IntelliPaat

Principal Component Analysis

Agenda

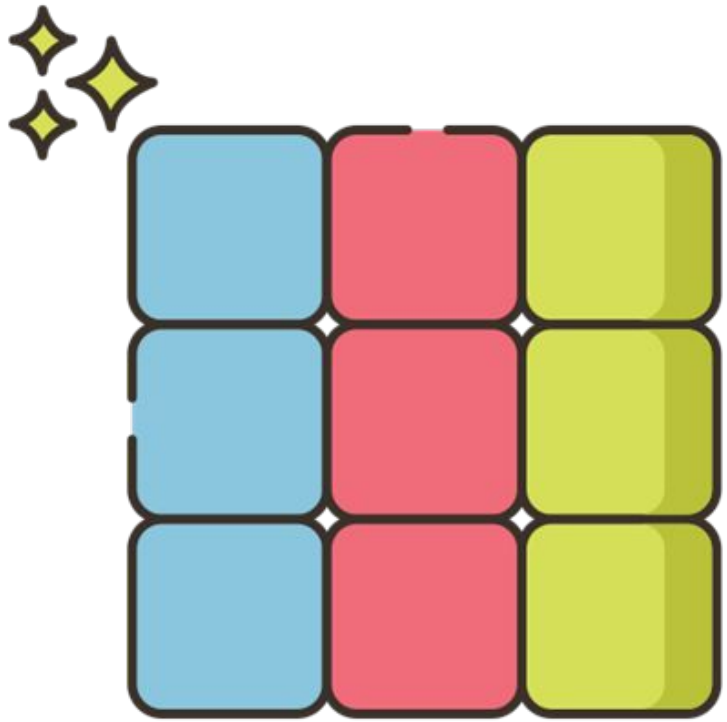
01 Need of PCA

02 Introduction

03 Detailed Approach to PCA

04 Limitations

05 Hands-on



Overview on Why PCA

Why PCA?

Imagine you are working on a large scale data and there are many features to deal with while building the model ,so you'll be facing constraints to select the features which are appropriate for your model.

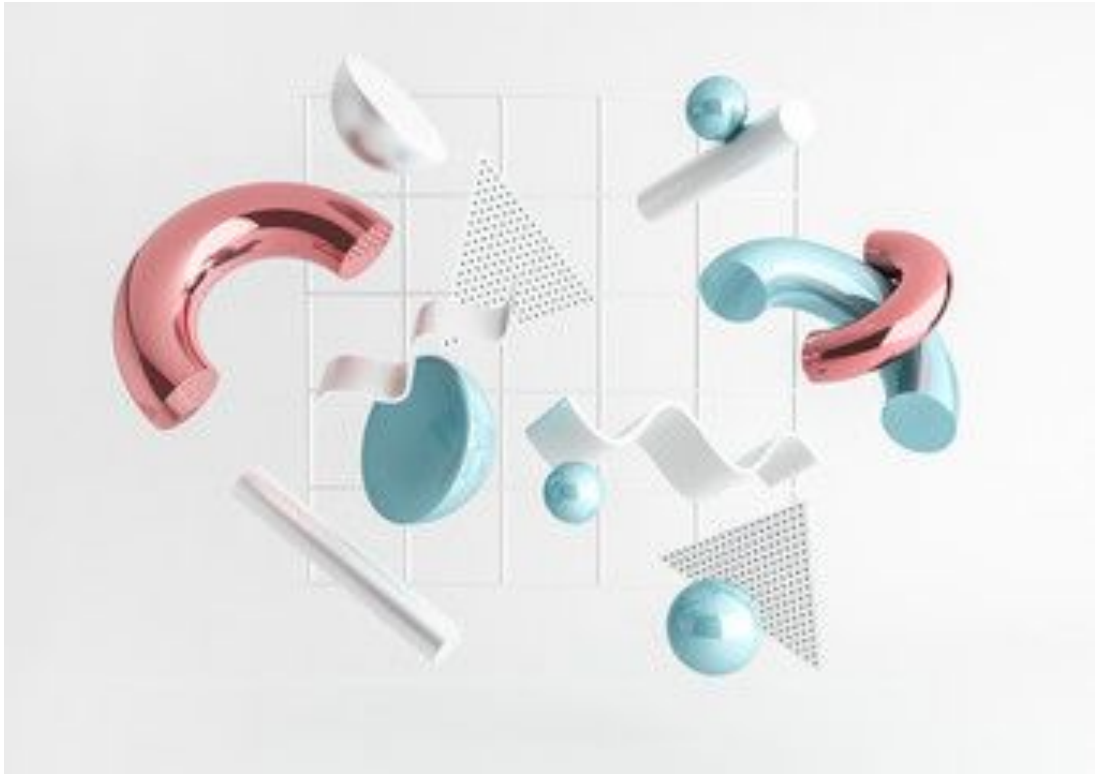
So below are the constraints :

Features will be highly correlated for analysis

Due to large scale of data,you''ll be ending up with confusion which feature to select and which not.

So this scenario will create too much confusion in model building

Why PCA?



So above mentioned constrained can be handled by performing PCA(Principal Component Analysis),where PCA takes all the features and only captures the relevant data needed for model building in the form of components.

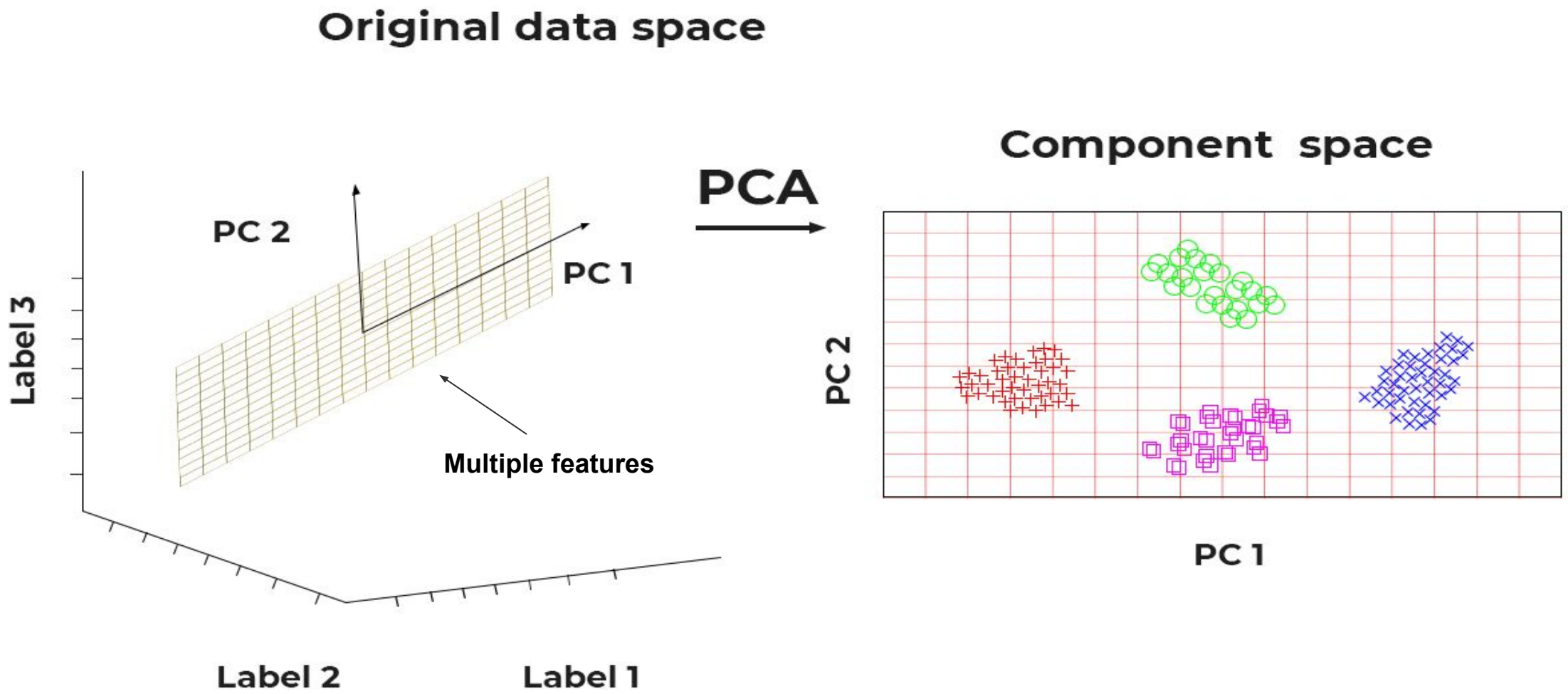
What is PCA?

What is PCA?

PCA is a method of obtaining important variables (as components) from a large set of variables available in a data set. It extracts a low dimensional feature set by taking an irrelevant size projection from a high dimensional data set with a reason to capture as much information as possible. With fewer variables obtained while minimizing information loss, visualization also becomes much more meaningful. PCA is particularly useful when dealing with three-dimensional or larger data.

Always performed on a symmetric correlation or covariance matrix. This means that the matrix must be numeric and have standardized data.

What is PCA?



Introduction

With the reference to the previous slide, you can observe that multiple features are reduced to 2 components .which makes building you model easy and speeds up the process of machine learning algorithms.

When to use PCA?

- The PCA technique is particularly useful in data processing where there is multicollinearity between features/variables.
- PCA can be used when the dimensions of the input features are large (e.g. many variables).
- PCA can also be used for noise reduction and data compression.

Detailed Approach to PCA

Step 1 :Scale the data

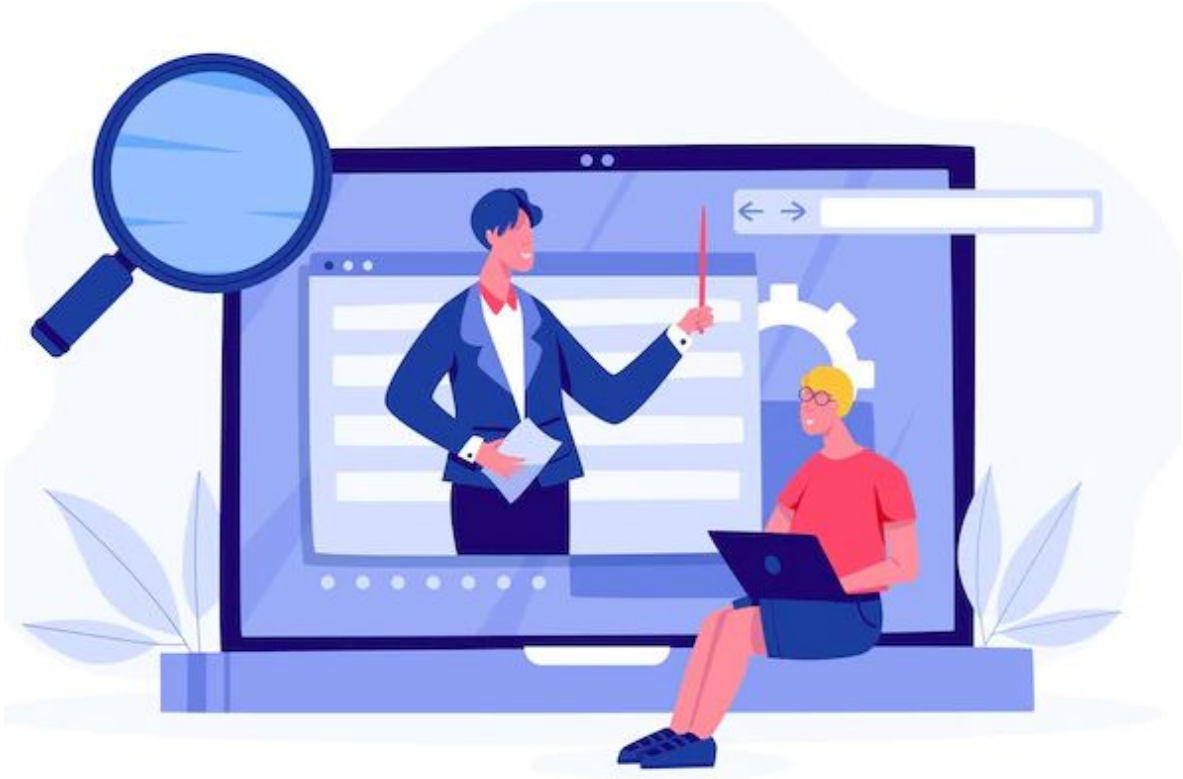
Standardising the data before using the data for implementation as it converts the data into same range which obtains optimal performance of machine learning algorithms.

Step:2 Eigen Decomposition

- The eigenvectors and eigenvalues of a covariance or correlation matrix represent the heart of a PCA:
- The eigenvectors which are called as Principal components determine the directions of the new feature space.
- The eigenvalues determine their magnitude. In other words, the eigenvalues explain the variance of the data along the new feature axes.

Step 3:fit the data

With the help of scikit learn library ,now fit the data to PCA which finally reduce your data into n PCA components.



Step 4:Check Explained_variance

- Explained_variance helps us to calculate how many variables can be attributed to each component.
- So this helps in avoiding overfitting of model by selecting the appropriate number of attributes

Step 5: Check Efficiency

Now compare the model efficiency before and after implementation PCA

Limitations

- Features become Less interpretable.

  features loses its readability after converting them into components

- Information loss

  As only highly variance components are selected ,there will huge loss in data

Hands-on on PCA

Hands-On

Import all necessary libraries and dataset.

```
import numpy as np
import pandas as pd
from sklearn.decomposition import PCA
from sklearn.cluster import KMeans
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline
```

```
from sklearn.datasets import load_digits
digits = load_digits()
```

```
data=pd.DataFrame(digits.data)
```

Hands-On

Scale the data

```
from sklearn.preprocessing import StandardScaler
X = data.values
# Data Normalization
X_std = StandardScaler().fit_transform(X)
```

```
X_std.shape
```

```
(1797, 64)
```

Hands-On

Decompose the data into eigen values and eigen vectors.and create a pair with eigen values and eigen vector and calculate the explained variance for all the columns

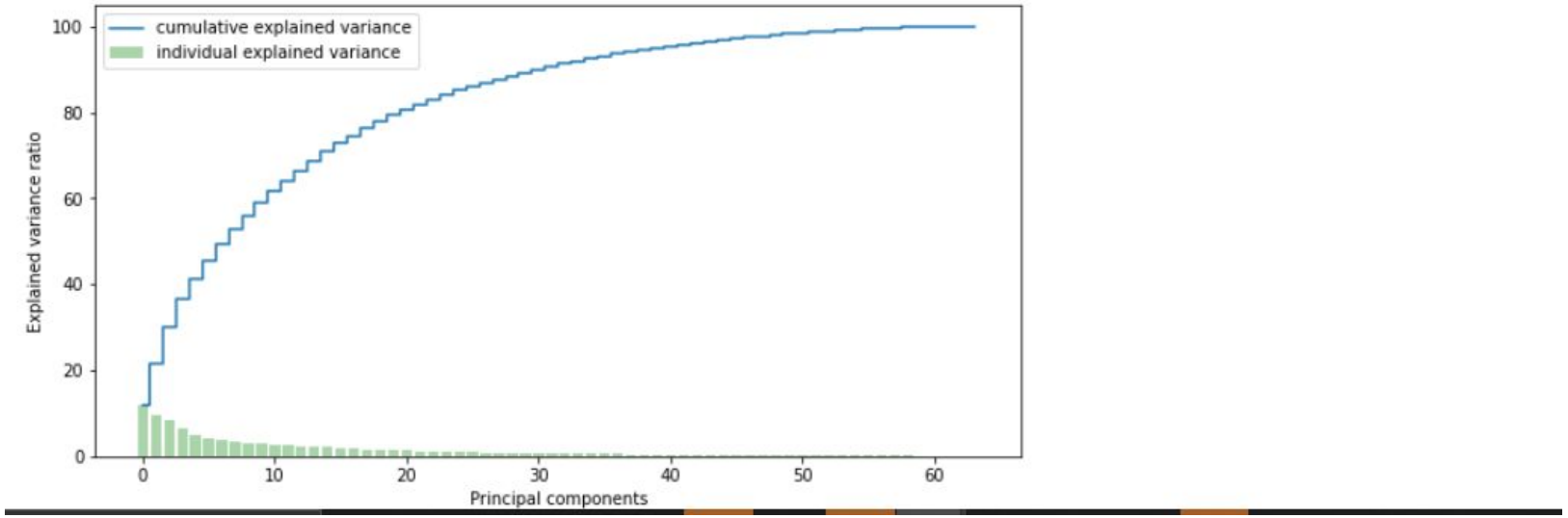
```
mean_vec = np.mean(X_std, axis=0)
cov_mat = np.cov(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(cov_mat)
```

```
# Create a list of (eigenvalue, eigenvector) tuples
eig_pairs = [ (np.abs(eig_vals[i]),eig_vecs[:,i]) for i in range(len(eig_vals))]

# Sort from high to low
eig_pairs.sort(key = lambda x: x[0], reverse= True)

# Calculation of Explained Variance from the eigenvalues
tot = sum(eig_vals)
var_exp = [(i/tot)*100 for i in sorted(eig_vals, reverse=True)] # Individual explained variance
cum_var_exp = np.cumsum(var_exp) # Cumulative explained variance
cum_var_exp
```

Hands-On

Plot step plot to select the total number of components to fit into PCA.

```
plt.figure(figsize=(10, 5))
plt.bar(range(len(var_exp)), var_exp, alpha=0.3333, align='center', label='individual explained variance', color = 'g')
plt.step(range(len(cum_var_exp)), cum_var_exp, where='mid',label='cumulative explained variance')
plt.ylabel('Explained variance ratio')
plt.xlabel('Principal components')
plt.legend(loc='best')
plt.show()
```



Hands-On

From the reference of previous plot you can select the n_components value as 20 and fit it PCA.

```
pca = PCA(n_components=20)
pca_x_train = pca.fit_transform(x_train)
pca_x_test = pca.transform(x_test)
```

PCA on Random Forest

Fit the above PCA data to Random forest model and find the accuracy of the model.

```
x_train,x_test,y_train,y_test=train_test_split(X_std,y)
```

```
pca = PCA(n_components=20)
pca_x_train = pca.fit_transform(x_train)
pca_x_test = pca.transform(x_test)
```

```
rf=DecisionTreeClassifier().fit(pca_x_train,y_train)
```

```
predicted=rf.predict(pca_x_test)
```

```
accuracy_score(predicted,y_test)
```

```
0.8333333333333334
```

PCA on Random Forest

Analysing misclassified data from the model.

```
# Plot the prediction
fig = plt.figure(figsize=(6, 6))  # figure size in inches
fig.subplots_adjust(left=0, right=1, bottom=0, top=1, hspace=0.05, wspace=0.05)

# plot the digits: each image is 8x8 pixels
for i in range(64):
    ax = fig.add_subplot(8, 8, i + 1, xticks=[], yticks=[])
    ax.imshow(x_test.reshape(-1, 8, 8)[i], cmap=plt.cm.binary,
              interpolation='nearest')

    # label the image with the target value
    if predicted[i] == expected[i]:
        ax.text(0, 7, str(predicted[i]), color='green')
    else:
        ax.text(0, 7, str(predicted[i]), color='red')
```

Visualising Misclassified Data