PCA—how to choose the number of components?

- In this article, I am going to show you how to choose the number of principal components when using principal component analysis for dimensionality reduction.
- Principal Component Analysis (PCA) is an unsupervised technique for dimensionality reduction.

How does PCA work exactly? ¶

- PCA is the eigenvalue decomposition of the covariance matrix obtained after centering the features, to find the directions of maximum variation. The eigenvalues represent the variance explained by each principal component.
- The purpose of PCA is to obtain an easier and faster way to both manipulate data set (reducing its dimensions) and retain most of the original information through the explained variance.

The question now is How many components should I use for dimensionality reduction? What is the "right" number?

- In this post, we will discuss some tips for selecting the optimal number of principal components by providing practical examples in Python, by:

```
  1. Observing the cumulative ratio of explained variance. (always t
ake variance greater than 90%)
  2. Apply Cross Validation
  3. Using the covariance matrix With Elbow Curve
```

- Here I discuss 3 Method To Choose n_copmponent in PCA. You can choose any of One among this three

Hands On Example

In [1]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.datasets import load_iris
%matplotlib inline
```

In [2]:
```
iris = load_iris()
iris_df = pd.DataFrame(iris.data,columns=[iris.feature_names])
iris_df.head()
```

Out[2]:

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)
0	5.1	3.5	1.4	0.2
1	4.9	3.0	1.4	0.2
2	4.7	3.2	1.3	0.2
3	4.6	3.1	1.5	0.2
4	5.0	3.6	1.4	0.2

In [3]:
```
iris_df['target'] = iris.target
```

In [4]:
```
X = iris.data
y = iris.target
```

In [5]:
```
X.shape,y.shape
```

Out[5]:
```
((150, 4), (150,))
```

Before Doing PCA You must scalling the Dataset

In [6]:
```
from sklearn.preprocessing import StandardScaler
X_std = StandardScaler().fit_transform(X)
```

In [7]:
```
X_std.shape
```

Out[7]:
```
(150, 4)
```

Method:1 Observing the cumulative ratio of explained variance.

- here 1st we have to Compute the co-variance matrix
- In Second Compute the eigen values and vectors.
- In last step Project the data along the top eigen vectors based on the n_components.

In [8]:
```
## Compute the eigen values and vectors And Arrange the eigen values in the desce
X_covariance_matrix = np.cov(X_std.T)
eig_vals, eig_vecs = np.linalg.eig(X_covariance_matrix)
print('Eigenvectors \n%s' %eig_vecs)
print('\nEigenvalues \n%s' %eig_vals)
```

```
Eigenvectors
[[ 0.52106591 -0.37741762 -0.71956635  0.26128628]
 [-0.26934744 -0.92329566  0.24438178 -0.12350962]
 [ 0.5804131  -0.02449161  0.14212637 -0.80144925]
 [ 0.56485654 -0.06694199  0.63427274  0.52359713]]

Eigenvalues
[2.93808505 0.9201649  0.14774182 0.02085386]
```

In [9]:
```
tot = sum(eig_vals)
var_exp = [(i / tot)*100 for i in sorted(eig_vals, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
print ("Variance captured by each component is \n",var_exp)
print(40 * '-')
print ("Cumulative variance captured as we travel each component \n",cum_var_exp)
```

```
Variance captured by each component is
 [72.96244541329985, 22.850761786701778, 3.668921889282875, 0.5178709107154802]
----------------------------------------
Cumulative variance captured as we travel each component
 [ 72.96244541  95.8132072   99.48212909 100.        ]
```

- Now you add all your Variance captured by each component and when it will be greater than 90 then you stop.
- Here Clearly the variance captured by the 1st 2 features when arranged in descending order contribute to almost 72+22 = 94% of the total variance by the features. Thus we can eliminate the remaining 2 features as they don't contribute much to the overall variance.
- Thus, the mystery of How to choose n_component in PCA is Solved
- n_components should be equal to the features which contribute a large number to the overall variance! The number depends on the business logic.

- So in this problem you take n_components = 2 as two feature have high value of Variance. we want the variance to be between 95–99%.

Method 2: Applying a cross-validation procedure

In [10]:
```
from sklearn.decomposition import PCA
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.pipeline import Pipeline
```

In [11]:
```
pca = PCA(len(iris.feature_names)-1)
log_reg = LogisticRegression(max_iter=1000)

pipe = Pipeline(steps=[('pca', pca), ('log_reg', log_reg)])

params = {
    'pca__n_components': list(range(1, len(iris.feature_names))),
    'log_reg__C': np.logspace(0.1, 1, 10)
}

random_search = RandomizedSearchCV (pipe, params)
random_search.fit(X, y)

print('Best parameters obtained from Grid Search:\n',random_search.best_params_,
      random_search.best_score_)
```

```
Best parameters obtained from Grid Search:
 {'pca__n_components': 3, 'log_reg__C': 3.9810717055349722} 0.9800000000000001
```

In [12]:
```
results = pd.DataFrame(random_search.cv_results_)
results.head()
```

Out[12]:

	mean_fit_time	std_fit_time	mean_score_time	std_score_time	param_pca__n_components	param
0	0.016112	0.004202	0.000300	0.000400	3	
1	0.010509	0.001265	0.000801	0.000400	3	
2	0.014011	0.002760	0.000700	0.000400	3	
3	0.010308	0.000401	0.000601	0.000491	2	
4	0.009807	0.001503	0.000601	0.000375	1	

Here you see in 3 component model give 96 and also 2 component model gives 96 so always take lowest feature because our aim to reduce dimensionality in dataset so here also you take 2 as n_component

Method 3: Using the covariance matrix With Elbow Curve

In [13]:
```
pca = PCA(n_components = len(iris.feature_names)-1)
principal_components = pca.fit_transform(X)

# calculating the covariance matrix
covMatrix = np.cov(principal_components.transpose())
```

In [14]:
```
np.set_printoptions(suppress=True)
covMatrix
```

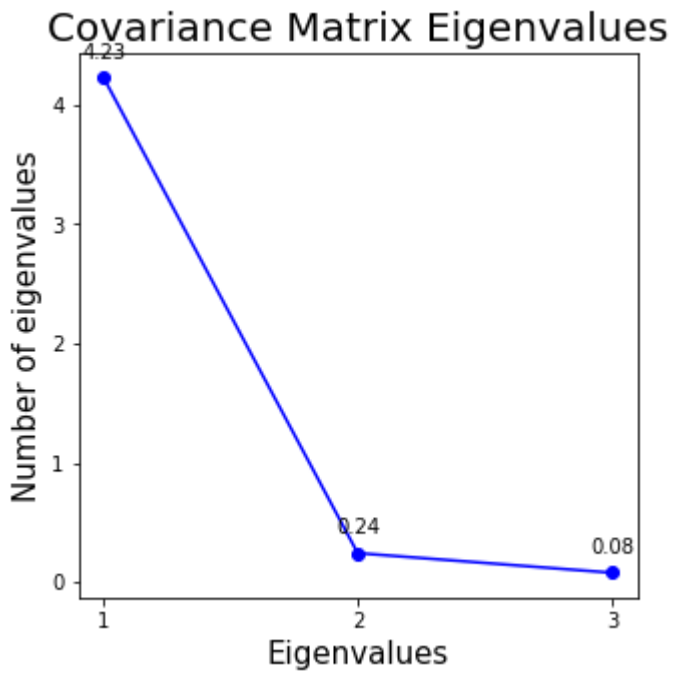
Out[14]:
```
array([[ 4.22824171, -0.        ,  0.        ],
       [-0.        ,  0.24267075, -0.        ],
       [ 0.        , -0.        ,  0.0782095 ]])
```

In [15]:
```
np.testing.assert_almost_equal(covMatrix - np.diag
                               (np.diagonal(covMatrix)), 0, decimal=10)
```

In [17]:
```
xp = np.arange(1, len(iris.feature_names), 1)
w = np.diag(covMatrix)

plt.clf()
plt.figure(figsize=(5,5))
plt.plot(xp, w, 'bo-')
plt.title("Covariance Matrix Eigenvalues", fontsize=20)
plt.xlabel('Eigenvalues', fontsize=15)
plt.ylabel('Number of eigenvalues',fontsize=15)
for x_plot,y_plot in zip(xp,w):
    label = "{:.2f}".format(y_plot)
    plt.annotate(label,
                 (x_plot,y_plot),
                 textcoords="offset points",
                 xytext=(0,10),
                 ha='center')
plt.xticks(np.arange(1,len(iris.feature_names),1))
plt.show()
```

```
<Figure size 432x288 with 0 Axes>
```



This Looks like your elbow method. By this also we calculate the n_component. Here n_copmonent also 2

Conclude :

- This tutorial is meant to provide a few tips on the selection of the number of components to be used for the dimensionality reduction in the PCA, showing practical demonstrations in Python.
- Finally, it is also explained how to perform the projection onto the reduced subspace of a new sample, information which is rarely found on tutorials on the subject.
- So Now i hope the mystry behind How to choose n_component in PCA is now Solved