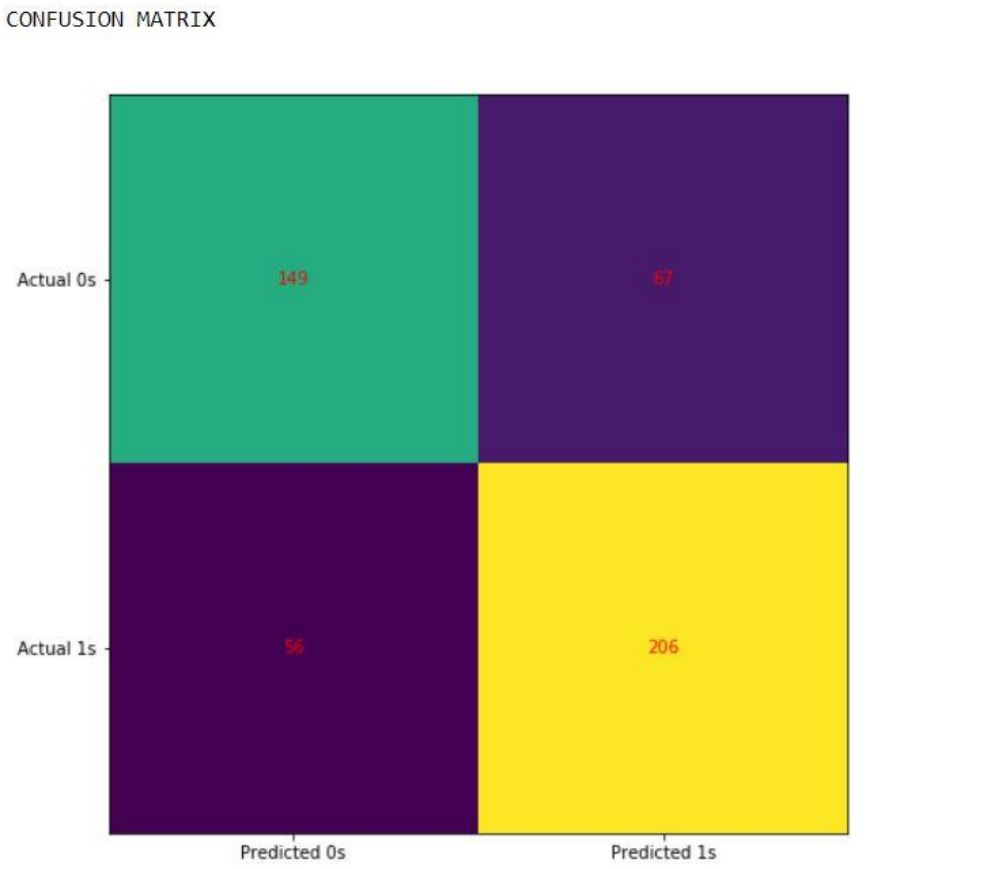
However I still have all the other best accuracy states and sizes, for example the next best accuracy with a lower training size of 0.64 is at randstate=12 , with accuracy of 73.2%

This really helped to get a higher accuracy. However, this code took atleast 20 mins to process as it is trying different training sizes and random states 1000 times.

But then I only had to run it once for getting the best result.

Analysis:

I used matplot lib extensively to form a heat map of the Confusion Matrix.



And then the clarification report of each class (here only 0 and 1).

```
X_train, X_test, y_train, y_test = train_test_split(X,y,random_state=11,train_size=0.8806)
model.fit(X_train,y_train)
print("Model Score- " ,model.score(X_test,y_test))
y_predicted=model.predict(X_test)
print("Accuracy in (%) =",accuracy_score(y_test,y_predicted)*100,"%")
print()
print("Classification Report")
print()
labels=["0 - Low Salary","1- High Salary"]
print(classification_report(y_test, y_predicted,target_names=labels))
```

```
Model Score-  0.7426778242677824
Accuracy in (%) = 74.26778242677824 %

Classification Report

                precision    recall  f1-score   support

0 - Low Salary       0.73      0.69      0.71       216
1- High Salary       0.75      0.79      0.77       262

      accuracy                           0.74       478
     macro avg       0.74      0.74      0.74       478
  weighted avg       0.74      0.74      0.74       478
```

As per me , just providing all the columns(fields) given in the data to us to the training set doesn’t gives us the most accurate data, we need to try out a different combinations of field that we know are of some importance to the end result , and if they don’t help to increase accuracy one can omit them , as they might be providing bad data to our machine learning model.

Also one should try out shuffling the data in different ways ,as say just picking the first 100 rows of our data and training on only them and testing on the ones below is a bad method.

Same goes with the training set sizes.