Eco482 A1

October 9, 2024

1 Algorithmic implementation

[11]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt

from sklearn import preprocessing
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import confusion_matrix, ConfusionMatrixDisplay
from sklearn.metrics import roc_auc_score, roc_curve
from sklearn.model_selection import train_test_split, cross_val_score

from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.neighbors import KNeighborsClassifier
```

1.1

[2]:
```
df = pd.read_csv("mobile_money.csv")


df = df.apply(lambda x: x.replace({'yes': True, 'no': False}) if x.dtype ==␣
 ↪'object' else x)

param = ['mpesa_user','cellphone', 'totexppc', 'wkexppc', 'wealth', 'size',␣
 ↪'education_years',
                'pos', 'user_neg', 'ag', 'user_sick', 'sendd', 'recdd',␣
 ↪'bank_acct',
                'mattress', 'sacco', 'merry', 'occ_farmer', 'occ_public',␣
 ↪'occ_prof',
                 'occ_help', 'occ_bus', 'occ_sales', 'occ_ind', 'occ_other',␣
 ↪'occ_ue']

features_name = ['cellphone', 'totexppc', 'wkexppc', 'wealth', 'size',␣
 ↪'education_years',
```

```
                'pos', 'user_neg', 'ag', 'user_sick', 'sendd', 'recdd',␣
↪'bank_acct',
                'mattress', 'sacco', 'merry', 'occ_farmer', 'occ_public',␣
↪'occ_prof',
                 'occ_help', 'occ_bus', 'occ_sales', 'occ_ind', 'occ_other',␣
↪'occ_ue']
df = df[param].dropna()
outcome = df['mpesa_user']
df_feat = df[features_name]

df
```

[2]:

```
      mpesa_user  cellphone  totexppc  wkexppc    wealth  size  \
0           True        1.0  150808.0  68640.0  202600.0   1.0
1          False        0.0   33866.4  15246.4   13300.0   5.0
2           True        1.0   36072.8  15194.4  149700.0   5.0
3           True        1.0   21632.0   9412.0   20000.0   5.0
4           True        1.0   29848.0  15236.0   31000.0   5.0
…            …        …         …        …         …     …
2277        True        1.0  150200.0  66456.0   89100.0   5.0
2278        True        1.0   41142.5  30030.0   55450.0   8.0
2279       False        1.0   14102.0  11934.0   57130.0  10.0
2280        True        0.0   21236.5  11381.5  133500.0   8.0
2281       False        1.0   25375.5  19383.0   62500.0   8.0

      education_years  pos  user_neg  ag  …  merry  occ_farmer  occ_public  \
0                 3.0    0         0   0  …  False           0           0
1                 8.0    0         0   0  …   True           0           0
2                 0.0    0         0   0  …   True           0           0
3                12.0    0         1   0  …  False           0           0
4                 0.0    0         1   0  …  False           0           0
…                …  …        …  ..  …    …      …          …
2277              8.0    0         1   0  …   True           0           0
2278             12.0    0         1   1  …   True           1           0
2279              7.0    0         0   0  …   True           0           0
2280              0.0    0         1   0  …  False           0           0
2281              3.0    0         0   0  …   True           0           0

     occ_prof  occ_help  occ_bus  occ_sales  occ_ind  occ_other  occ_ue
0           0         0        0          0        0          1       0
1           0         0        1          0        0          0       0
2           1         0        0          0        0          0       0
3           1         0        0          0        0          0       0
4           0         0        0          1        0          0       0
…       …       …       …        …       …        …      …
2277        1         0        0          0        0          0       0
2278        0         0        0          0        0          0       0
```

```
2279         1        0       0         0        0          0       0
2280         0        0       0         0        0          1       0
2281         0        1       0         0        0          0       0

[2261 rows x 26 columns]
```

1.2

[3]:
```python
outcome.describe()
```

[3]:
```
count      2261
unique        2
top        True
freq       1667
Name: mpesa_user, dtype: object
```

1.3

[20]:
```python
grouped_stats = df.groupby('mpesa_user')[features_name].describe()
grouped_stats.T.head(50)
for var in features_name: print(df.groupby('mpesa_user')[var].describe())
```

```
             count      mean       std  min  25%  50%  75%  max
mpesa_user
False        594.0  0.427609  0.495149  0.0  0.0  0.0  1.0  1.0
True        1667.0  0.922615  0.267281  0.0  1.0  1.0  1.0  1.0
             count          mean            std     min        25%        50%  \
mpesa_user
False        594.0  54237.974805   93427.263844   480.0  18554.200  31136.065
True        1667.0  84476.756585  102949.985330  2306.0  34411.905  57204.000

                75%        max
mpesa_user
False       53756.5  1576484.0
True        98570.0  1870776.0
             count          mean           std     min        25%        50%  \
mpesa_user
False        594.0  28578.544063  28099.286689     0.0  13000.000  20297.335
True        1667.0  35493.869666  27961.784440  1397.5  18412.625  27726.400

                  75%       max
mpesa_user
False       32844.500  237484.0
True        42734.095  263380.0
             count          mean           std  min      25%      50%  \
mpesa_user
False        594.0   76478.690236  2.970897e+05  0.0   7062.5  20250.0
```

```
True        1667.0  214923.106779  1.460980e+06  0.0  24950.0  54000.0

                 75%         max
mpesa_user
False        50225.0   4753200.0
True        112600.0  47200000.0
             count      mean       std  min  25%  50%  75%   max
mpesa_user
False        594.0  4.126263  2.449328  1.0  2.0  4.0  6.0  12.0
True        1667.0  4.262747  2.176337  1.0  3.0  4.0  6.0  13.0
             count      mean       std  min  25%  50%   75%   max
mpesa_user
False        594.0  6.489899  4.624234  0.0  2.0  7.0  10.0  19.0
True        1667.0  8.373725  5.287427  0.0  5.0  9.0  12.0  19.0
             count      mean       std  min  25%  50%  75%  max
mpesa_user
False        594.0  0.048822  0.215676  0.0  0.0  0.0  0.0  1.0
True        1667.0  0.072585  0.259533  0.0  0.0  0.0  0.0  1.0
             count      mean       std  min  25%  50%  75%  max
mpesa_user
False        594.0  0.000000  0.000000  0.0  0.0  0.0  0.0  0.0
True        1667.0  0.529694  0.499267  0.0  0.0  1.0  1.0  1.0
             count      mean       std  min  25%  50%  75%  max
mpesa_user
False        594.0  0.139731  0.347000  0.0  0.0  0.0  0.0  1.0
True        1667.0  0.108578  0.311203  0.0  0.0  0.0  0.0  1.0
             count      mean       std  min  25%  50%  75%  max
mpesa_user
False        594.0  0.000000  0.000000  0.0  0.0  0.0  0.0  0.0
True        1667.0  0.389922  0.487879  0.0  0.0  0.0  1.0  1.0
             count      mean       std  min  25%  50%  75%  max
mpesa_user
False        594.0  0.198653  0.399323  0.0  0.0  0.0  0.0  1.0
True        1667.0  0.615477  0.486628  0.0  0.0  1.0  1.0  1.0
             count      mean       std  min  25%  50%  75%  max
mpesa_user
False        594.0  0.173401  0.378912  0.0  0.0  0.0  0.0  1.0
True        1667.0  0.518296  0.499815  0.0  0.0  1.0  1.0  1.0
           count unique    top  freq
mpesa_user
False        594      2  False   432
True        1667      2   True  1188
           count unique   top  freq
mpesa_user
False        594      2  True   496
True        1667      2  True  1145
           count unique    top  freq
mpesa_user
```

```
False         594      2  False   527
True         1667      2  False  1289
            count unique    top freq
mpesa_user
False         594      2  False  356
True         1667      2  False  851
              count      mean       std  min  25%  50%  75%  max
mpesa_user
False         594.0  0.367003  0.482394  0.0  0.0  0.0  1.0  1.0
True         1667.0  0.136773  0.343710  0.0  0.0  0.0  0.0  1.0
              count      mean       std  min  25%  50%  75%  max
mpesa_user
False         594.0  0.005051  0.070947  0.0  0.0  0.0  0.0  1.0
True         1667.0  0.051590  0.221264  0.0  0.0  0.0  0.0  1.0
              count      mean       std  min  25%  50%  75%  max
mpesa_user
False         594.0  0.132997  0.339857  0.0  0.0  0.0  0.0  1.0
True         1667.0  0.240552  0.427547  0.0  0.0  0.0  0.0  1.0
              count      mean       std  min  25%  50%  75%  max
mpesa_user
False         594.0  0.085859  0.280391  0.0  0.0  0.0  0.0  1.0
True         1667.0  0.144571  0.351773  0.0  0.0  0.0  0.0  1.0
              count      mean       std  min  25%  50%  75%  max
mpesa_user
False         594.0  0.171717  0.377452  0.0  0.0  0.0  0.0  1.0
True         1667.0  0.162567  0.369081  0.0  0.0  0.0  0.0  1.0
              count      mean       std  min  25%  50%  75%  max
mpesa_user
False         594.0  0.074074  0.262112  0.0  0.0  0.0  0.0  1.0
True         1667.0  0.112777  0.316415  0.0  0.0  0.0  0.0  1.0
              count      mean       std  min  25%  50%  75%  max
mpesa_user
False         594.0  0.031987  0.176112  0.0  0.0  0.0  0.0  1.0
True         1667.0  0.026995  0.162116  0.0  0.0  0.0  0.0  1.0
              count      mean       std  min  25%  50%  75%  max
mpesa_user
False         594.0  0.042088  0.200958  0.0  0.0  0.0  0.0  1.0
True         1667.0  0.044991  0.207347  0.0  0.0  0.0  0.0  1.0
              count      mean       std  min  25%  50%  75%  max
mpesa_user
False         594.0  0.089226  0.285309  0.0  0.0  0.0  0.0  1.0
True         1667.0  0.079184  0.270107  0.0  0.0  0.0  0.0  1.0
```

1.4

[5]:
```
#Split the data set into training and test 80-20

x_train, x_test, y_train, y_test = train_test_split(df_feat, outcome,
                                                    test_size = 0.2,␣
 ↪random_state = 21)

#Standardizing the data set
#stage 1
scaler = preprocessing.StandardScaler()
#stage2
scaler.fit(x_train)
#stage 3
x_train_scaled = scaler.transform(x_train)
x_test_scaled = scaler.transform(x_test)

#Logistic regression
log_reg = LogisticRegression()
log_reg.fit(x_train_scaled, y_train)

#Random Forest  Classifier
rforest = RandomForestClassifier()
rforest.fit(x_train_scaled, y_train)

#LDA
lda = LinearDiscriminantAnalysis()
lda.fit(x_train_scaled, y_train)
```

[5]:
```
LinearDiscriminantAnalysis()
```

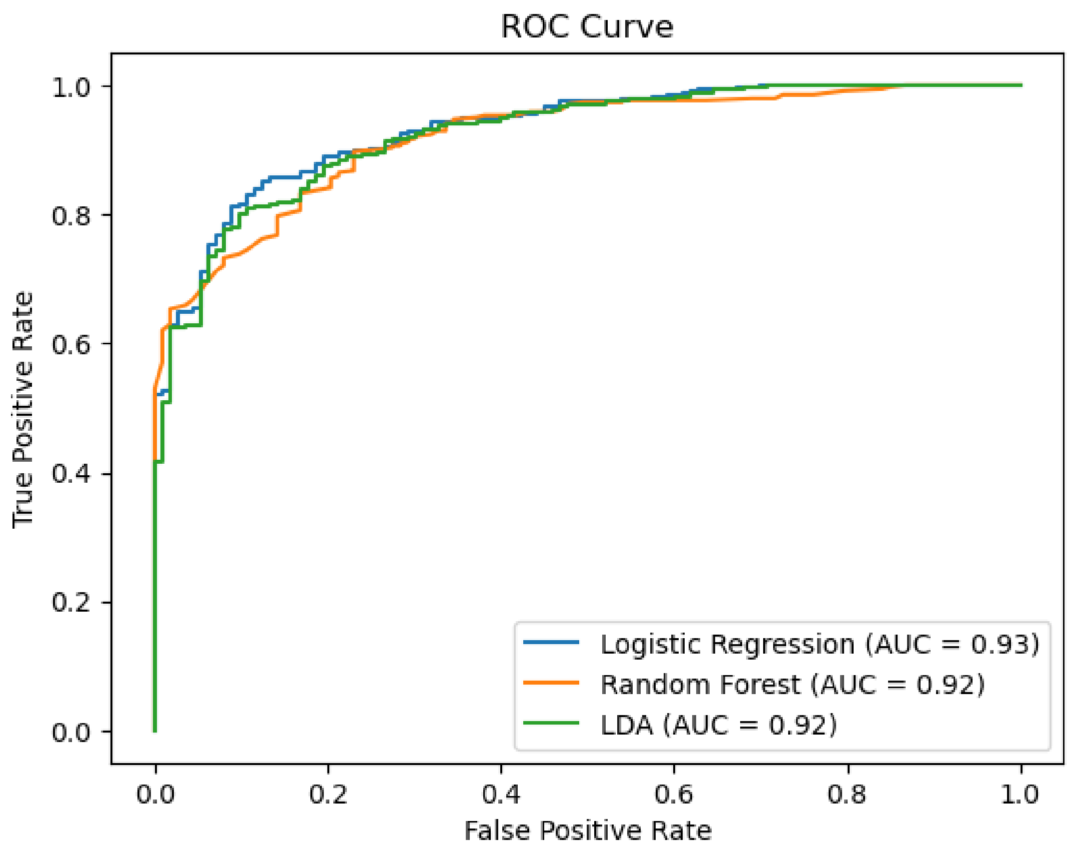
1.5

The best classifier is the Logistic Regression, as can be seen in the following.

[6]:
```
#Getting accuracies for the classifiers
accuracy = dict()

log_score = log_reg.score(x_test_scaled, y_test)

rforest_score = rforest.score(x_test_scaled, y_test)

lda_score = lda.score(x_test_scaled, y_test)

accuracy['Logistic regression accuracy'] = log_score
accuracy['Random forest accuracy'] = rforest_score
accuracy['LDA accuracy'] = lda_score
```

```
accuracy_df = pd.DataFrame(accuracy, index=[0])
print(accuracy_df)


#Getting the AUC scores for the classifiers

log_auc = roc_auc_score(y_test, log_reg.predict_proba(x_test_scaled)[:, 1])
fpr_log, tpr_log, _ = roc_curve(y_test, log_reg.predict_proba(x_test_scaled)[:,␣
↪1])

rf_auc = roc_auc_score(y_test, rforest.predict_proba(x_test_scaled)[:, 1])
fpr_rf, tpr_rf, _ = roc_curve(y_test, rforest.predict_proba(x_test_scaled)[:,␣
↪1])

lda_auc = roc_auc_score(y_test, lda.predict_proba(x_test_scaled)[:, 1])
fpr_lda, tpr_lda, _ = roc_curve(y_test, lda.predict_proba(x_test_scaled)[:, 1])

#Plotting the curves
plt.figure()
plt.plot(fpr_log, tpr_log, label=f'Logistic Regression (AUC = {log_auc:.2f})')
plt.plot(fpr_rf, tpr_rf, label=f'Random Forest (AUC = {rf_auc:.2f})')
plt.plot(fpr_lda, tpr_lda, label=f'LDA (AUC = {lda_auc:.2f})')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend(loc='lower right')
plt.show()
```

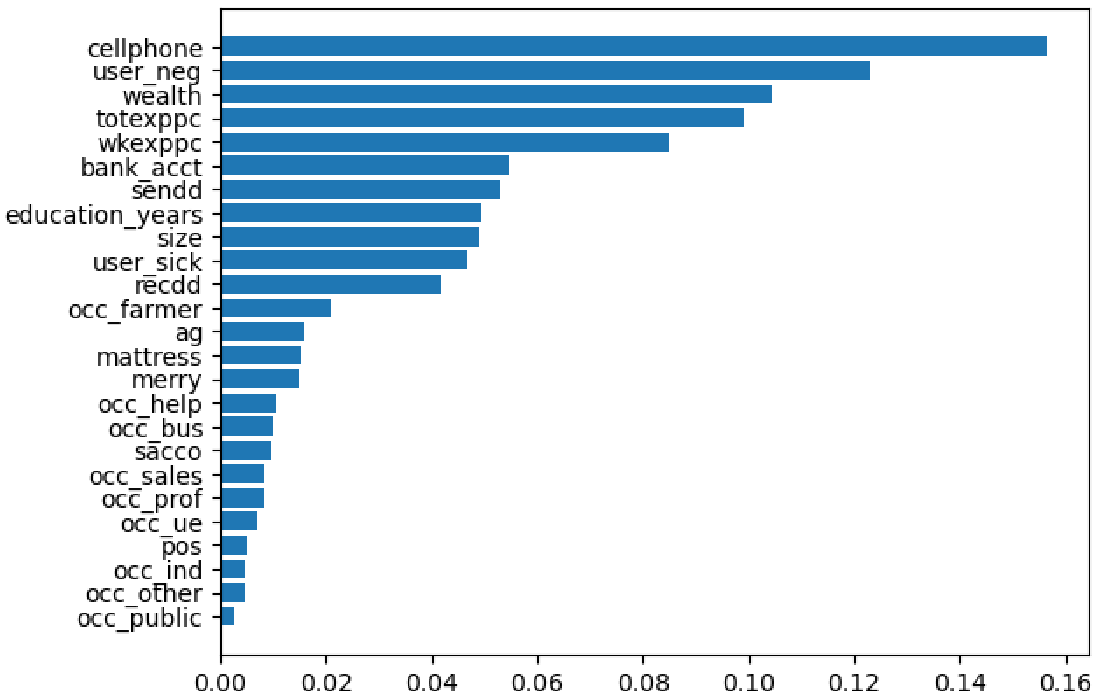
```
   Logistic regression accuracy  Random forest accuracy  LDA accuracy
0                      0.871965                0.863135       0.86755
```



1.6

[7]:
```
snames, simportances = zip(*sorted(zip(features_name, rforest.
↪feature_importances_),
                                  key=lambda pair: pair[1]))
plt.barh(snames, simportances)
```

[7]: <BarContainer object of 25 artists>



1.7

[12]:
```python
k_value = range(1, 11)
accuracies = []

for k in k_value:
    knn = KNeighborsClassifier(n_neighbors=k)
    cv_scores = cross_val_score(knn, x_train_scaled, y_train, cv=5)
    accuracies.append(cv_scores.mean())

for k, acc in zip(k_value, accuracies):
    print(f"K={k}: Mean Accuracy={acc:.4f}")
```

```
K=1: Mean Accuracy=0.8081
K=2: Mean Accuracy=0.7799
K=3: Mean Accuracy=0.8346
K=4: Mean Accuracy=0.8258
K=5: Mean Accuracy=0.8490
K=6: Mean Accuracy=0.8440
K=7: Mean Accuracy=0.8523
K=8: Mean Accuracy=0.8468
K=9: Mean Accuracy=0.8518
K=10: Mean Accuracy=0.8507
```

1.8

[21]:
```
log_regl = LogisticRegression(solver='liblinear')
param_grid = {'penalty':['l1', 'l2'],
              'C':[0.1, 1, 10, 100]}

grid_search = GridSearchCV(estimator = log_regl, param_grid = param_grid,
                           cv = 5)

grid_search.fit(x_train_scaled, y_train)

for mean_score, params in zip(grid_search.cv_results_['mean_test_score'],␣
 ↪grid_search.cv_results_['params']):
    print(f"C={params['C']}, Penalty={params['penalty']}: Mean CV␣
 ↪Score={mean_score:.4f}")
```

```
C=0.1, Penalty=l1: Mean CV Score=0.8794
C=0.1, Penalty=l2: Mean CV Score=0.8739
C=1, Penalty=l1: Mean CV Score=0.8778
C=1, Penalty=l2: Mean CV Score=0.8766
C=10, Penalty=l1: Mean CV Score=0.8766
C=10, Penalty=l2: Mean CV Score=0.8766
C=100, Penalty=l1: Mean CV Score=0.8766
C=100, Penalty=l2: Mean CV Score=0.8766
```

1.9

In our result, it is observed that when C increases from 0.1 to 100 (meaning regularization becomes weaker), the Mean CV Score stabilizes at approximately 0.8766. This suggests that when C reaches a specific point, additional decrease in regularization does not enhance the model's effectiveness. The optimal result is achieved when C=0.1 with l1 regularization, indicating that a moderate amount of regularization assists the model in managing bias and variance, resulting in improved generalization. Nevertheless, as the C rises, there is a possibility of slight overfitting in the model, leading to a plateau in the score without notable improvements. Regularization helps make sure the model is less complex and reduces the chances of overfitting to noise in the training data, resulting in improved cross-validation performance with optimal regularization (moderate C).

1.10

The Logistic Regression is the best classifier for our data. It has the highest cross-validation accuracy of 0.8794 for C=0.1, L1 penalty and the highest AUC of 0.93 compared to the other classifiers. While KNN, Random Forest and LDA has good accuracy, their accuracy and AUC values are lower compared to the Logistic Regression's.

1.11

The main findings from the research on M-Pesa adoption include: 1.Cell phone possession was identified as the most important factor, with having a bank account, overall wealth, sending money transfers, and receiving money transfers following closely behind. 2.Logistic Regression stood out among the classifiers, delivering top performance with a 0.879 cross-validation accuracy and a 0.93 AUC, establishing it as the most dependable model for forecasting M-Pesa adoption. Additional classifiers such as Random Forest, K-Nearest Neighbors (KNN), and Linear Discriminant Analysis (LDA) had strong performances, but Logistic Regression consistently outshined them in terms of accuracy and AUC. 3.These results indicate that having access to communication technology and being financially included are essential for the adoption of M-Pesa in Kenya.