PCA v/s LDA

Riya Goyal 2148054

Importing libraries

In [1]:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import math
from sklearn.model_selection import train_test_split
```

Data Description:

- The dataset contains transactions made by credit cards in September 2013 by European cardholders.
- This dataset presents transactions that occurred in two days, where we have 492 frauds out of 284,807 transactions. The dataset is highly unbalanced, the positive class (frauds) account for 0.172% of all transactions.
- Feature **'Time'** contains the seconds elapsed between each transaction and the first transaction in the dataset.
- Feature **'Amount'** is the transaction Amount, this feature can be used for example-dependant cost-sensitive learning.
- Feature **'Class'** is the response variable and it takes value 1 in case of fraud and 0 otherwise.

In [3]:

```
# importing the dataset
df = pd.read_csv('D:/Downloads/creditcard.csv/creditcard.csv')

# displaying the first 5 rows of the dataset
df.head()
```

Out[3]:

	Time	V1	V2	V3	V4	V5	V6	V7	V8
0	0.0	-1.359807	-0.072781	2.536347	1.378155	-0.338321	0.462388	0.239599	0.098698
1	0.0	1.191857	0.266151	0.166480	0.448154	0.060018	-0.082361	-0.078803	0.085102
2	1.0	-1.358354	-1.340163	1.773209	0.379780	-0.503198	1.800499	0.791461	0.247676
3	1.0	-0.966272	-0.185226	1.792993	-0.863291	-0.010309	1.247203	0.237609	0.377436
4	2.0	-1.158233	0.877737	1.548718	0.403034	-0.407193	0.095921	0.592941	-0.270533

5 rows × 31 columns

In [4]:

```
df.shape
```

Out[4]:

```
(284807, 31)
```

Dropping unnecessary columns, which contains null values

In [5]:

```
# columns containing null values
df.columns[df.isnull().any()]
```

Out[5]:

```
Index([], dtype='object')
```

There is no missing data in the dataset

In [6]:

```
df['Class'].value_counts()
```

Out[6]:

```
0    284315
1       492
Name: Class, dtype: int64
```

There is strong imbalance in the target value counts. To solve this issue, we will oversample the dataset.

Oversampling the dataset

In [7]:

```python
def _oversample_positives(df, target):
    """ Oversample the minority classes to match
    the majority class.

    :param df: pandas dataframe - input df.
    :param target: string - classification target column.

    :return: pandas datframe - oversampled version
    """

    class_count = df[target].value_counts()

    print("Before oversampling: %s" % class_count)

    for i in range(1,len(class_count)):
        df_i = df[df[target] == i]
        oversampling_factor_i = class_count[0] / float(class_count[i])
        print(len(df_i))
        print("Oversampling factor for class %i: %s" %(i, str(oversampling_factor_i)))

        # Integer part of oversampling
        df = df.append(
            [df_i] * int(math.floor(oversampling_factor_i) - 1),
            ignore_index=False)

        # Float part of oversampling
        df = df.append(
            [df_i.sample(frac=oversampling_factor_i % 1)],
            ignore_index=False)

    print("After oversampling: %s" % df[target].value_counts())
    print("Shape after oversampling: %s" % str(df.shape))

    return df
```

In [8]:

```python
df_oversampled = _oversample_positives(df, 'Class')
```

```
Before oversampling: 0    284315
1       492
Name: Class, dtype: int64
492
Oversampling factor for class 1: 577.8760162601626
After oversampling: 0    284315
1    284315
Name: Class, dtype: int64
Shape after oversampling: (568630, 31)
```

Splitting the dataset into Independent and Dependent variables

In [9]:

```
X = df_oversampled.drop(['Class'],axis = 1)
y = df_oversampled['Class']
```

In [10]:

```
y.value_counts()
```

Out[10]:

```
0    284315
1    284315
Name: Class, dtype: int64
```

Training data and testing data

In [11]:

```
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,random_state=1)
```

Standardizing the data

In [12]:

```
from sklearn.preprocessing import StandardScaler
sc = StandardScaler()
X_train = sc.fit_transform(X_train)
X_test = sc.transform(X_test)
```

Linear Discriminant Analysis

Linear Discriminant Analysis as its name suggests is a linear model for classification and dimensionality reduction. Most commonly used for feature extraction in pattern classification problems.

Assumptions:

LDA makes some assumptions about the data:

- Assumes the data to be distributed normally or Gaussian distribution of data points i.e. each feature must make a bell-shaped curve when plotted.
- Each of the classes has identical covariance matrices.

However, it is worth mentioning that LDA performs quite well even if the assumptions are violated.

In [13]:

```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis as LDA
lda = LDA(n_components=1) #n_components to be less than the n_classes count
X_train = lda.fit_transform(X_train, y_train)
X_test = lda.transform(X_test)
LDA_df = pd.DataFrame(X_test)
LDA_df.head(10)
```

Out[13]:

	0
0	1.756222
1	1.403838
2	-0.690059
3	-2.580363
4	1.359297
5	-0.805975
6	0.152504
7	0.526020
8	-0.153876
9	0.276160

Modelling - Logistic Regression

In [14]:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
y_pred2 = classifier.predict(X_train)
```

Model Evaluation

In [15]:

```
from sklearn.metrics import confusion_matrix,accuracy_score

cm = confusion_matrix(y_test, y_pred)

print(cm)
print('Accuracy: ' + str(accuracy_score(y_test, y_pred)))
```

```
[[54632  2294]
 [ 5739 51061]]
Accuracy: 0.9293653166382357
```

Principal Component Analysis

Principal component analysis (PCA) is a technique for reducing the dimensionality of such datasets, increasing interpretability but at the same time minimizing information loss. It does so by creating new uncorrelated variables that successively maximize variance.

In [16]:

```
# Importing standardscalar module
from sklearn.preprocessing import StandardScaler
df = pd.DataFrame(X)
y_target = pd.DataFrame(y)
scalar = StandardScaler()

# fitting
scalar.fit(df)
scaled_data = scalar.transform(df)

# Importing PCA
from sklearn.decomposition import PCA

# Let's say, components = 2
pca = PCA(n_components = 0.95)
pca.fit(scaled_data)
x_pca = pca.transform(scaled_data)
print(x_pca.shape)
PCA_df = pd.DataFrame(x_pca)
PCA_df.head()
```

```
(568630, 17)
```

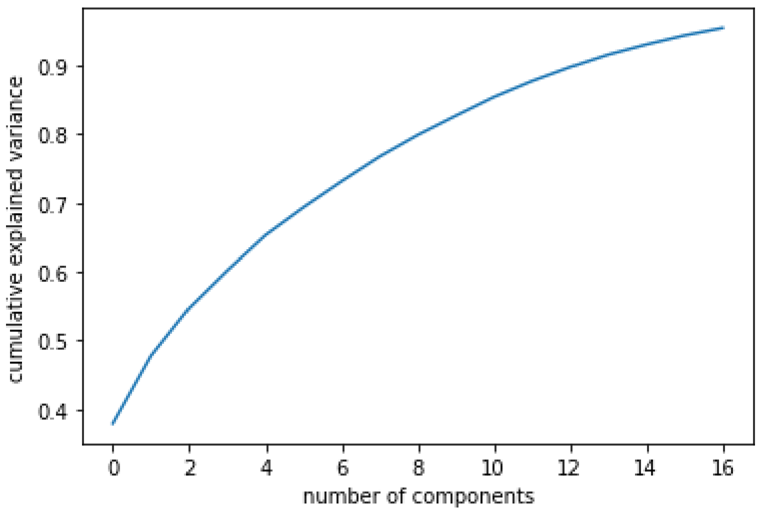
Out[16]:

	0	1	2	3	4	5	6	7	
0	-2.017367	-0.079548	0.270467	0.674413	-0.385413	1.409607	-1.026681	0.507275	-1.8059
1	-1.938590	-0.518897	0.009026	-0.092390	-0.703586	0.711123	-1.065528	-0.913804	-1.0238
2	-2.099515	0.313603	0.095663	0.335357	0.099324	2.515473	-0.768505	-0.864811	-1.5881
3	-2.340741	-0.038775	-0.452240	-1.047377	-1.110085	1.074225	0.101366	-1.381797	-0.7284
4	-1.898929	-0.053807	0.789989	0.434038	-0.245655	0.886225	-1.582754	-0.982458	0.1321

17 variables justify 95% variability in my data

In [22]:

```
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlabel('number of components')
plt.ylabel('cumulative explained variance');
```



Training data and testing data

In [18]:

```
X1_train, X1_test, y1_train, y1_test = train_test_split(PCA_df, y, test_size=0.2,rando
```

Logistic Regression Model

In [19]:

```
from sklearn.linear_model import LogisticRegression
classifier = LogisticRegression(random_state=0)
classifier.fit(X1_train, y1_train)
y1_pred = classifier.predict(X1_test)
y1_pred2 = classifier.predict(X1_train)
```

Model Evaluation

In [20]:

```
from sklearn.metrics import confusion_matrix,accuracy_score

cm1 = confusion_matrix(y1_test, y1_pred)

print(cm1)
print('Accuracy: ' + str(accuracy_score(y1_test, y1_pred)))
```

```
[[55549  1377]
 [ 5479 51321]]
Accuracy: 0.9397147530028314
```

Inference:

LDA & PCA both performed similarly on the datatset contributing to the model accuracy which was observed to be **92.93%** and **93.96%** respectively. The **information contained within 60 features was retained in 17 features approximately** and we did not lose much of information as we saw good accuracy.

In [ ]:

```

```