Exploration of the Satander Dataset

First, we start with LDA and then we go to PCA

Note: The number of components allowed in LDA ranges from 1 to n-1, where n is the number of target class

Note: The number of components allowed in PCA ranges from 1 to n-1, where n is the number of features

We will be using the Santander datatset

In [1]:

```
# We import dependencies, data manipulation and visualization

%matplotlib inline
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

In [21]:

```
from sklearn.feature_selection import VarianceThreshold
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.decomposition import PCA
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
from sklearn.preprocessing import StandardScaler
from sklearn.ensemble import RandomForestClassifier
```

We Import our Data

In [3]:

```
data = pd.read_csv('C:/Users/dell/Desktop/santander.csv', nrows=20000)
data.shape
```

Out[3]:

```
(20000, 371)
```

In [4]:

```
data.head()
```

Out[4]:

	ID	var3	var15	imp_ent_var16_ult1	imp_op_var39_comer_ult1	imp_op_var39_comer_ult3	imp_op_var40_comer_ult1	imp_op_var40_c
0	1	2	23	0.0	0.0	0.0	0.0	
1	3	2	34	0.0	0.0	0.0	0.0	
2	4	2	23	0.0	0.0	0.0	0.0	
3	8	2	37	0.0	195.0	195.0	0.0	
4	10	2	39	0.0	0.0	0.0	0.0	

5 rows × 371 columns

In [5]:

```
X = data.drop('TARGET', axis=1)
y = data.TARGET
X.shape, y.shape
```

Out[5]:

```
((20000, 370), (20000,))
```

In [6]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, stratify=y, test_size=0.2, random_state=0)
```

We Remove the Constant, Quasi-Constant and Duplicate Features

In [9]:

```
# We remove constant and quasi constant features

constant_filter = VarianceThreshold(threshold=0.01)
constant_filter.fit(X_train)
X_train_filter = constant_filter.transform(X_train)
X_test_filter = constant_filter.transform(X_test)
```

In [10]:

```
X_train_filter.shape, X_train.shape
```

Out[10]:

```
((16000, 245), (16000, 370))
```

In [12]:

```
# We remove duplicate features

X_train_T = X_train_filter.T
X_test_T = X_test_filter.T
```

In [13]:

```
X_train_T = pd.DataFrame(X_train_T)
X_test_T = pd.DataFrame(X_test_T)
```

In [14]:

```
duplicated_features = X_train_T.duplicated()
duplicated_features
```

Out[14]:

```
0      False
1      False
2      False
3      False
4      False
       ...
240    False
241    False
242    False
243    False
244    False
Length: 245, dtype: bool
```

In [16]:

```
features_to_keep = [not i for i in duplicated_features]

X_train_unique = X_train_T[features_to_keep].T
X_test_unique = X_test_T[features_to_keep].T
```

We Scale our Data

In [35]:

```
scaler = StandardScaler().fit(X_train_unique)
X_train_scaled = scaler.transform(X_train_unique)
X_test_scaled = scaler.transform(X_test_unique)      # We do not need to fit for the test data
```

In [36]:

```
X_train_scaled = pd.DataFrame(X_train_scaled)
X_test_scaled = pd.DataFrame(X_test_scaled)
X_train_scaled.shape, X_test_scaled.shape
```

Out[36]:

```
((16000, 227), (4000, 227))
```

We Remove Correlated Features

In [37]:

```
corrmatrix = X_train_scaled.corr()
corrmatrix
```

Out[37]:

	0	1	2	3	4	5	6	7	8	9	...	217	218
0	1.000000	-0.025277	-0.001942	0.003594	0.004054	-0.001697	-0.015882	-0.019807	0.000956	-0.000588	...	-0.001337	0.002051
1	-0.025277	1.000000	-0.007647	0.001819	0.008981	0.009232	0.001638	0.001746	0.000614	0.000695	...	0.000544	0.000586
2	-0.001942	-0.007647	1.000000	0.030919	0.106245	0.109140	0.048524	0.055708	0.004040	0.005796	...	0.025522	0.020168
3	0.003594	0.001819	0.030919	1.000000	0.029418	0.024905	0.014513	0.013857	-0.000613	-0.000691	...	0.014032	-0.000583
4	0.004054	0.008981	0.106245	0.029418	1.000000	0.888789	0.381632	0.341266	0.012927	0.019674	...	0.002328	0.016743
...	...	...	...	...	...	...	...	...	...	...	...	...	...
222	0.008825	0.000922	0.041321	0.000541	-0.001905	0.000871	-0.000818	-0.000866	-0.000309	-0.000349	...	0.012705	0.021540
223	-0.009174	0.000598	0.016172	-0.000577	-0.000635	0.007096	-0.000515	-0.000545	-0.000195	-0.000220	...	-0.000173	-0.000185
224	0.012031	0.000875	0.043577	0.000231	-0.002552	-0.001672	-0.000779	-0.000825	-0.000295	-0.000332	...	0.027515	0.012393
225	0.012128	0.000942	0.044281	0.000235	-0.002736	-0.001844	-0.000839	-0.000888	-0.000317	-0.000358	...	0.023072	0.014523
226	0.006612	0.000415	-0.000810	0.000966	0.003656	0.002257	0.004448	0.002427	-0.000739	-0.000811	...	-0.003399	-0.000773

227 rows × 227 columns

In [38]:

```
# We will extract all the features with a high correlation

def get_corr_features(df, threshold):
    corr_features = set()
    corrmatrix = df.corr()
    for i in range(len(corrmatrix.columns)):
        for j in range(i):
            if abs(corrmatrix.iloc[i, j])>threshold:
                colname = corrmatrix.columns[i]
                corr_features.add(colname)
    return corr_features
```

In [39]:

```
corr_features = get_corr_features(X_train_unique, 0.7)
print(corr_features)
```

```
{5, 7, 9, 10, 11, 12, 14, 15, 16, 17, 18, 23, 24, 26, 28, 29, 30, 32, 33, 34, 35, 36, 38, 40, 42, 46
, 47, 50, 51, 52, 53, 54, 55, 56, 57, 58, 60, 61, 62, 65, 67, 68, 69, 70, 72, 75, 76, 79, 80, 81, 82
, 83, 84, 85, 86, 87, 88, 91, 93, 95, 98, 100, 101, 103, 104, 109, 110, 111, 115, 117, 119, 120, 121
, 125, 136, 138, 143, 145, 146, 149, 153, 154, 157, 158, 161, 162, 163, 164, 167, 168, 169, 170, 171
, 173, 174, 179, 180, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 197, 198
, 199, 204, 205, 206, 207, 208, 210, 211, 212, 213, 215, 216, 217, 219, 220, 221, 223, 224, 225, 227
, 228, 229, 230, 231, 232, 234, 235, 236, 237, 238, 239, 240, 241, 242, 243}
```

In [40]:

```
len(corr_features)
```

Out[40]:

148

In [41]:

```
# We remove the highly correlated features from X_train_unique

X_train_uncorr = X_train_unique.drop(labels=corr_features, axis=1)
X_test_uncorr = X_test_unique.drop(labels=corr_features, axis=1)
X_train_uncorr.shape, X_train.shape
```

Out[41]:

((16000, 79), (16000, 370))

Feature Reduction by LDA

In [50]:

```
lda = LDA(n_components=1)     # The number of components ranges from 1 to n-1. Note that n=2 (i.e. it is a biclas
s)
lda.fit(X_train_uncorr, y_train)
X_train_lda = lda.transform(X_train_uncorr)
X_test_lda = lda.transform(X_test_uncorr)   # We do not need to fit for test data
```

In [51]:

```
def run_model(X_train, X_test, y_train, y_test):
    rf = RandomForestClassifier(n_estimators=100, random_state=0, n_jobs=-1)
    rf.fit(X_train, y_train)
    y_pred = rf.predict(X_test)
    print('Accuracy score:', accuracy_score(y_test, y_pred))
```

In [52]:

```
print('---Data with All Features')
run_model(X_train, X_test, y_train, y_test)
```

```
---Data with All Features
Accuracy score: 0.9585
```

In [53]:

```
print('---Data with LDA---')
run_model(X_train_lda, X_test_lda, y_train, y_test)
```

```
---Data with LDA---
Accuracy score: 0.93025
```

In [54]:

```
X_train_lda.shape, X_train.shape
```

Out[54]:

```
((16000, 1), (16000, 370))
```

Feature Reduction by PCA

In [55]:

```
pca = PCA(n_components=2)     # The number of components ranges from 1 to n-1. Note that n is nos of features (i.
e. 79)
pca.fit(X_train_uncorr, y_train)
X_train_pca = pca.transform(X_train_uncorr)
X_test_pca = pca.transform(X_test_uncorr)   # We do not need to fit for test data
```

In [56]:

```
print('---Data with PCA---')
run_model(X_train_pca, X_test_pca, y_train, y_test)
```

```
---Data with PCA---
Accuracy score: 0.95925
```

In [57]:

```
for k in range(1, 79):
    pca = PCA(n_components=k)     # The nos of comp ranges from 1 to n-1. Note that n is nos of features (i.e. 79
)
    pca.fit(X_train_uncorr, y_train)
    X_train_pca = pca.transform(X_train_uncorr)
    X_test_pca = pca.transform(X_test_uncorr)
    print('Number of features:', k)
    run_model(X_train_pca, X_test_pca, y_train, y_test)
    print()
```

```
Number of features: 1
Accuracy score: 0.95925

Number of features: 2
Accuracy score: 0.95925

Number of features: 3
Accuracy score: 0.95925

Number of features: 4
Accuracy score: 0.959
```

```
Number of features: 5
Accuracy score: 0.959

Number of features: 6
Accuracy score: 0.92125

Number of features: 7
Accuracy score: 0.92225

Number of features: 8
Accuracy score: 0.92425

Number of features: 9
Accuracy score: 0.95425

Number of features: 10
Accuracy score: 0.956

Number of features: 11
Accuracy score: 0.957

Number of features: 12
Accuracy score: 0.9565

Number of features: 13
Accuracy score: 0.95675

Number of features: 14
Accuracy score: 0.95675

Number of features: 15
Accuracy score: 0.95675

Number of features: 16
Accuracy score: 0.95725

Number of features: 17
Accuracy score: 0.9565

Number of features: 18
Accuracy score: 0.95775

Number of features: 19
Accuracy score: 0.95675

Number of features: 20
Accuracy score: 0.9575

Number of features: 21
Accuracy score: 0.95725

Number of features: 22
Accuracy score: 0.95775

Number of features: 23
Accuracy score: 0.95775

Number of features: 24
Accuracy score: 0.9575

Number of features: 25
Accuracy score: 0.95725

Number of features: 26
Accuracy score: 0.95725

Number of features: 27
Accuracy score: 0.95725

Number of features: 28
Accuracy score: 0.9575

Number of features: 29
Accuracy score: 0.95775

Number of features: 30
Accuracy score: 0.95725

Number of features: 31
Accuracy score: 0.957

Number of features: 32
Accuracy score: 0.95775
```

```
Number of features: 33
Accuracy score: 0.95775

Number of features: 34
Accuracy score: 0.958

Number of features: 35
Accuracy score: 0.9575

Number of features: 36
Accuracy score: 0.958

Number of features: 37
Accuracy score: 0.958

Number of features: 38
Accuracy score: 0.95775

Number of features: 39
Accuracy score: 0.9575

Number of features: 40
Accuracy score: 0.9575

Number of features: 41
Accuracy score: 0.95825

Number of features: 42
Accuracy score: 0.95775

Number of features: 43
Accuracy score: 0.958

Number of features: 44
Accuracy score: 0.95725

Number of features: 45
Accuracy score: 0.95725

Number of features: 46
Accuracy score: 0.9575

Number of features: 47
Accuracy score: 0.95775

Number of features: 48
Accuracy score: 0.95775

Number of features: 49
Accuracy score: 0.957

Number of features: 50
Accuracy score: 0.958

Number of features: 51
Accuracy score: 0.958

Number of features: 52
Accuracy score: 0.95775

Number of features: 53
Accuracy score: 0.958

Number of features: 54
Accuracy score: 0.95725

Number of features: 55
Accuracy score: 0.958

Number of features: 56
Accuracy score: 0.9575

Number of features: 57
Accuracy score: 0.95725

Number of features: 58
Accuracy score: 0.958

Number of features: 59
Accuracy score: 0.95725

Number of features: 60
```

```
Accuracy score: 0.95675

Number of features: 61
Accuracy score: 0.9575

Number of features: 62
Accuracy score: 0.95825

Number of features: 63
Accuracy score: 0.958

Number of features: 64
Accuracy score: 0.95825

Number of features: 65
Accuracy score: 0.95725

Number of features: 66
Accuracy score: 0.957

Number of features: 67
Accuracy score: 0.9575

Number of features: 68
Accuracy score: 0.958

Number of features: 69
Accuracy score: 0.9575

Number of features: 70
Accuracy score: 0.958

Number of features: 71
Accuracy score: 0.957

Number of features: 72
Accuracy score: 0.95775

Number of features: 73
Accuracy score: 0.95825

Number of features: 74
Accuracy score: 0.95725

Number of features: 75
Accuracy score: 0.9575

Number of features: 76
Accuracy score: 0.957

Number of features: 77
Accuracy score: 0.9575

Number of features: 78
Accuracy score: 0.9575
```

In [ ]:

```
```

In [ ]:

```
```