BUAN 6341
APPLIED MACHINE LEARNING
ASSIGNMENT 4
Clustering, Feature Selection and Transformation and Neural Network

Report Summary

- Prepared Two Datasets (Sgemm Product Dataset and Bank Dataset) for classification
- Implemented Neural Networks and K-Nearest Neighbors Classification algorithms to calculate different performance metrics on the datasets
- Implemented Neural Networks Classification with different number of layers and nodes, different activation functions, loss functions and optimizers
- Implemented KNN Classification with different number of neighbors and distance metrics

Data Description

- Dataset 1 is the dataset of factors affecting the GPU Kernel Runtime. This dataset was also used in the assignment 1 for Linear and Logistic Regression. The data has 241600 rows and 18 columns out of which last 4 columns (dependent variable columns) have been combined into one for classification purposes.
- Dataset 2 is a bank client dataset and the dependent variable is y (Has the client subscribed a term deposit? Y/N). This data has 20 features and 1 dependent binary variable. The features include data regarding to clients which help find out if the client will subscribe to a deposit or not.

Data Preparation

Dataset 1

- Took average of 4 RunTime columns to get 1 single dependent variable Run_Avg
- The values of the dependent variable (Run_Avg) are continuous which could not be used for classification and sampled 30% of total values for less implementation time.
- Divided the column values into 1 and 0 by changing all the values equal and above the median value to 1 and other values to 0
- Normalized the independent variables in the dataset to get better classification results

Dataset 2

- Converted Text columns including the dependent variable to numerical values
- Applied filters to convert unknown values to known values
- Dependent variable y contained ‘Yes’ and ‘No’ values so converted them to 1 and 0
- Normalized the independent variables in the dataset to get better classification results

Clustering

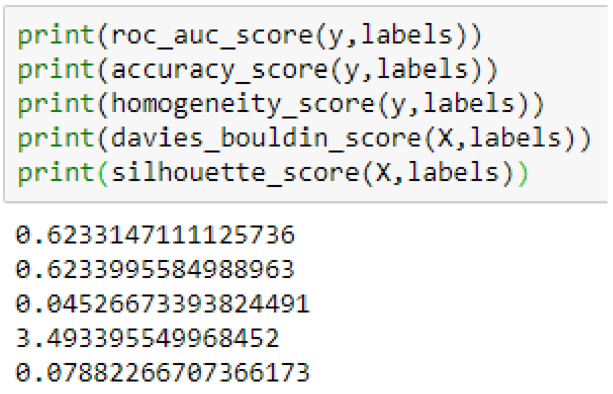
- Implemented K-Means and Expectation Maximization on both the Datasets.
- Obtained Different Clusters compared with the class labels when k=2
- Implemented Neural Networks by splitting the data into train test and using the cluster labels as the class labels (task 5)
- The clusters formed are less compact because of the categorical nature of the data

Dataset 1

K-Means

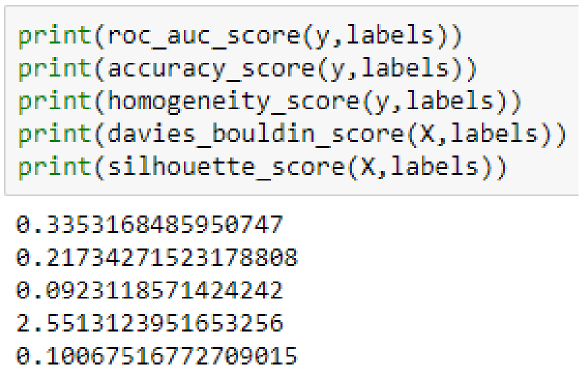
We applied K-Means with 2 clusters and compared the generated cluster labels with the actual labels(y) and found out that they are not the same and the clusters form naturally on their own.

This is a performance plot of 2 clusters vs 5 clusters:



```
print(roc_auc_score(y,labels))
print(accuracy_score(y,labels))
print(homogeneity_score(y,labels))
print(davies_bouldin_score(X,labels))
print(silhouette_score(X,labels))
```

```
0.6233147111125736
0.6233995584988963
0.04526673393824491
3.493395549968452
0.07882266707366173
```

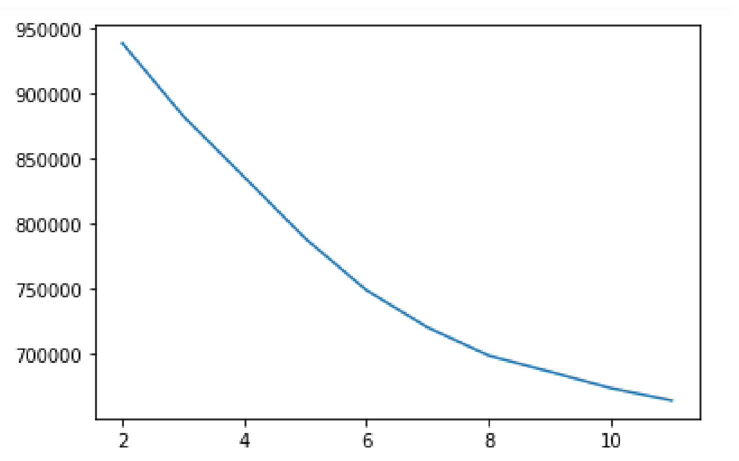


```
print(roc_auc_score(y,labels))
print(accuracy_score(y,labels))
print(homogeneity_score(y,labels))
print(davies_bouldin_score(X,labels))
print(silhouette_score(X,labels))
```

```
0.3353168485950747
0.21734271523178808
0.0923118571424242
2.5513123951653256
0.10067516772709015
```

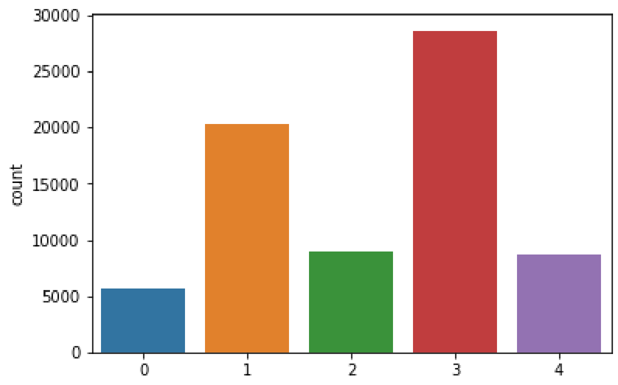
The Davies Bouldin Score(Intra Cluster distances/Inter Cluster Distances) and Silhouette Score are both better for 5 clusters.

Elbow Plot:



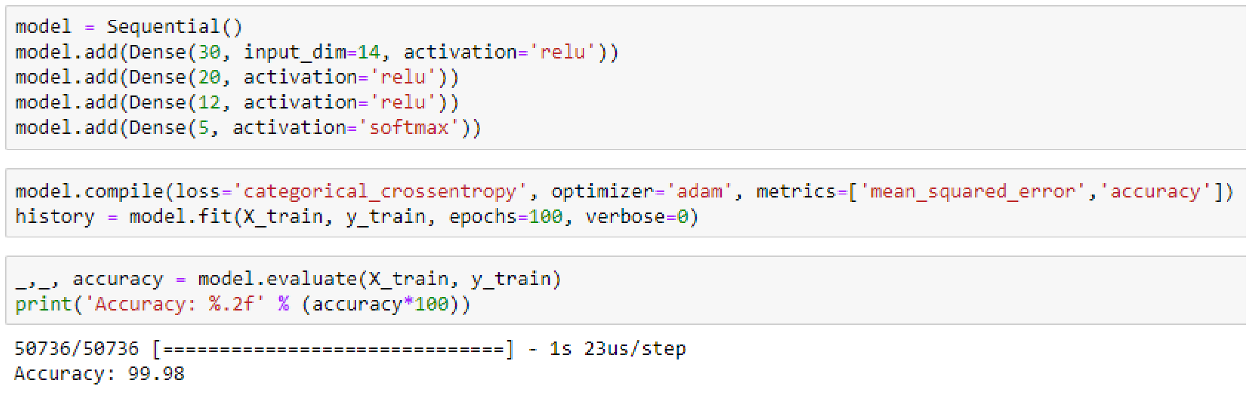
We decide how many clusters to choose based on the elbow plot. We can see the elbow at 5-6 clusters. After a few experiments 5 clusters were found to be performing the best.

Below is a count plot of the clusters:



Neural Networks with K-Means:

This shows the accuracy obtained when 5 clusters labels are passed as class labels to the neural networks.

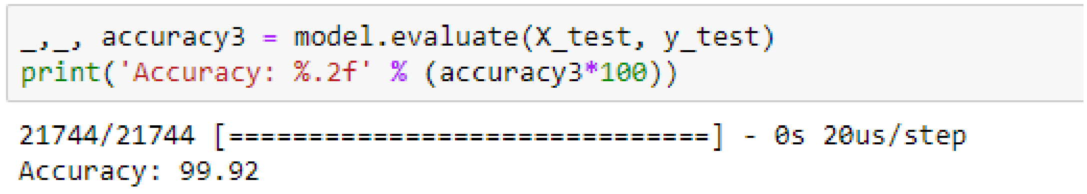


```
model = Sequential()
model.add(Dense(30, input_dim=14, activation='relu'))
model.add(Dense(20, activation='relu'))
model.add(Dense(12, activation='relu'))
model.add(Dense(5, activation='softmax'))
```

```
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['mean_squared_error','accuracy'])
history = model.fit(X_train, y_train, epochs=100, verbose=0)
```

```
_,_, accuracy = model.evaluate(X_train, y_train)
print('Accuracy: %.2f' % (accuracy*100))
```

```
50736/50736 [==============================] - 1s 23us/step
Accuracy: 99.98
```



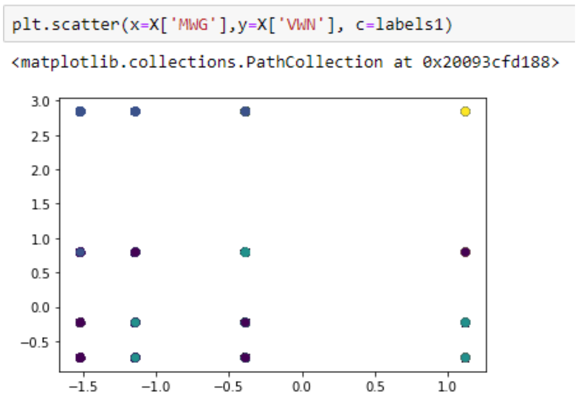
```
_,_, accuracy3 = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy3*100))
```

```
21744/21744 [==============================] - 0s 20us/step
Accuracy: 99.92
```

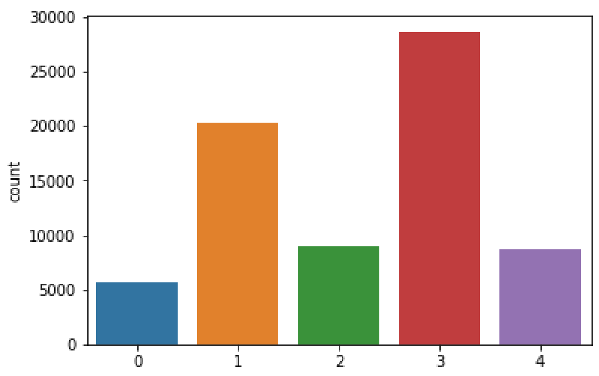
The test accuracy is 99.92% which is much better than the accuracy which was obtained by classification based on the class labels in the previous assignments.

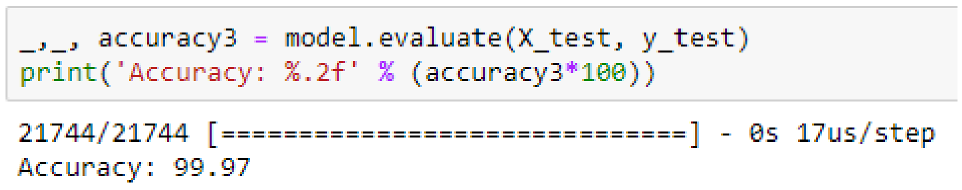
Expectation Maximization

Below on the left is a plot of 2 variables plotted again each other and the color represents the clusters. On the right is a plot of the cluster distribution. We can see the clusters are similar to those formed in K-Means. Expectation Maximization has been implemented for 5 clusters as it was performing the best Being a categorical data, the clusters are formed in the below way.



```
plt.scatter(x=X['MWG'],y=X['VWN'], c=labels1)
<matplotlib.collections.PathCollection at 0x20093cfd188>
```





```
_,_, accuracy3 = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy3*100))
```

```
21744/21744 [==============================] - 0s 17us/step
Accuracy: 99.97
```

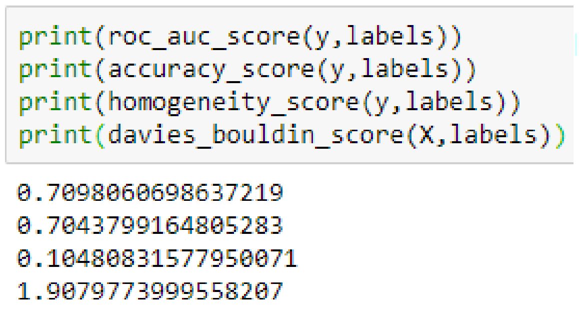
Above is the accuracy of neural networks when implemented on test data after training using the clustering labels obtained by Expectation Maximization as the class labels.

Dataset 2

K-Means

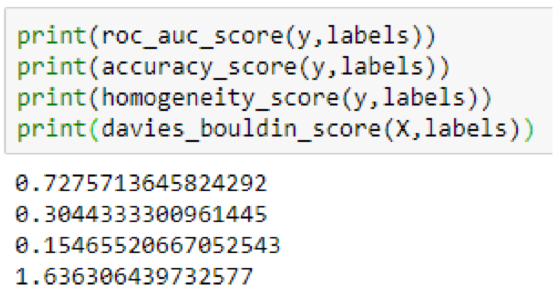
We applied K-Means with 2 clusters and compared the generated cluster labels with the actual labels(y) and found out that they are not the same and the clusters form naturally on their own based on the homogeneity score.

This is a performance plot of 2 clusters vs 7 clusters:



```
print(roc_auc_score(y,labels))
print(accuracy_score(y,labels))
print(homogeneity_score(y,labels))
print(davies_bouldin_score(X,labels))
```

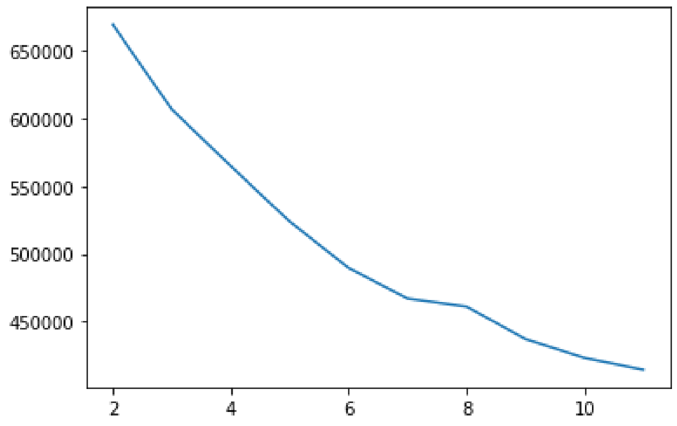
```
0.7098060698637219
0.7043799164805283
0.10480831577950071
1.9079773999558207
```



```
print(roc_auc_score(y,labels))
print(accuracy_score(y,labels))
print(homogeneity_score(y,labels))
print(davies_bouldin_score(X,labels))
```

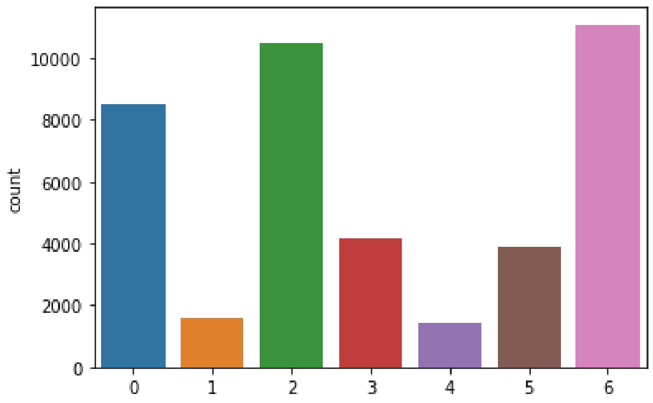
```
0.7275713645824292
0.3044333300961445
0.15465520667052543
1.636306439732577
```

Elbow Plot:



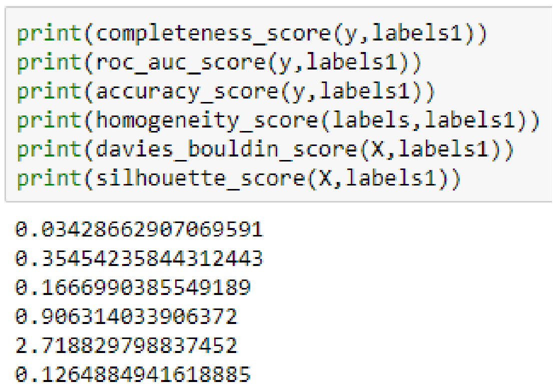
Using the results from the elbow plot we decide to proceed with 7 clusters.

Below is the countplot of the distribution of the 7 clusters.



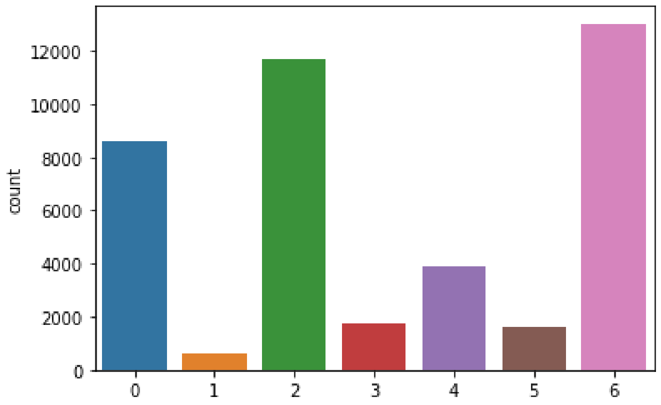
The neural networks implementation on this dataset after clustering gives a test accuracy of 99.79% which is slightly better than the KMeans accuracy.

Expectation Maximization

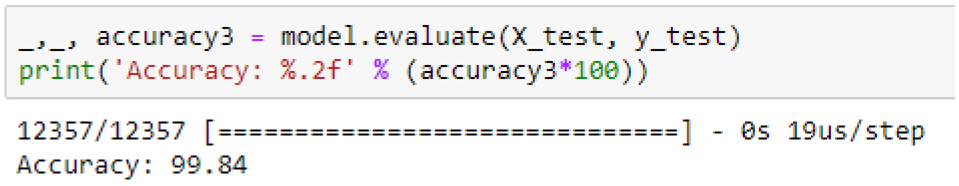


```
print(completeness_score(y,labels1))
print(roc_auc_score(y,labels1))
print(accuracy_score(y,labels1))
print(homogeneity_score(labels,labels1))
print(davies_bouldin_score(X,labels1))
print(silhouette_score(X,labels1))
```

```
0.03428662907069591
0.35454235844312443
0.1666990385549189
0.906314033906372
2.718829798837452
0.1264884941618885
```



The above plot shows the performance of the expectation maximization clustering algorithm and the distribution of the clusters.



```
_,_, accuracy3 = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy3*100))
```

```
12357/12357 [==============================] - 0s 19us/step
Accuracy: 99.84
```

Neural Networks when applied with the cluster labels gave an accuracy of 99.84 on test data.

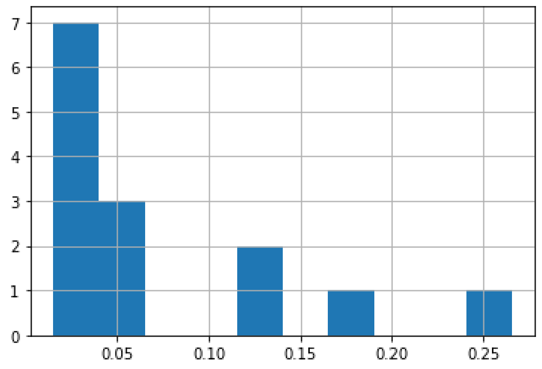
Feature Selection and Transformation

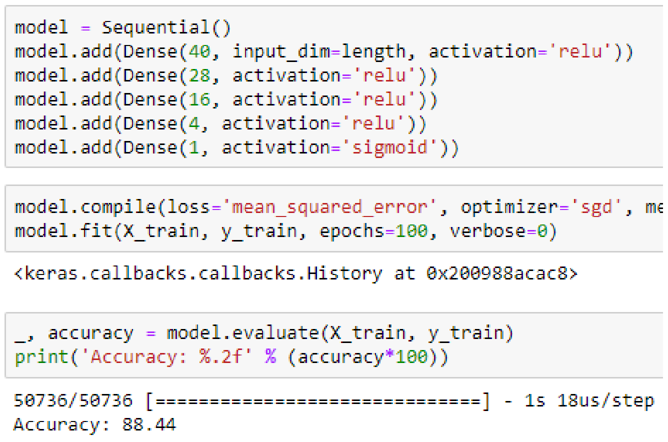
- We have applied random forest feature selection method on both datasets.
- PCA, ICA, Random Projections are the 3 feature transformation methods applied on both the datasets.
- We have applied clustering on the transformed datasets and then we have applied neural network algorithms to find the effect of these transformations on the datasets.
- All plots have not been shown in the report below but are available in the code execution.

Dataset 1

Decision Tree Feature Selection

Below plot shows the feature importance after selecting few important features using RandomForest algorithm using 100 tree estimators on the left and on the right is the neural networks model used to train the selected data.





```
model = Sequential()
model.add(Dense(40, input_dim=length, activation='relu'))
model.add(Dense(28, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='mean_squared_error', optimizer='sgd', me
model.fit(X_train, y_train, epochs=100, verbose=0)
```

```
<keras.callbacks.callbacks.History at 0x200988acac8>
```

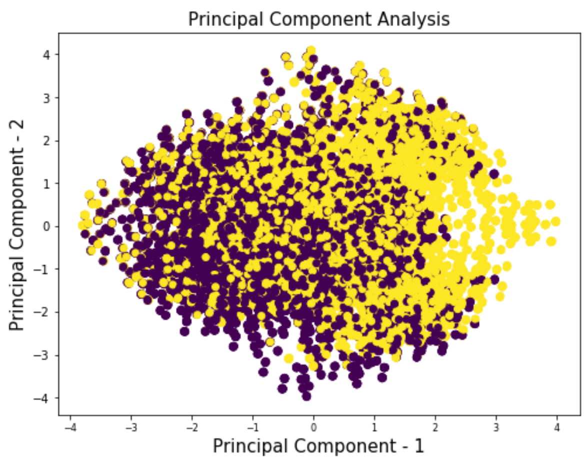
```
_, accuracy = model.evaluate(X_train, y_train)
print('Accuracy: %.2f' % (accuracy*100))
```

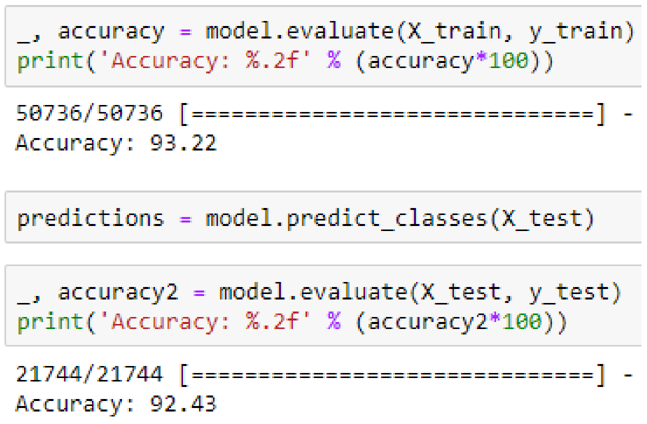
```
50736/50736 [==============================] - 1s 18us/step
Accuracy: 88.44
```

After implementing neural networks on the selected features, it gives 88% accuracy is less as compared to the score obtained by other models.

PCA (Principal Component Analysis)

Below plots show the first 2 features after being transformed using PCA and the color denotes the class labels. On the right the plot shows the training and testing accuracy of the neural networks model executed on the transformed and dimension reduced data.





```
_, accuracy = model.evaluate(X_train, y_train)
print('Accuracy: %.2f' % (accuracy*100))
```

```
50736/50736 [==============================] -
Accuracy: 93.22
```

```
predictions = model.predict_classes(X_test)
```

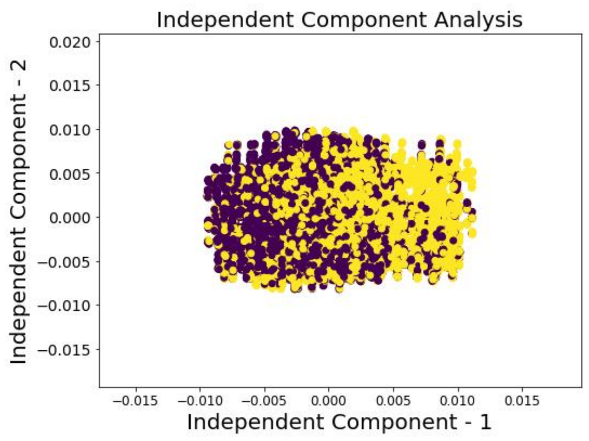
```
_, accuracy2 = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy2*100))
```

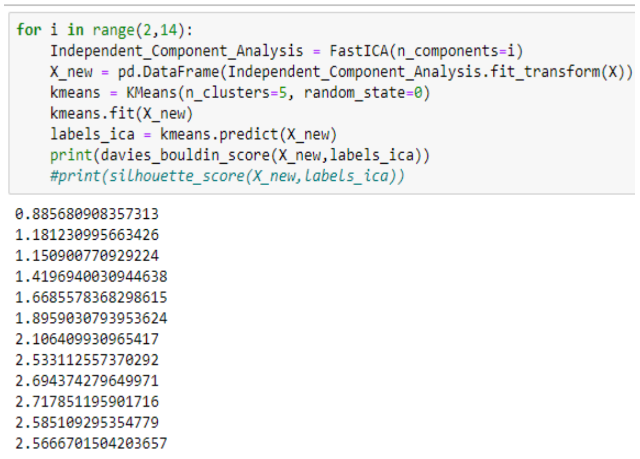
```
21744/21744 [==============================] -
Accuracy: 92.43
```

Clustering is also performed on the transformed data and the clusters formed are different and better than the previous clusters.

ICA (Independent Component Analysis)

The left plot shows the first 2 features plot against each other and color represented by the class labels. The right plot shows the method used to select the number of components.





```
for i in range(2,14):
    Independent_Component_Analysis = FastICA(n_components=i)
    X_new = pd.DataFrame(Independent_Component_Analysis.fit_transform(X))
    kmeans = KMeans(n_clusters=5, random_state=0)
    kmeans.fit(X_new)
    labels_ica = kmeans.predict(X_new)
    print(davies_bouldin_score(X_new,labels_ica))
    #print(silhouette_score(X_new,labels_ica))
```

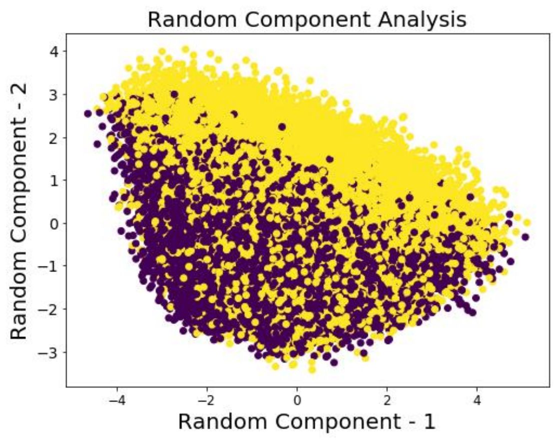
```
0.885680908357313
1.181230995663426
1.150900770929224
1.4196940030944638
1.6685578368298615
1.8959030793953624
2.106409930965417
2.533112557370292
2.694374279649971
2.717851195901716
2.585109295354779
2.5666701504203657
```

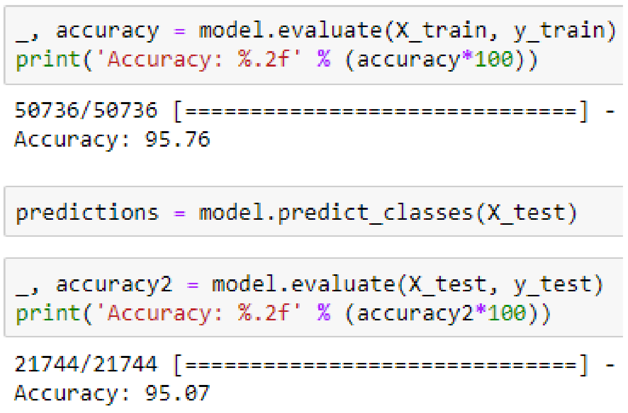
Clustering has been used in the right plot to find out when the cluster performs fairly to select the number of components.

Neural Networks gave an accuracy of 81% on the test data after providing the transformed data as the input.

Randomized Projections

Left plot shows the first and second features and color shows the labels. The right plot shows the Neural Networks execution accuracy which is 95.07% on the test data.





```
_, accuracy = model.evaluate(X_train, y_train)
print('Accuracy: %.2f' % (accuracy*100))
```

```
50736/50736 [==============================] -
Accuracy: 95.76
```

```
predictions = model.predict_classes(X_test)
```

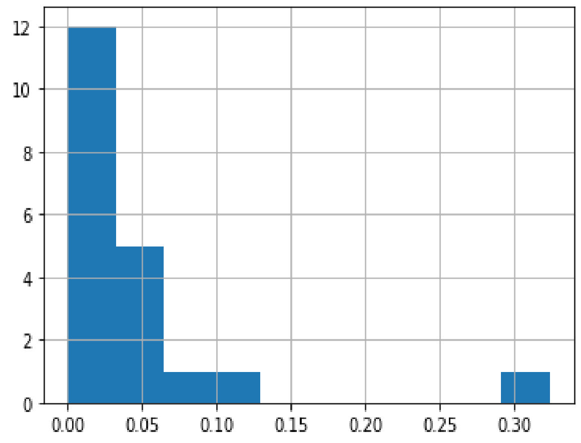
```
_, accuracy2 = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy2*100))
```

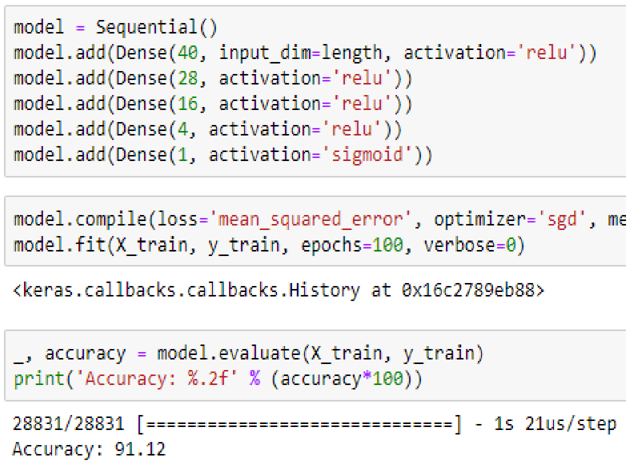
```
21744/21744 [==============================] -
Accuracy: 95.07
```

Dataset 2

Decision Tree Feature Selection

The plot on left shows the feature importance plot and the plot on the right shows the accuracy given by the Neural Networks implemented on the train data.





```
model = Sequential()
model.add(Dense(40, input_dim=length, activation='relu'))
model.add(Dense(28, activation='relu'))
model.add(Dense(16, activation='relu'))
model.add(Dense(4, activation='relu'))
model.add(Dense(1, activation='sigmoid'))
```

```
model.compile(loss='mean_squared_error', optimizer='sgd', me
model.fit(X_train, y_train, epochs=100, verbose=0)
```

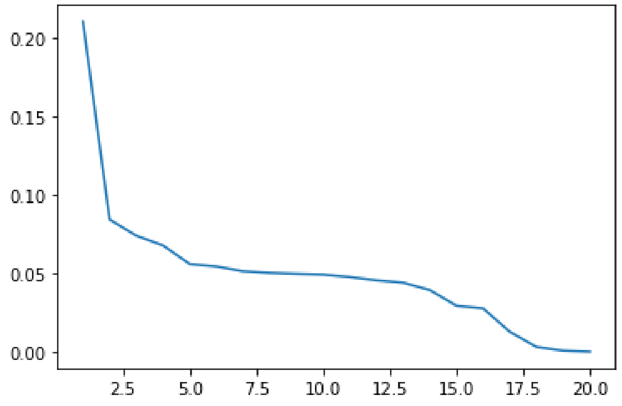
```
<keras.callbacks.callbacks.History at 0x16c2789eb88>
```

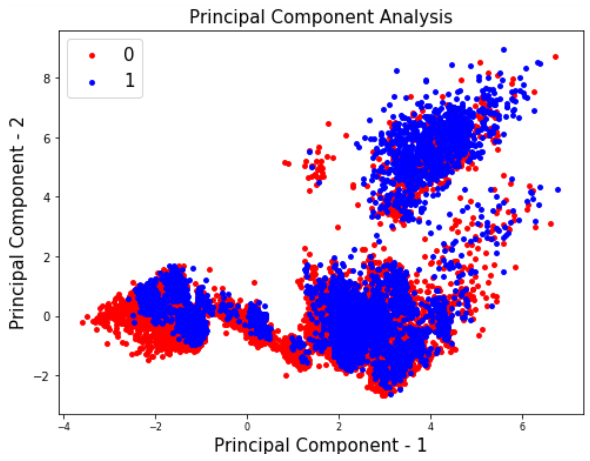
```
_, accuracy = model.evaluate(X_train, y_train)
print('Accuracy: %.2f' % (accuracy*100))
```

```
28831/28831 [==============================] - 1s 21us/step
Accuracy: 91.12
```

PCA

Below plots show the first 2 features after being transformed using PCA and the color denotes the class labels. On the left the plot shows the variance plot for all the features in the dataset with the first two features having high variances and it goes decreasing towards the end.





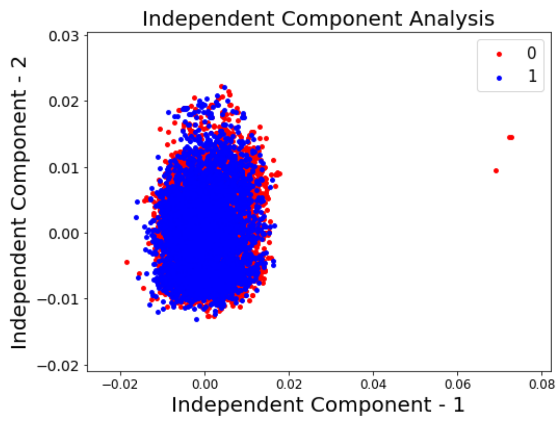
```
_, accuracy2 = model.evaluate(X_test, y_test)
print('Accuracy: %.2f' % (accuracy2*100))
```

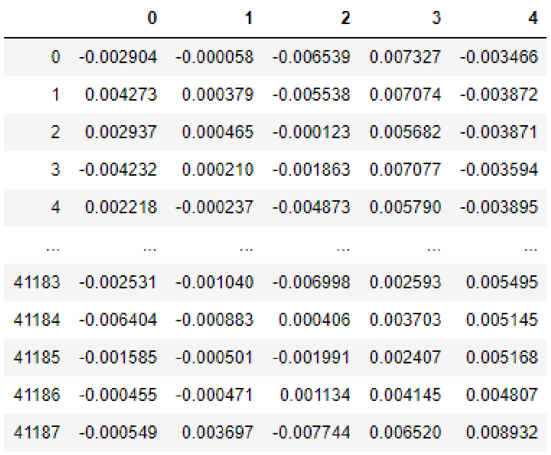
```
12357/12357 [==============================] - 0s 22us/step
Accuracy: 90.52
```

The above plot shows the execution of Neural Network on the data transformed by PCA which gives accuracy of 90.52%.

ICA (Independent Component Analysis)

The left plot shows first and second feature and label as color. The right plot shows a snippet of the obtained data after applying the ICA transformation.

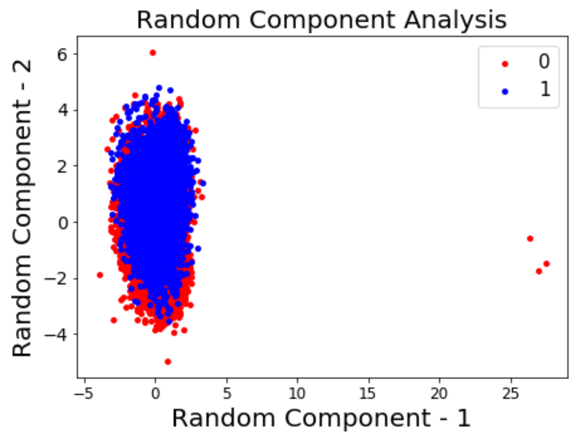


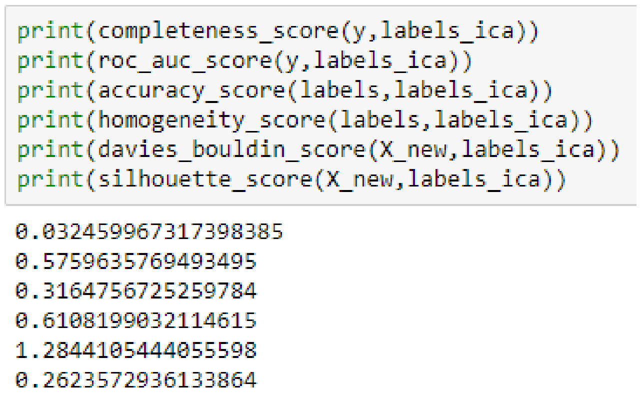


	0	1	2	3	4
0	-0.002904	-0.000058	-0.006539	0.007327	-0.003466
1	0.004273	0.000379	-0.005538	0.007074	-0.003872
2	0.002937	0.000465	-0.000123	0.005682	-0.003871
3	-0.004232	0.000210	-0.001863	0.007077	-0.003594
4	0.002218	-0.000237	-0.004873	0.005790	-0.003895
...	...	...	...	...	...
41183	-0.002531	-0.001040	-0.006998	0.002593	0.005495
41184	-0.006404	-0.000883	0.000406	0.003703	0.005145
41185	-0.001585	-0.000501	-0.001991	0.002407	0.005168
41186	-0.000455	-0.000471	0.001134	0.004145	0.004807
41187	-0.000549	0.003697	-0.007744	0.006520	0.008932

Randomized Projections

The plot on the left shows the first Random Component vs the second Random Component. The two colors red and blue denote the points with class 0 and 1 respectively. The plot on the right shows the clustering scores obtained on the data transformed by Randomized Projections.





```
print(completeness_score(y,labels_ica))
print(roc_auc_score(y,labels_ica))
print(accuracy_score(labels,labels_ica))
print(homogeneity_score(labels,labels_ica))
print(davies_bouldin_score(X_new,labels_ica))
print(silhouette_score(X_new,labels_ica))
```

```
0.032459967317398385
0.5759635769493495
0.316475672525978 4
0.6108199032114615
1.284410544405559 8
0.2623572936133864
```

Conclusions

- Neural Networks performed better with Randomized Projections for the Dataset 1 while it performed equally well for all the three methods for Dataset 2.
- The performance of the neural networks was best when it was provided with the cluster labels as the class labels which gave a near 100 accuracy for both the Datasets. It was a huge performance jump for the dataset 2 which gave 90-91% accuracy with the class labels (assignment3)
- The Feature Selection method of Decision Trees performed well for the second dataset and average for the first Dataset.
- PCA gave an average performance for both the datasets and the major reason for that being not getting high variances for any variable. Hence, PCA is very helpful if there are small number of features with very high variances. Still PCA performed fairly well with the Dataset 2 where there were few features with high variances.
- For ICA and Random Projections, the number of features to be selected from the total features was done based on experimentation. ICA performed much better on Dataset 2 as compared to its performance for Dataset 1. I have provided a case where neural networks did good on Dataset 2 while did poorly on Dataset 1 after implementing ICA for feature Transformation.
- Clustering Performed better on dataset 2 after PCA and ICA (Performance of cluster measured by Davies Bouldin Score and Silhouette Score. Similarity between the clustering labels of 2 different algorithms has been calculated using Homogeneity Score).
- The Data generated is completely different from the actual data in the transformed domain. ICA transforms the data into very small values approximately 10^-3 times the original data. PCA transforms the data such that the data in the first column would have the maximum variance and then followed the second best and so on. The data transformed by Randomized Projections looks similar to the original values but here the rows are orthogonal to each other.