

EDA + Logistic Regression + PCA

Table of Contents

The contents of this kernel is divided into various topics which are as follows:-

- The Curse of Dimensionality
- Introduction to Principal Component Analysis
- Import Python libraries
- Import dataset
- Exploratory data analysis
- Split data into training and test set
- Feature engineering
- Feature scaling
- Logistic regression model with all features
- Logistic Regression with PCA
- Select right number of dimensions
- Plot explained variance ratio with number of dimensions
- Conclusion
- References

Import python Libraries

```
In [1]: ▶ import numpy as np  
import pandas as pd
```

import libraries for plotting

```
In [2]: ▶ import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

Ignore warning

```
In [3]: ▶ import warnings
warnings.filterwarnings("ignore")
```

import dataset

```
In [4]: ▶ df = pd.read_csv(r'E:\NIT(Data Science)by (prakash senapati sir (kodi))\Da
◀ [Progress Bar] ▶
```

Exploratory Data Analysis

check the dataset

```
In [5]: ▶ df.shape
```

Out[5]: (32561, 15)

there are 32561 instances and 15 attribute in the dataset

preview data

In [6]: `df.head()`

Out[6]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationsh
0	90	?	77053	HS-grad	9	Widowed	?	Not-in-far
1	82	Private	132870	HS-grad	9	Widowed	Exec-manage	Not-in-far
2	66	?	186061	Some-college	10	Widowed	?	Unmarri
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarri
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-ch

View summary of dataframe

In [7]: `df.info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype  
---  -
0   age                    32561 non-null  int64  
1   workclass              32561 non-null  object  
2   fnlwgt                 32561 non-null  int64  
3   education              32561 non-null  object  
4   education.num          32561 non-null  int64  
5   marital.status         32561 non-null  object  
6   occupation             32561 non-null  object  
7   relationship           32561 non-null  object  
8   race                   32561 non-null  object  
9   sex                    32561 non-null  object  
10  capital.gain           32561 non-null  int64  
11  capital.loss           32561 non-null  int64  
12  hours.per.week         32561 non-null  int64  
13  native.country         32561 non-null  object  
14  income                 32561 non-null  object  
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

Summary of the dataset shows that there are no missing values. But the preview shows that the dataset contains values coded as ?. So, I will encode ? as NaN values.

Encode ? as NaNs

```
In [8]: df[df == "?"] = np.nan
```

```
In [9]: df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 32561 entries, 0 to 32560
Data columns (total 15 columns):
#   Column                Non-Null Count  Dtype
---  -
0   age                   32561 non-null  int64
1   workclass             30725 non-null  object
2   fnlwgt               32561 non-null  int64
3   education            32561 non-null  object
4   education.num        32561 non-null  int64
5   marital.status       32561 non-null  object
6   occupation           30718 non-null  object
7   relationship         32561 non-null  object
8   race                 32561 non-null  object
9   sex                  32561 non-null  object
10  capital.gain         32561 non-null  int64
11  capital.loss         32561 non-null  int64
12  hours.per.week       32561 non-null  int64
13  native.country       31978 non-null  object
14  income               32561 non-null  object
dtypes: int64(6), object(9)
memory usage: 3.7+ MB
```

impute missing values with code

```
In [10]: for col in ["workclass", "occupation", "native.country"]:
         df[col].fillna(df[col].mode()[0], inplace=True)
```

check again for missing value

```
In [11]: df.isnull().sum()
```

```
Out[11]: age                0
workclass                0
fnlwgt                  0
education                0
education.num            0
marital.status           0
occupation              0
relationship            0
race                    0
sex                     0
capital.gain             0
capital.loss             0
hours.per.week           0
native.country           0
income                  0
dtype: int64
```

now there is no missing value

Setting feature vector and target variable

```
In [12]: x = df.drop(["income"], axis=1)
```

```
In [13]: y = df["income"]
```

```
In [14]: x.head()
```

Out[14]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	relationsh
0	90	Private	77053	HS-grad	9	Widowed	Prof-specialty	Not-in-far
1	82	Private	132870	HS-grad	9	Widowed	Exec-managerial	Not-in-far
2	66	Private	186061	Some-college	10	Widowed	Prof-specialty	Unmarri
3	54	Private	140359	7th-8th	4	Divorced	Machine-op-inspct	Unmarri
4	41	Private	264663	Some-college	10	Separated	Prof-specialty	Own-ch

Split data into separate train and test dataset

```
In [15]:  from sklearn.model_selection import train_test_split
```

```
In [16]:  x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3,
```

Feature Engineering

Encode categorical variables

```
In [17]:  from sklearn import preprocessing
```

```
In [18]:  categorical = ["workclass", "education", "marital.status", "occupation", "
for feature in categorical:
    le = preprocessing.LabelEncoder()
    x_train[feature] = le.fit_transform(x_train[feature])
    x_test[feature] = le.transform(x_test[feature])
```

Feature Scaling

```
In [19]:  from sklearn.preprocessing import StandardScaler
```

```
In [20]:  scaler = StandardScaler()
```

```
In [21]:  x_train = pd.DataFrame(scaler.fit_transform(x_train), columns = x.columns)
```

```
In [22]: ▶ pd.DataFrame(scaler.transform(x_test), columns = x.columns)
```

Out[22]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation
0	1.273263	-0.090641	0.798307	-1.107252	-1.980744	2.255673	1.232533
1	-1.436476	-0.090641	0.448823	0.184396	-0.423425	0.926666	-0.278542
2	-1.143531	-0.090641	-0.608164	1.217715	-0.034095	0.926666	0.225150
3	-0.118225	-2.781760	-1.332357	-0.332263	1.133894	-0.402341	0.728841
4	0.760610	-0.090641	2.202540	0.442726	1.523223	-0.402341	1.232533
...
9764	-0.118225	1.703439	-1.518569	0.184396	-0.423425	-0.402341	-0.530388
9765	-0.923823	-0.090641	-0.228829	0.184396	-0.423425	-0.402341	1.232533
9766	-0.997059	-0.090641	-0.312141	1.217715	-0.034095	0.926666	0.728841
9767	-0.337933	-0.090641	-0.393536	0.184396	-0.423425	0.926666	-0.026696
9768	0.833846	-0.090641	-0.901761	-0.590592	0.355234	-0.402341	-1.034080

9769 rows × 14 columns

```
In [23]: ▶ x_train.head()
```

Out[23]:

	age	workclass	fnlwgt	education	education.num	marital.status	occupation	re
0	0.101484	2.600478	-1.494279	-0.332263	1.133894	-0.402341	-0.782234	
1	0.028248	-1.884720	0.438778	0.184396	-0.423425	-0.402341	-0.026696	
2	0.247956	-0.090641	0.045292	1.217715	-0.034095	0.926666	-0.782234	
3	-0.850587	-1.884720	0.793152	0.184396	-0.423425	0.926666	-0.530388	
4	-0.044989	-2.781760	-0.853275	0.442726	1.523223	-0.402341	-0.782234	

Logistic Regression Model with all Features

```
In [24]: ▶ from sklearn.linear_model import LogisticRegression
```

```
In [25]: ▶ from sklearn.metrics import accuracy_score
```

```
In [26]: logreg = LogisticRegression()
```

```
In [27]: logreg.fit(x_train, y_train)
```

```
Out[27]: LogisticRegression
LogisticRegression()
```

```
In [28]: y_pred = logreg.predict(x_test)
```

```
In [29]: print("Logistic Regression accuracy score with all the feature: {0:0.4f}").
```

Logistic Regression accuracy score with all the feature: 0.2415

Logistic Regression with PCA

lets get to the PCA implementation

```
In [30]: from sklearn.decomposition import PCA
```

```
In [31]: pca = PCA()
```

```
In [32]: x_train = pca.fit_transform(x_train)
```

```
In [33]: pca.explained_variance_ratio_
```

```
Out[33]: array([0.14757168, 0.10182915, 0.08147199, 0.07880174, 0.07463545,
0.07274281, 0.07009602, 0.06750902, 0.0647268 , 0.06131155,
0.06084207, 0.04839584, 0.04265038, 0.02741548])
```

We can see that approximately 97.25% of variance is explained by the first 13 variables.

-Only 2.75% of variance is explained by the last variable. So, we can assume that it carries little information.

-So, I will drop it, train the model again and calculate the accuracy.

Logistic Regression with first 13 feature


```
In [34]: x = df.drop(["income", "native.country"], axis=1)
```

```
In [35]: y = df["income"]
```

```
In [36]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3,
```

```
In [37]: categorical = ['workclass', 'education', 'marital.status', 'occupation',  
for feature in categorical:  
    le = preprocessing.LabelEncoder()  
    x_train[feature] = le.fit_transform(x_train[feature])  
    x_test[feature] = le.transform(x_test[feature])
```

```
In [38]: x_train = pd.DataFrame(scaler.fit_transform(x_train), columns = x.columns)
```

```
In [39]: x_test = pd.DataFrame(scaler.transform(x_test), columns = x.columns)
```

```
In [40]: logreg = LogisticRegression()
```

```
In [41]: logreg.fit(x_train, y_train)
```

```
Out[41]: LogisticRegression  
LogisticRegression()
```

```
In [42]: y_pred = logreg.predict(x_test)
```

```
In [43]: print('Logistic Regression accuracy score with the first 13 features: {0:6f}'.format(logreg.score(x_test, y_test)))
```

Logistic Regression accuracy score with the first 13 features: 0.8213

Logistic Regression WITH FIRST 12 feature

```
In [44]: x = df.drop(["income", "native.country", "hours.per.week"], axis=1)
```

```
In [45]: y = df["income"]
```

```
In [46]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3,
```

```
In [47]: categorical = ['workclass', 'education', 'marital.status', 'occupation',
for feature in categorical:
    le = preprocessing.LabelEncoder()
    x_train[feature] = le.fit_transform(x_train[feature])
    x_test[feature] = le.transform(x_test[feature])
```

```
In [48]: x_train = pd.DataFrame(scaler.fit_transform(x_train), columns = x.columns)
```

```
In [49]: x_test = pd.DataFrame(scaler.transform(x_test), columns = x.columns)
```

```
In [50]: logreg.fit(x_train, y_train)
```

```
Out[50]: LogisticRegression
LogisticRegression()
```

```
In [51]: y_pred = logreg.predict(x_test)
```

```
In [52]: print('Logistic Regression accuracy score with the first 13 features: {0:6f}'.format(accuracy_score(y_test, y_pred)))
```

Logistic Regression accuracy score with the first 13 features: 0.8227

Logistic Regression WITH FIRST 11 feature

```
In [53]: x = df.drop(["income", "native.country", "hours.per.week", "capital.loss"], axis=1)
```

```
In [54]: y = df["income"]
```

```
In [55]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 42)
```

```
In [56]: categorical = ['workclass', 'education', 'marital.status', 'occupation',
for feature in categorical:
    le = preprocessing.LabelEncoder()
    x_train[feature] = le.fit_transform(x_train[feature])
    x_test[feature] = le.transform(x_test[feature])
```

```
In [57]: x_train = pd.DataFrame(scaler.fit_transform(x_train), columns = x.columns)
```

```
In [58]: x_test = pd.DataFrame(scaler.transform(x_test), columns = x.columns)
```

```
In [59]: logreg.fit(x_train, y_train)
```

```
Out[59]: LogisticRegression
LogisticRegression()
```

```
In [60]: y_pred = logreg.predict(x_test)
```

```
In [61]: print('Logistic Regression accuracy score with the first 13 features: {0:6f}'.format(logreg.score(x_test, y_test)))

Logistic Regression accuracy score with the first 13 features: 0.8186
```

:-We can see that accuracy has significantly decreased to 0.8187 if I drop the last three features.

-Our aim is to maximize the accuracy. We get maximum accuracy with the first 12 features and the accuracy is 0.8227.

Select right number of dimentions

```
In [62]: x = df.drop(["income"], axis=1)
```

```
In [63]: y = df["income"]
```

```
In [64]: x_train, x_test, y_train, y_test = train_test_split(x, y, test_size = 0.3, random_state = 42)
```

```
In [65]: categorical = ['workclass', 'education', 'marital.status', 'occupation',
for feature in categorical:
    le = preprocessing.LabelEncoder()
    x_train[feature] = le.fit_transform(x_train[feature])
    x_test[feature] = le.transform(x_test[feature])
```

```
In [66]: x_train = pd.DataFrame(scaler.fit_transform(x_train), columns = x.columns)
```

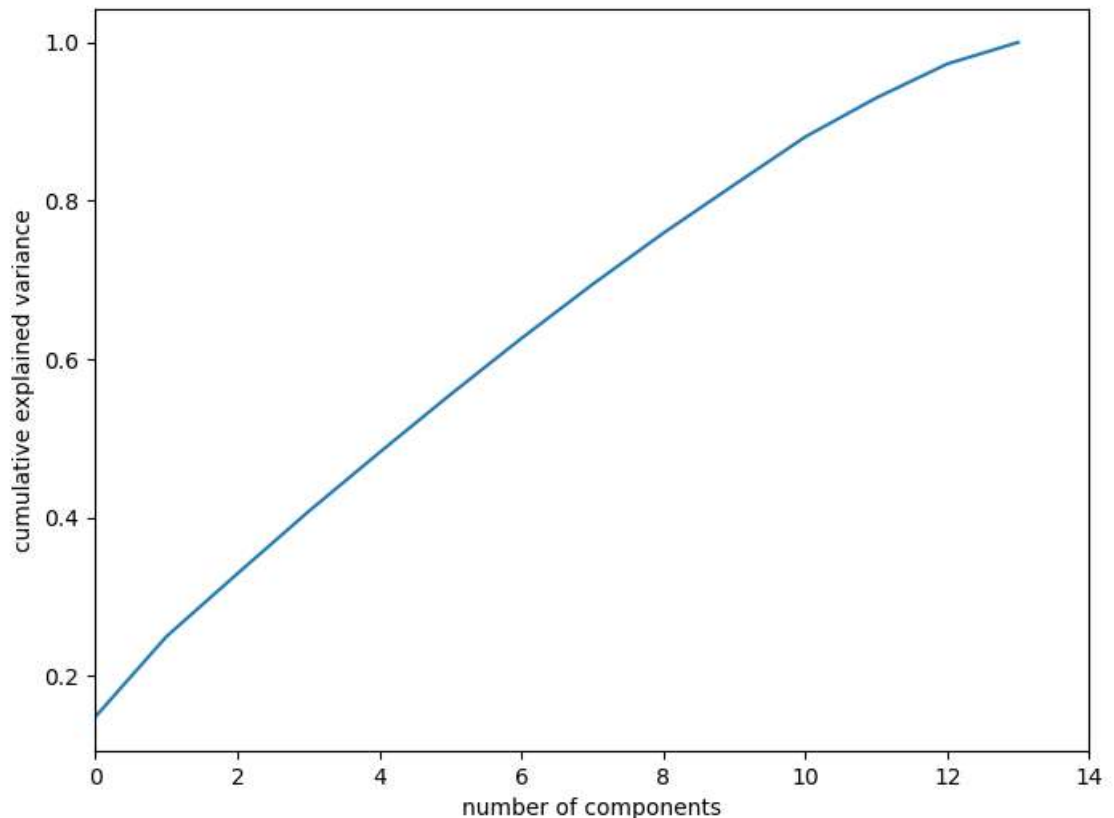
```
In [67]: pca = PCA()
pca.fit(x_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
dim = np.argmax(cumsum >= 0.90) + 1
```

```
In [68]: print("The numnber of dimensions required to preserve 90% of variance is", dim)

The number of dimensions required to preserve 90% of variance is 12
```

plot explained variance ratio with number of dimensions

```
In [69]: ▶ plt.figure(figsize=(8,6))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0,14,1)
plt.xlabel("number of components")
plt.ylabel("cumulative explained variance")
plt.show()
```



Conclusion

-In this kernel, I have discussed Principal Component Analysis – the most popular dimensionality reduction technique.

-I have demonstrated PCA implementation with Logistic Regression on the adult dataset.

-I found the maximum accuracy with the first 12 features and it is found to be 0.8227.

-As expected, the number of dimensions required to preserve 90 % of variance is found to be 12.

-Finally, I plot the explained variance ratio with number of dimensions. The graph confirms that approximately 90% of variance is explained by the first 12 components.

In []: ▶