PRINCIPAL COMPONENT ANALYSIS PCA

Principal Component Analysis

Goal

PCA finds a new set of dimensions such that all the dimensions are orthogonal (and hence linearly independent) and ranked according to the variance of data along them.

Find a transformation such that

- The tranformed features are linearly independent
- Dimensionality can be reduced by taking only the dimensions with the highest importance
- Those newly found dimensions should minimize the projection error
- The projected points should have maximum spread, i.e. maximum variance

Variance

How much variation or spread the data has.

$$\boxed{Var(X) = \frac{1}{n}\sum(X_i - \bar{X})^2}$$

Covariance Matrix

Indicates the level to which two variables vary together.

$$Cov(X,Y) = \frac{1}{n}\sum(X_i - \bar{X})(Y_i - \bar{Y})^T$$

$$Cov(X,X) = \frac{1}{n}\sum(X_i - \bar{X})(X_i - \bar{X})^T$$

Eigenvector, Eigenvalues

The eigenvectors point in the direction of the maximum variance, and the corresponding eigenvalues indicates the importance of its corresponding eigen vector.

$$A\vec{v} = \lambda\vec{v}$$

Approach
-Subtract the mean from X
-Calculate Cov(X,X)
-Calculate eigenvectors and eigenvalues of covariance matrix
-Sort the eigenvectors according to their eigenvalues in decreasing order
-Choose first k eigenvectors and that will be the new k dimensions
-Transform the original n dimensional data points into k dimensions (=Projections with dot product)

Using Iris test data from SKlearn

```
#PRINCIPAL COMPONENT ANALYSIS IN PYTHON
import numpy as np


##----- Begin PCA Implementation
class PCA:
    def __init__(self,n_components):
        self.n_components = n_components
        self.components = None
        self.mean = None

    def fit(self,X): #X is the data to be transformed
    #sort eigenvectors
    #store first n eigenvectors

        #mean
        self.mean = np.mean(X,axis=0)
```

```
        X = X - self.mean

        #covariance
        #row = 1 sample, columns=features so we T transpose
        cov = np.cov(X.T)  #feature
        #eigenvalues eigenvectors
        eigenvalues,eigenvectors = np.linalg.eig(cov) #return as col vectors
        #v[:,i]
        #sort eigenvectors
        eigenvectors = eigenvectors.T #transformed
        #sort in descendig order using -1
        idxs = np.argsort(eigenvalues)[::-1]
        eigenvalues = eigenvalues[idxs]
        eigenvectors = eigenvectors[idxs]
        #store first n eigenvectors
        self.components = eigenvectors[0:self.n_components]



    def transform(self,X):
        #project data
        X = X - self.mean
        return np.dot(X,self.components.T)
##----- End PCA Implementation

from sklearn import datasets
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split

data = datasets.load_iris()
```

```
X = data.data
y = data.target

#we have 105 samples and four features
pca = PCA(2)
pca.fit(X)
X_projected = pca.transform(X)
print('Shape of X: ', X.shape)
print('Shape of transformed X: ', X_projected.shape)

x1 = X_projected[:,0]
x2 = X_projected[:,1]
plt.scatter(x1,x2,c=y, edgecolors='none',alpha=0.8,cmap=plt.cm.get_cmap('viridis',3))
plt.xlabel('Principal Component 1')
plt.ylabel('Principal Component 2')
plt.colorbar()
plt.show()
################# end of program #################
```

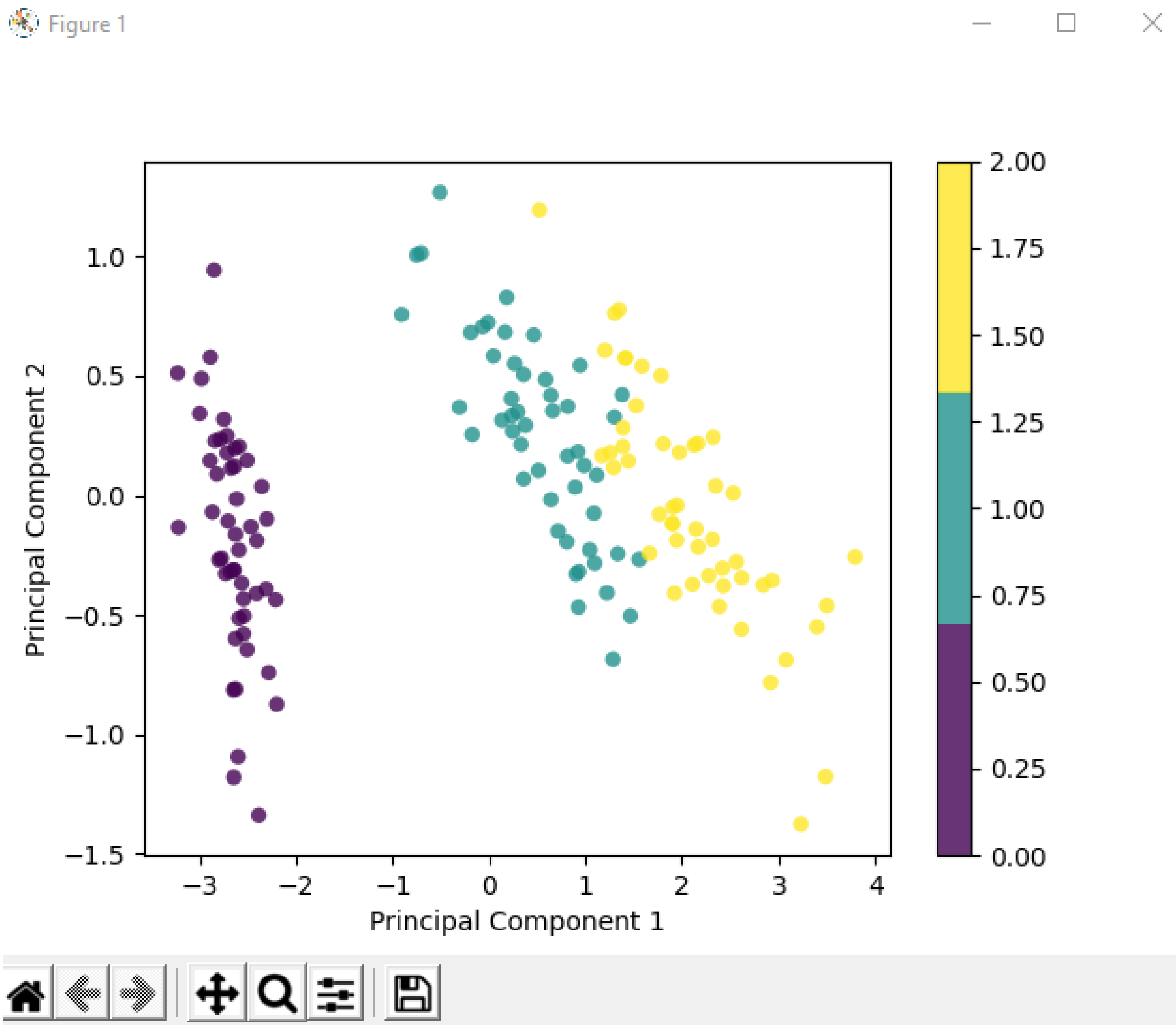

Figure 1
Principal Component 2
Principal Component 1
1.0
0.5
0.0
−0.5
−1.0
−1.5
−3
−2
−1
0
1
2
3
4
2.00
1.75
1.50
1.25
1.00
0.75
0.50
0.25
0.00