Day52_PCA_Logistic_Regression

July 25, 2025

PCA – Principal Component Analysis

What is PCA?

Principal Component Analysis (PCA) is a powerful statistical technique used for **dimensionality reduction**. It transforms a dataset with many features (columns) into a smaller set that still contains most of the important information (variance).

If your dataset has too many attributes (e.g. 30), it can lead to **overfitting**. PCA helps reduce the number of attributes by projecting them onto **Principal Components** — new axes that capture maximum variance in the data.

Why Use PCA?

- Reduce overfitting
- Simplify models by removing redundant features
- Improve model training speed
- Make visualization easier in 2D or 3D
- Improve generalization on unseen data

PCA is also useful in: - Clustering - Preprocessing before classification - Noise filtering

How PCA Helps Reduce Overfitting

PCA removes irrelevant/noisy features and focuses only on components that explain variance.

You said:

> *"PCA reduces overfitting by using only top components (PC1, PC2…) instead of all original features."*

Reducing dimensions avoids letting the model memorize noise, hence improving generalization.

PCA in a Machine Learning Pipeline

Your class used this flow:

`Raw Data → StandardScaler → PCA → Logistic Regression`

This works well with:

- Logistic Regression
- K-Nearest Neighbors (KNN)
- SVM
- Neural Networks (as preprocessing)

Summary Table

Term	Description
PCA	Principal Component Analysis
Dimensionality	Number of features (columns)
Overfitting	Model performs well on train data but poorly on test data
PC1, PC2,...	Principal components with highest variance
Eigenvalue	Importance of a principal component
Eigenvector	Direction of the component (used to rotate original axes)
$\mathbf{Z} = \mathbf{X} \cdot \mathbf{W}$	Data projected onto new reduced axis

Real Example from Class

You had 30 original attributes:

- After PCA, PC1 becomes a combination of those 30.
- Instead of using all 30 features, you use top 5–10 that explain most variance.
- Model is then trained on **reduced input**, leading to faster and more accurate results.

Conclusion

- PCA helps simplify your data without losing essential information.
- It's a go-to tool when working with **high-dimensional datasets**.
- Ideal when you're worried about **overfitting**, **noise**, or **slow training**.

1 Import Python libraries

[1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from sklearn import preprocessing

import warnings
warnings.filterwarnings('ignore')
%matplotlib inline
```

2 Import dataset

[2]:
```
df = pd.read_csv(r"C:\Users\Lenovo\OneDrive\Desktop\24 July\21st, 22nd -␣
↪logistic, pca\Project\adult.csv\adult.csv")
```

3 Exploratory Data Analysis

[3]:
```
df.shape
```

[3]:
```
(32561, 15)
```

We can see that there are 32561 instances and 15 attributes in the data set.

[4]:
```
df.head()
```

[4]:
```
   age workclass  fnlwgt     education  education.num marital.status  \
0   90         ?   77053       HS-grad              9        Widowed
1   82   Private  132870       HS-grad              9        Widowed
2   66         ?  186061  Some-college             10        Widowed
3   54   Private  140359       7th-8th              4       Divorced
4   41   Private  264663  Some-college             10      Separated

          occupation   relationship   race     sex  capital.gain  \
0                  ?  Not-in-family  White  Female             0
1    Exec-managerial  Not-in-family  White  Female             0
2                  ?      Unmarried  Black  Female             0
3  Machine-op-inspct      Unmarried  White  Female             0
4     Prof-specialty      Own-child  White  Female             0

   capital.loss  hours.per.week native.country income
0          4356              40  United-States  <=50K
1          4356              18  United-States  <=50K
2          4356              40  United-States  <=50K
3          3900              40  United-States  <=50K
4          3900              40  United-States  <=50K
```

[5]:
```
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.

4 Encode ? as NaNs

[6]:
```python
df[df == '?'] = np.nan
```

5 Again check the summary of dataframe

[7]:
```python
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
```

Now, the summary shows that the variables - workclass, occupation and native.country contain missing values.

All of these variables are categorical data type. So, I will impute the missing values with the most frequent value- the mode.

6 Impute missing values with mode

[8]:
```
for col in ['workclass', 'occupation', 'native.country']:
    df[col].fillna(df[col].mode()[0], inplace=True)
```

7 Check again for missing values

[9]:
```
df.isnull().sum()
```

[9]:
```
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 we can see that there are no missing values in the dataset.

[10]:
```
df.columns
```

[10]:
```
Index(['age', 'workclass', 'fnlwgt', 'education', 'education.num',
       'marital.status', 'occupation', 'relationship', 'race', 'sex',
       'capital.gain', 'capital.loss', 'hours.per.week', 'native.country',
       'income'],
      dtype='object')
```

8 Feature and Target Split

[11]:
```python
X = df.drop(['income'], axis=1)

y = df['income']
```

[12]:
```python
X.head()
```

[12]:
```
   age workclass  fnlwgt     education  education.num marital.status  \
0   90   Private   77053       HS-grad              9        Widowed
1   82   Private  132870       HS-grad              9        Widowed
2   66   Private  186061  Some-college             10        Widowed
3   54   Private  140359       7th-8th              4       Divorced
4   41   Private  264663  Some-college             10      Separated

          occupation   relationship   race     sex  capital.gain  \
0     Prof-specialty  Not-in-family  White  Female             0
1    Exec-managerial  Not-in-family  White  Female             0
2     Prof-specialty      Unmarried  Black  Female             0
3  Machine-op-inspct      Unmarried  White  Female             0
4     Prof-specialty      Own-child  White  Female             0

   capital.loss  hours.per.week native.country
0          4356              40  United-States
1          4356              18  United-States
2          4356              40  United-States
3          3900              40  United-States
4          3900              40  United-States
```

9 Split data into separate training and test set

[13]:
```python
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
↪random_state = 0)
```

10 Feature Engineering

10.1 Encode categorical variables

[14]:
```python
categorical = ['workclass', 'education', 'marital.status', 'occupation',␣
↪'relationship', 'race', 'sex', 'native.country']
for feature in categorical:
        le = preprocessing.LabelEncoder()
        X_train[feature] = le.fit_transform(X_train[feature])
        X_test[feature] = le.transform(X_test[feature])
```

11 Feature Scaling

[15]:
```
scaler = StandardScaler()

X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X.columns)

X_test = pd.DataFrame(scaler.transform(X_test), columns = X.columns)
```

[16]:
```
X_train.head()
```

[16]:
```
        age  workclass    fnlwgt  education  education.num  marital.status  \
0  0.101484   2.600478 -1.494279  -0.332263       1.133894       -0.402341
1  0.028248  -1.884720  0.438778   0.184396      -0.423425       -0.402341
2  0.247956  -0.090641  0.045292   1.217715      -0.034095        0.926666
3 -0.850587  -1.884720  0.793152   0.184396      -0.423425        0.926666
4 -0.044989  -2.781760 -0.853275   0.442726       1.523223       -0.402341

   occupation  relationship     race       sex  capital.gain  capital.loss  \
0   -0.782234      2.214196  0.39298 -1.430470     -0.145189     -0.217407
1   -0.026696     -0.899410  0.39298  0.699071     -0.145189     -0.217407
2   -0.782234     -0.276689  0.39298 -1.430470     -0.145189     -0.217407
3   -0.530388      0.968753  0.39298  0.699071     -0.145189     -0.217407
4   -0.782234     -0.899410  0.39298  0.699071     -0.145189     -0.217407

   hours.per.week  native.country
0       -1.662414        0.262317
1       -0.200753        0.262317
2       -0.038346        0.262317
3       -0.038346        0.262317
4       -0.038346        0.262317
```

12 Logistic Regression model with all features

[17]:
```
logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)
print('Logistic Regression accuracy score with all the features: {0:0.4f}'.
 ↪format(accuracy_score(y_test, y_pred)))
```

```
Logistic Regression accuracy score with all the features: 0.8218
```

Logistic Regression with PCA

Scikit-Learn's PCA class implements PCA algorithm using the code below. Before diving deep, I will explain another important concept called explained variance ratio.

Explained Variance Ratio

A very useful piece of information is the explained variance ratio of each principal component. It is available via the explained_variance_ratio_ variable. It indicates the proportion of the dataset's variance that lies along the axis of each principal component.

Now, let's get to the PCA implementation.

13 PCA implementation.

[18]:
```
from sklearn.decomposition import PCA
pca = PCA()
X_train = pca.fit_transform(X_train)
pca.explained_variance_ratio_
```

[18]:
```
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])
```

Comment

- 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.

13.1 Logistic Regression with first 13 features

[19]:
```
X = df.drop(['income','native.country'], axis=1)
y = df['income']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state = 0)


categorical = ['workclass', 'education', 'marital.status', 'occupation',␣
 ↪'relationship', 'race', 'sex']
for feature in categorical:
        le = preprocessing.LabelEncoder()
        X_train[feature] = le.fit_transform(X_train[feature])
        X_test[feature] = le.transform(X_test[feature])


X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X.columns)

X_test = pd.DataFrame(scaler.transform(X_test), columns = X.columns)

logreg = LogisticRegression()
logreg.fit(X_train, y_train)
```

```
y_pred = logreg.predict(X_test)

print('Logistic Regression accuracy score with the first 13 features: {0:0.4f}'.
 ↪ format(accuracy_score(y_test, y_pred)))
```

```
Logistic Regression accuracy score with the first 13 features: 0.8213
```

Comment

- We can see that accuracy has been decreased from 0.8218 to 0.8213 after dropping the last feature.
- Now, if I take the last two features combined, then we can see that approximately 7% of variance is explained by them.
- I will drop them, train the model again and calculate the accuracy.

13.2 Logistic Regression with first 12 features

[20]:
```
X = df.drop(['income','native.country', 'hours.per.week'], axis=1)
y = df['income']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
 ↪random_state = 0)


categorical = ['workclass', 'education', 'marital.status', 'occupation',␣
 ↪'relationship', 'race', 'sex']
for feature in categorical:
        le = preprocessing.LabelEncoder()
        X_train[feature] = le.fit_transform(X_train[feature])
        X_test[feature] = le.transform(X_test[feature])


X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X.columns)

X_test = pd.DataFrame(scaler.transform(X_test), columns = X.columns)

logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)

print('Logistic Regression accuracy score with the first 12 features: {0:0.4f}'.
 ↪ format(accuracy_score(y_test, y_pred)))
```

```
Logistic Regression accuracy score with the first 12 features: 0.8227
```

Comment

- Now, it can be seen that the accuracy has been increased to 0.8227, if the model is trained with 12 features.
- Lastly, I will take the last three features combined. Approximately 11.83% of variance is explained by them.
- I will repeat the process, drop these features, train the model again and calculate the accuracy.

13.3 Logistic Regression with first 11 features

[21]:
```
X = df.drop(['income','native.country', 'hours.per.week', 'capital.loss'],␣
↪axis=1)
y = df['income']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
↪random_state = 0)


categorical = ['workclass', 'education', 'marital.status', 'occupation',␣
↪'relationship', 'race', 'sex']
for feature in categorical:
        le = preprocessing.LabelEncoder()
        X_train[feature] = le.fit_transform(X_train[feature])
        X_test[feature] = le.transform(X_test[feature])


X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X.columns)

X_test = pd.DataFrame(scaler.transform(X_test), columns = X.columns)

logreg = LogisticRegression()
logreg.fit(X_train, y_train)
y_pred = logreg.predict(X_test)

print('Logistic Regression accuracy score with the first 11 features: {0:0.4f}'.
↪ format(accuracy_score(y_test, y_pred)))
```

```
Logistic Regression accuracy score with the first 11 features: 0.8186
```

Comment

- 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 dimensions

- The above process works well if the number of dimensions are small.
- But, it is quite cumbersome if we have large number of dimensions.

- In that case, a better approach is to compute the number of dimensions that can explain significantly large portion of the variance.
- The following code computes PCA without reducing dimensionality, then computes the minimum number of dimensions required to preserve 90% of the training set variance.

[22]:
```
X = df.drop(['income'], axis=1)
y = df['income']


X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.3,␣
↪random_state = 0)


categorical = ['workclass', 'education', 'marital.status', 'occupation',␣
↪'relationship', 'race', 'sex', 'native.country']
for feature in categorical:
        le = preprocessing.LabelEncoder()
        X_train[feature] = le.fit_transform(X_train[feature])
        X_test[feature] = le.transform(X_test[feature])


X_train = pd.DataFrame(scaler.fit_transform(X_train), columns = X.columns)


pca= PCA()
pca.fit(X_train)
cumsum = np.cumsum(pca.explained_variance_ratio_)
dim = np.argmax(cumsum >= 0.90) + 1
print('The number of dimensions required to preserve 90% of variance is',dim)
```

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

Comment

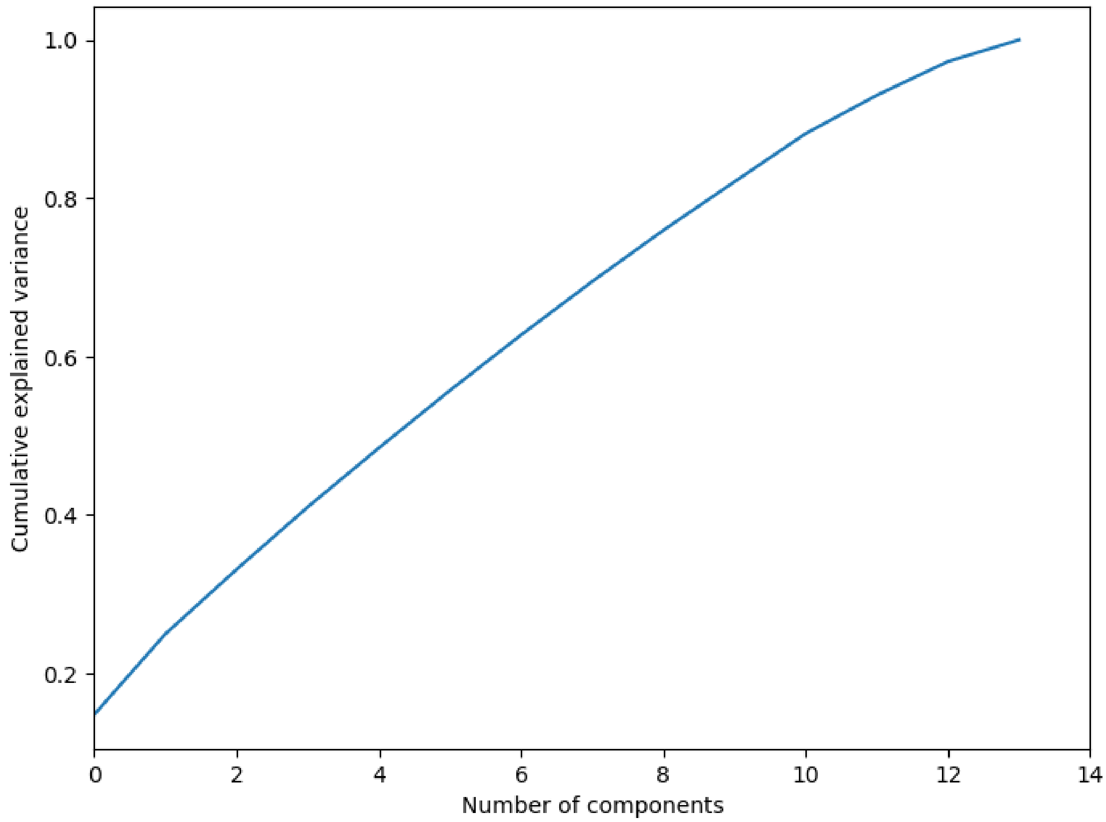
With the required number of dimensions found, we can then set number of dimensions to dim and run PCA again.

With the number of dimensions set to dim, we can then calculate the required accuracy.

Plot explained variance ratio with number of dimensions

- An alternative option is to plot the explained variance as a function of the number of dimensions.
- In the plot, we should look for an elbow where the explained variance stops growing fast.
- This can be thought of as the intrinsic dimensionality of the dataset.
- Now, I will plot cumulative explained variance ratio with number of components to show how variance ratio varies with number of components.

[23]:
```
plt.figure(figsize=(8,6))
plt.plot(np.cumsum(pca.explained_variance_ratio_))
plt.xlim(0,14)
plt.xlabel('Number of components')
plt.ylabel('Cumulative explained variance')
plt.show()
```



Comment The above plot shows that almost 90% of variance is explained by the first 12 components.

14 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.