EDA + Logistic Regression + PCA

Principal Component Analysis - a **Dimensionality Reduction** technique.

Table of Contents

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

The Curse of Dimensionality

Generally, real world datasets contain thousands or millions of features to train for. This is very time consuming task as this makes training extremely slow. In such cases, it is very difficult to find a good solution. This problem is often referred to as the curse of dimensionality.

The curse of dimensionality refers to various phenomena that arise when we analyze and organize data in high dimensional spaces (often with hundreds or thousands of dimensions) that do not occur in low-dimensional settings. The problem is that when the dimensionality increases, the volume of the space increases so fast that the available data become sparse. This sparsity is problematic for any method that requires statistical significance.

In real-world problems, it is often possible to reduce the number of dimensions considerably. This process is called **dimensionality reduction**. It refers to the process of reducing the number of dimensions under consideration by obtaining a set of principal variables. It helps to speed up training and is also extremely useful for data visualization.

The most popular dimensionality reduction technique is Principal Component Analysis (PCA), which is discussed below.

Introduction to Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a dimensionality reduction technique that can be used to reduce a larger set of feature variables into a smaller set that still contains most of the variance in the larger set.

Preserve the variance

PCA, first identifies the hyperplane that lies closest to the data and then it projects the data onto it. Before, we can project the training set onto a lower-dimensional hyperplane, we need to select the right hyperplane. The projection can be done in such a way so as to preserve the maximum variance. This is the idea behind PCA.

Principal Components

PCA identifies the axes that accounts for the maximum amount of cumulative sum of variance in the training set. These are called Principal Components. PCA assumes that the dataset is centered around the origin. Scikit-Learn's PCA classes take care of centering the data automatically.

Projecting down to d Dimensions

Once, we have identified all the principal components, we can reduce the dimensionality of the dataset down to d dimensions by projecting it onto the hyperplane defined by the first d principal components. This ensures that the projection will preserve as much variance as possible.

Now, let's get to the implementation.

```
## Import Python libraries

import numpy as np

import pandas as pd #

# import libraries for plotting

import matplotlib.pyplot as plt

import seaborn as sns

%matplotlib inline

# ignore warnings

import warnings

warnings.filterwarnings('ignore')

df = pd.read_csv(r"C:\Users\Hanshu\Desktop\excel data_ML\adult.csv\adult.csv")

import os

folder_path = r"C:\Users\Hanshu\Desktop\excel data_ML\adult.csv"
```

```
print(os.listdir(folder_path))
```

```
### Check file size

print('# File Sizes')

for f in os.listdir(folder_path):

   print(f.ljust(30) + str(round(os.path.getsize(folder_path) / 1000000, 2)) + 'MB')

## Import dataset

%%time


file = (r"C:\Users\Hanshu\Desktop\excel data_ML\adult.csv\adult.csv")

df = pd.read_csv(file, encoding='latin-1')

## Exploratory Data Analysis

### Check shape of dataset

df.shape
```

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

```
### Preview dataset

df.head()

### View summary of dataframe

df.info()
```

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`

df[df == '?'] = np.nan

### Again check the summary of dataframe

df.info()
```

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.

```
### Impute missing values with mode

for col in ['workclass', 'occupation', 'native.country']:
```

```
    df[col].fillna(df[col].mode()[0], inplace=True)
```

Check again for missing values

```
df.isnull().sum()
```

Now we can see that there are no missing values in the dataset.

Setting feature vector and target variable

```
x = df.drop(['income'], axis =1)
y = df['income']
x.head()
```

Split data into separate training and test set

```
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)
```

Feature Engineering

Encode categorical variables

```
from sklearn import preprocessing

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])
```

Feature Scaling

```
from sklearn.preprocessing import StandardScaler

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)
x_train.head()
```

Logistic Regression model with all features

```
from sklearn.linear_model import LogisticRegression

from sklearn.metrics import accuracy_score


logreg = LogisticRegression()

logreg.fit(x_train, y_train)

y_pred = logreg.predict(x_test)


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

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.

```
from sklearn.decomposition import PCA

pca = PCA()

x_train = pca.fit_transform(x_train)

pca.explained_variance_ratio_
```

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.

Logistic Regression with first 13 features

```
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)))
```

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.

Logistic Regression with first 11 features

```
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)))

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.

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)

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.

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()