University of Hertfordshire
School of Computer Science
BSc Computer Science with Artificial Intelligence

Module: Machine Learning and Neural Computing
Coursework: Data Classification

Experimental Report

Mohamed Shaheen
Level 6
Academic Year 2022-2023

Task1 – Data Exploration

The first thing needed to start the assignment is importing all the necessary libraries and functions. For this Task, pandas, matplotlib, and sklearn libraries were needed.

```
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn import metrics
from sklearn.metrics import confusion_matrix
from sklearn.metrics import classification_report
import seaborn as sns; sns.set()
```

(a) Afterward, the pandas library was used to load the CSV data we need to work on using the pandas read_csv function and providing it with the CSV file path. Both the training audit data and test audit data were loaded in training_set1 and test_set1 respectively as follows.

```
training_set1 = pd.read_csv(r'C:\Users\ADMIN\Desktop\training_audit_data.csv')
test_set1 = pd.read_csv(r'C:\Users\ADMIN\Desktop\test_audit_data-1.csv')
```

(b) The data labels of the training set were retrieved by using the iloc property of the training set. As shown below, all rows of the last column have been retrieved in the Labels variable.

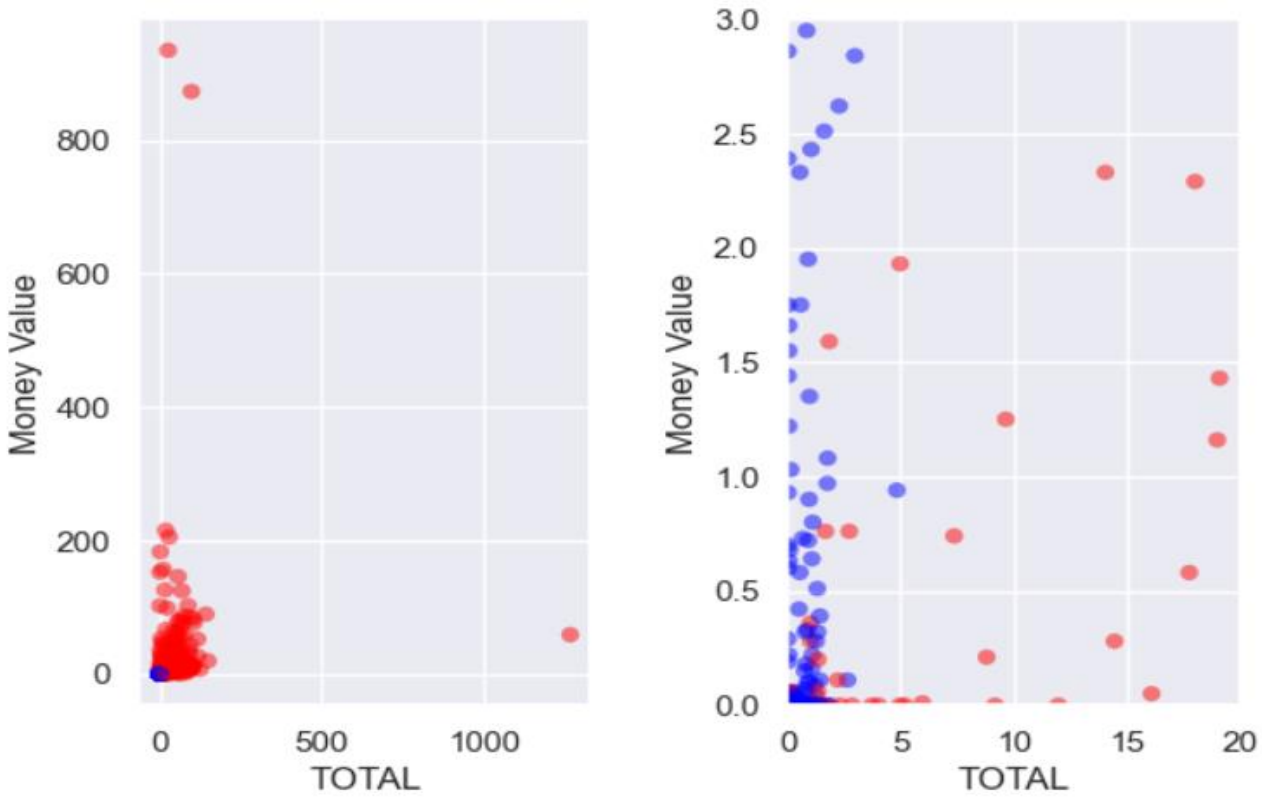
```
Labels = training_set1.iloc[:,-1]
```

Afterward, the Total column in the data (fifth column index 4) and the Money Value column (7th column index 6) is plotted two times. Two subplots are created as shown below on 1 row and 2 columns. The colors are set according to the data point label with blue if the label is 0 (indicating no Risk) and red if the label is 1 (indicating Risk).

```
fig, axes = plt.subplots(nrows=1, ncols=2)
fig.tight_layout(pad=2.50)

plt.subplot(121)
dots_trn=plt.scatter(training_set1.iloc[:,4],training_set1.iloc[:,6], c=['blue' if i == 0 else 'red' for i in Labels],
                     edgecolor='none', alpha=0.5)
plt.xlabel('TOTAL')
plt.ylabel('Money Value')


plt.subplot(122)
dots_trn2=plt.scatter(training_set1.iloc[:,4],training_set1.iloc[:,6], c=['blue' if i == 0 else 'red' for i in Labels],
                      edgecolor='none', alpha=0.5)
plt.xlabel("TOTAL")
plt.ylabel("Money Value")
plt.xlim([0,20])
plt.ylim([0,3])
```

As shown below, the first graph showed that the data has outliers which made the graph not represent the data in a proper way. In the second graph, the data was limited to a maximum of 20 in total and a maximum of 3 on Money Value, which clarified the data. The second graph does better at representing the data without the outliers as we can see now that the no-risk data lies in the range [0: 5] Total and can be seen to go up to 3 in money value.

```
(0.0, 3.0)
```



(c) The next step is to normalize the training and test data in preparation for principal component analyses and capturing the variances of the data. The following code indicates all training data and test data except the last column of each to be saved at Inputs and Inputs_test respectively. Following that, a StandardScaler object was defined and was used to fit the training data extracting the mean and standard deviation of each feature, then it was stored in the statistics variable. Afterward, the means of the training set and test set were removed and both sets were scaled accordingly to have a unit variance between 0 and 1 as shown in the code below.

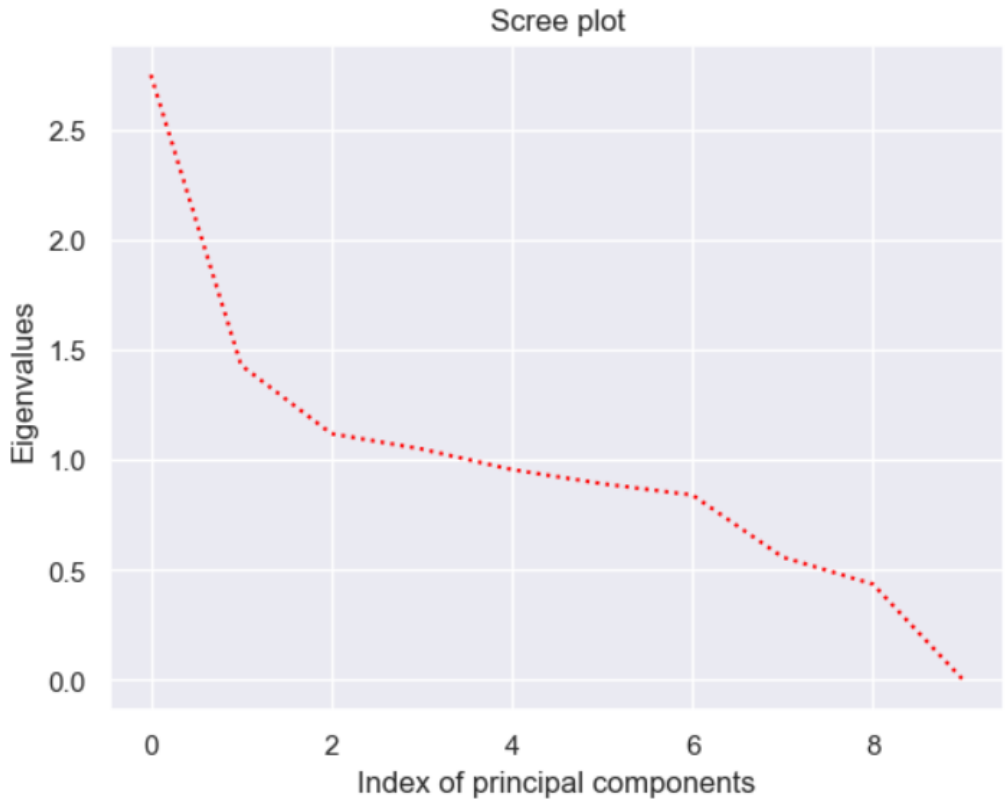
```
Inputs = training_set1.iloc[:,:-1]
Inputs_test = test_set1.iloc[:,:-1]

statistics = StandardScaler().fit(Inputs)
x1 = statistics.transform(Inputs)
x2 = statistics.transform(Inputs_test)
```

(d) Next step is to perform a PCA on the scaled data using sklearn PCA. Firstly, a PCA object is stored in the pca variable and then is used to fit transform the scaled training set (x1) from the last step in order to obtain the training set PCA data projection. Following that a Scree plot was made to project the explained variance ratio of each principal component as shown in the code below. Moreover, the explained variance ratios were printed out as well as their sum to get a better understanding of the data.

```
pca = PCA()
data_projections = pca.fit_transform(x1)

figure = plt.figure()
ax = plt.gca()
plt.plot(pca.explained_variance_, color='red', linestyle='dotted')
ax.set_title("Scree plot")
ax.set_xlabel("Index of principal components")
ax.set_ylabel("Eigenvalues")
print("Explained Variance ratios: ",pca.explained_variance_ratio_)
print('\n Total Variance Explained:', round(sum(list(pca.explained_variance_ratio_))*100, 2))
```

As shown in the figure below, the total explained variance captured by this data reached 100% and the variance ratio of each feature is shown and clarified through the figure. We can see from the figures that the line has a negative slope indicating that the explained variance of the first components is high and it decreases as we go through the other components. From the figure, it is derived that more than 80% of the variance is explained through the first 6 components.

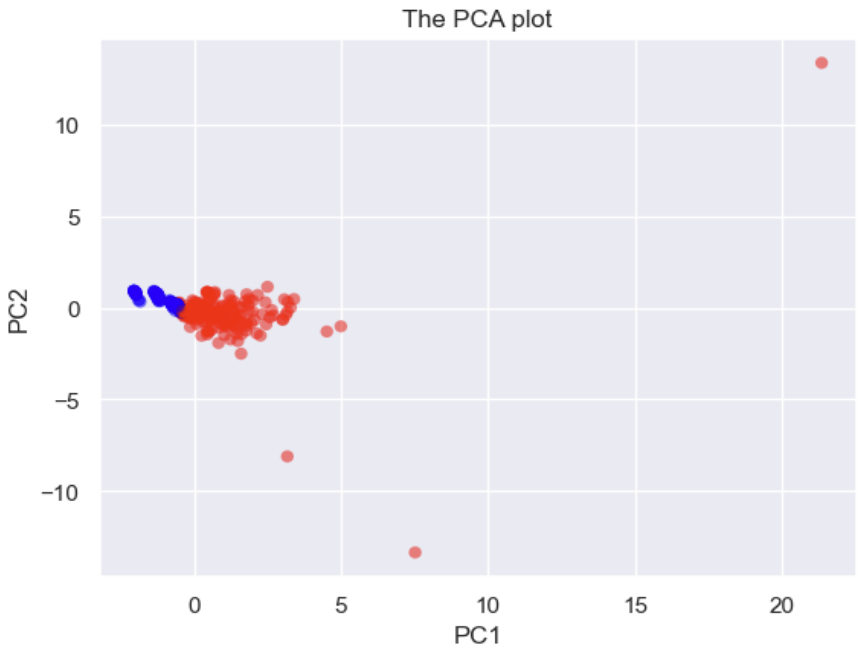
```
Explained Variance ratios:  [2.74123511e-01 1.42593272e-01 1.11590565e-01 1.04623242e-01
 9.53081367e-02 8.89026520e-02 8.37932726e-02 5.56670926e-02
 4.33982556e-02 1.14160993e-35]

 Total Variance Explained: 100.0
```



(e) Next step is to plot projections of the first principal component against the second principal component obtained from the PCA data projections in the previous step. As can be seen in the below figure, PCA data projections index 0 and index 1 of training data were plotted against each other, and the following graph shows the presence of outliers in the Label 1 Risk data points (Red points). Moreover, the graph shows that the data are separated into two clusters the left cluster is for the 0 Risk label data and the right one is for the 1 Risk label data. However, both categories overlap each other to a certain extent however it wasn't visible in this graph due to the presence of outliers.

```
figure = plt.figure()
ax = plt.gca()
plt.scatter(data_projections[:,0],data_projections[:,1], c=['blue' if i == 0 else 'red' for i in Labels],
            edgecolor='none', alpha=0.5)
ax.set_title("The PCA plot")
ax.set_xlabel("PC1")
ax.set_ylabel("PC2")
```

```
Text(0, 0.5, 'PC2')
```



(f) Lastly, the projections for the test data set were obtained on the same scale as that of the training set by transforming it based on the training set fit object as follows.

```
data_projections_test = scaler_data.transform(x2)
```

Task2 – Data Preparation

(a) In this stage, the training data will be further split into a training set and a validation set in order to test the SVM classifier in the next task. Furthermore, the data will be normalized as was done before passing it to the SVM classifier. The data was split into a 30% validation set (118 rows) and a 70% training set (273 rows) as seen below by using the sklearn train_test_split function.

```
Xtrain, Xtest, ytrain, ytest = train_test_split(training_set1.iloc[:,:-1], training_set1.iloc[:,-1], test_size = 0.30,
                                                random_state=42)
```

```
Xtrain.shape
```

```
(273, 10)
```

```
Xtest.shape
```

```
(118, 10)
```

(b) The data was scaled afterward was done in Task1 (c) as follows.

```
Scaler = StandardScaler().fit(Xtrain)
scaled_trnX = Scaler.transform(Xtrain)

scaled_valX = Scaler.transform(Xtest)
```

Task3 – SVM Classification

(a) (i) In this task, multiple parameters are tried on the SVM classifier to see which one performs the best on the data and use it for our final classification model. The following figure shows the best combination that was tried which used the Gaussian radial basis kernel with a gamma of 10 and penalty parameter (C) of 5, for the rest of the combinations that were used please refer to the appendix.

```
svc3 = SVC(kernel='rbf', class_weight='balanced', C=5, gamma=10)
model3 = svc3.fit(scaled_trnX, ytrain)

vyfit3 = model3.predict(scaled_valX)

print('Accuracy:', metrics.accuracy_score(ytest,vyfit3))

print(classification_report(ytest, vyfit3, target_names=['0','1']))
```

```
Accuracy: 0.9915254237288136
              precision    recall  f1-score   support

           0       1.00      0.98      0.99        63
           1       0.98      1.00      0.99        55

    accuracy                           0.99       118
   macro avg       0.99      0.99      0.99       118
weighted avg       0.99      0.99      0.99       118
```

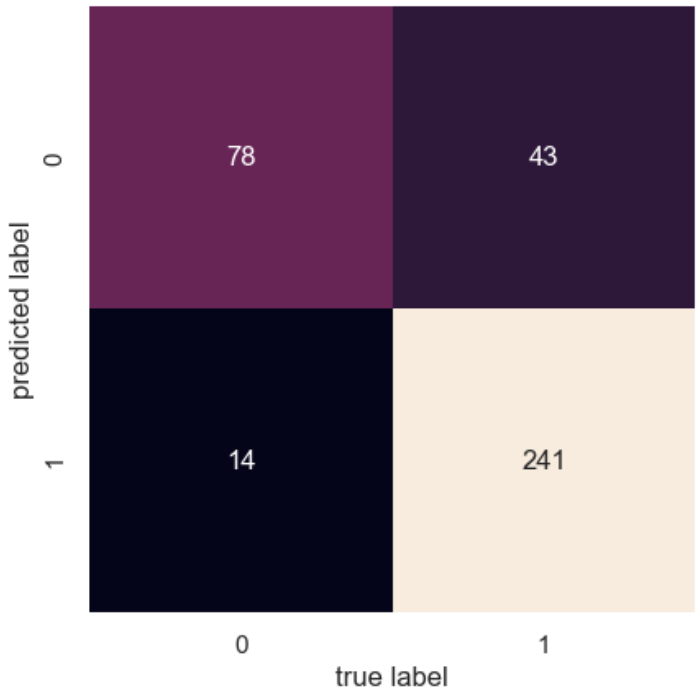
The above figure shows the SVM classifier was trained on the second training set and was made to predict the scaled test training data in which the model achieved an accuracy of 99%. The classification report shows that for the 0 Risk label class the model predictions were all correct achieving 100% precision accuracy and recall shows that 98% of the positive predictions for this class were correctly classified by the model and vice versa for the 1 Risk label class. The f1-score is the weighted harmonic mean of precision and recall and it shows that 99% of both classes' positive prediction was correct. The support column shows how many occurrences of the class were present in the dataset.

(a) (ii) The next step is to use the best parameters found in the previous step to train a model on the whole normalized training set and test it on the whole original normalized test set as follows. In the figure below, the model was tested on the scaled original test set and a classification report, and a confusion matrix were produced.

```
trainingX1 = training_set1.iloc[:,:-1]
testX1 = test_set1.iloc[:,:-1]

train_validation1 = training_set1.iloc[:,-1]
test_validation1 = test_set1.iloc[:,-1]

statistics1 = StandardScaler().fit(trainingX1)

scaled_training_set1 = statistics1.transform(trainingX1)
scaled_test_set1 = statistics1.transform(testX1)

svc5 = SVC(kernel='rbf', class_weight='balanced', C=penalty_parameter, gamma=suitable_gamma)
model5 = svc5.fit(scaled_training_set1,train_validation1)
vyfit5 = model5.predict(scaled_test_set1)

print('Accuracy:', metrics.accuracy_score(test_validation1,vyfit5))
print(classification_report(test_validation1, vyfit5, target_names=['0','1']))

mat1 = confusion_matrix(test_validation1, vyfit5)
sns.heatmap(mat1.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=['0','1'],
            yticklabels=['0','1'])
plt.xlabel('true label')
plt.ylabel('predicted label');
```

In the below figure, the model was shown to have approximately an 85% accuracy rate and it was shown to classify 1 labeled Risk data much more efficiently than 0 Labeled data it can be explained by the imbalanced data as most of the labels were 1 Risk category (284 out of 376) shown by the support column.

```
Accuracy: 0.848404255319149
              precision    recall  f1-score   support

           0       0.64      0.85      0.73        92
           1       0.95      0.85      0.89       284

    accuracy                           0.85       376
   macro avg       0.79      0.85      0.81       376
weighted avg       0.87      0.85      0.85       376
```



(b) (i) From the plot generated at task 1(d) it was noticed that the first 6 components are responsible for 81.71% of the data variance which makes them suitable for feature reduction as they can well represent the whole set of data.

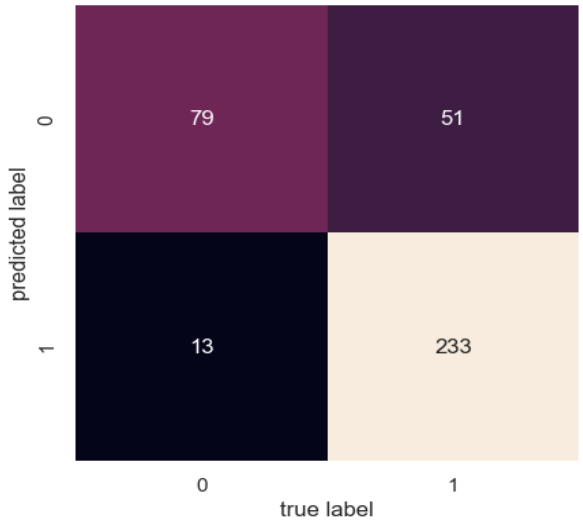
(b) (ii) The features were then reduced using the PCA n_components parameter which takes the number of features we want to retain as follows.

```
pca_final = PCA(n_components = 6)
scaler_final = pca_final.fit(scaled_final_trainX)
train_data_projections = pca_final.transform(scaled_final_trainX)
test_data_projections = pca_final.transform(scaled_final_testX)
```

(b) (iii) The final model was generated with the reduced features using PCA and the suitable parameters found in task 3(a). Moreover, a classification report and a confusion matrix were made as follows.

```
svc6 = SVC(kernel='rbf', class_weight='balanced', C=penalty_parameter, gamma=suitable_gamma)
model6 = svc6.fit(train_data_projections,final_train_validation)

vyfit6 = model6.predict(test_data_projections)

print('Accuracy:', metrics.accuracy_score(final_test_validation,vyfit6))

print(classification_report(final_test_validation, vyfit6, target_names=['0','1']))

mat = confusion_matrix(final_test_validation, vyfit6)
sns.heatmap(mat.T, square=True, annot=True, fmt='d', cbar=False,
            xticklabels=['0','1'],
            yticklabels=['0','1'])
plt.xlabel('true label')
plt.ylabel('predicted label');
```

```
Accuracy: 0.8297872340425532
              precision    recall  f1-score   support

           0       0.61      0.86      0.71        92
           1       0.95      0.82      0.88       284

    accuracy                           0.83       376
   macro avg       0.78      0.84      0.80       376
weighted avg       0.86      0.83      0.84       376
```



As can be seen in the above report and confusion matrix, the accuracy of the model dropped by around 2% achieving a much similar result to the previous model. However, the precision on the 0-label class seemed to have dropped slightly more.

Task4 – Summary and Conclusion

From the data loaded and the graph representing the data the presence of outliers was clear. The data was imbalanced as it was biased towards the 1 Risk labeled data which in turn made the model biased toward label 1 class classification. More than 80% of the variance was noticed to be explained by the first 6 components and was seen in the scree plot as the variance ratio had a negative slope through the features. The first component and the second component of the PCA seemed to divide the data into two clusters with some overlap. The Model which used the reduced PCA features is overall better though its accuracy decreased on the model generated by the full features by 2%, as the model is much less complex reducing 10 features into 6 which means a 40% reduction in the complexity of the model. Moreover, fewer features mean less computation time for the model to be executed and less storage space as fewer data is needed. Model accuracy was expected to increase by feature reduction as it means less misleading data however the misbalanced data led to a decrease in precision of only the 0 labeled data.

Appendix

```
svc1 = SVC(kernel='rbf', class_weight='balanced', C=1, gamma=10)
model1 = svc1.fit(scaled_trnX, ytrain)

vyfit1 = model1.predict(scaled_valX)

print('Accuracy:', metrics.accuracy_score(ytest,vyfit1))

print(classification_report(ytest, vyfit1, target_names=['0','1']))
```

```
Accuracy: 0.9830508474576272
              precision    recall  f1-score   support

           0       1.00      0.97      0.98        63
           1       0.96      1.00      0.98        55

    accuracy                           0.98       118
   macro avg       0.98      0.98      0.98       118
weighted avg       0.98      0.98      0.98       118
```

```
svc2 = SVC(kernel='rbf', class_weight='balanced', C=1, gamma=0.5)
model2 = svc2.fit(scaled_trnX, ytrain)

vyfit2 = model2.predict(scaled_valX)

print('Accuracy:', metrics.accuracy_score(ytest,vyfit2))

print(classification_report(ytest, vyfit2, target_names=['0','1']))
```

```
Accuracy: 0.9745762711864406
              precision    recall  f1-score   support

           0       0.95      1.00      0.98        63
           1       1.00      0.95      0.97        55

    accuracy                           0.97       118
   macro avg       0.98      0.97      0.97       118
weighted avg       0.98      0.97      0.97       118
```

```
svc4 = SVC(kernel='linear', class_weight='balanced', C=0.1)
model4 = svc4.fit(scaled_trnX, ytrain)

vyfit4 = model4.predict(scaled_valX)

print('Accuracy:', metrics.accuracy_score(ytest,vyfit4))

print(classification_report(ytest, vyfit4, target_names=['0','1']))
```

```
Accuracy: 0.9152542372881356
              precision    recall  f1-score   support

           0       0.86      1.00      0.93        63
           1       1.00      0.82      0.90        55

    accuracy                           0.92       118
   macro avg       0.93      0.91      0.91       118
weighted avg       0.93      0.92      0.91       118
```