UL Mini Project

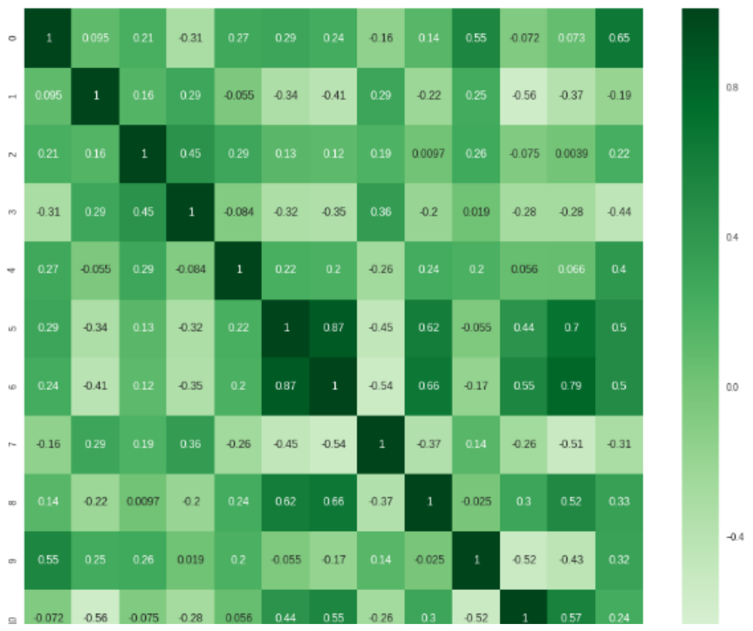
Principal Component Analysis of Wine Data

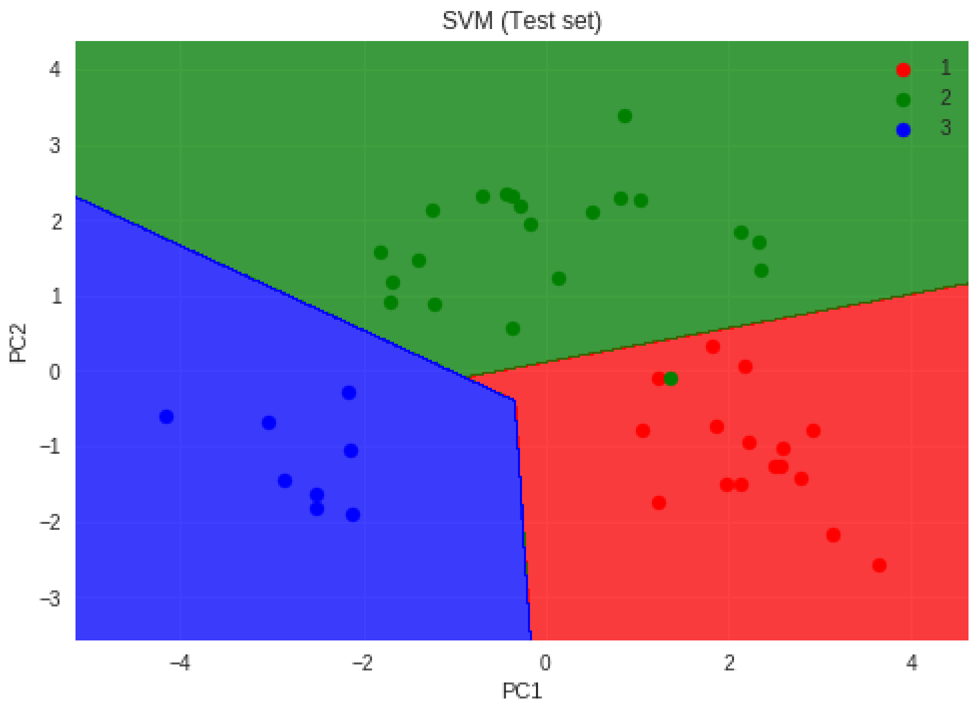
Project By:

Dalon Francis Lobo

Lokesh Sharma

Sunpreet Singh





Content:

Table of Contents

Problem Description:

Applying Principal Component Analysis (PCA) on the Wine data set.

PCA involves following broad level steps –

1. Standardize the d-dimensional dataset.
2. Construct the covariance matrix.
3. Decompose the covariance matrix into its eigenvectors and eigenvalues.
4. Select k eigenvectors that correspond to the k largest eigenvalues, where k is the dimensionality of the new   feature subspace (k≤d)
5. Construct a projection matrix W from the "top" k eigenvectors.
6. Transform the d-dimensional input dataset x using the projection matrix W to obtain the new k-dimensional feature subspace

Dataset is acquired from

https://archive.ics.uci.edu/ml/machine-learning-databases/wine/wine.data

Attribute definition

1st attribute is **class identifier (1-3)**. Other attributes are below:

1. Alcohol
2. Malic acid
3. Ash
4. Alcalinity of ash
5. Magnesium
6. Total phenols
7. Flavanoids
8. Nonflavanoid phenols
9. Proanthocyanins
10. Color intensity
11. Hue
12. OD280/OD315 of diluted wines
13. Proline

So we will consider 13 attributes for PCA.

Steps:

Step 1: Standardizing the 13-dimensional dataset

We will use the StandardScaler class from sklearn.preprocessing library to standardize the dataset.

```
In [7]: # Standardize the dataset
        sc_X = StandardScaler()
        X_std = sc_X.fit_transform(X)
        X_std.shape

Out[7]: (178, 13)
```

```
In [8]: # Display the standardized dataset
        X_std[:3, :]

Out[8]: array([[ 1.51861254, -0.5622498 ,  0.23205254, -1.16959318,  1.91390522,
                 0.80899739,  1.03481896, -0.65956311,  1.22488398,  0.25171685,
                 0.36217728,  1.84791957,  1.01300893],
               [ 0.24628963, -0.49941338, -0.82799632, -2.49084714,  0.01814502,
                 0.56864766,  0.73362894, -0.82071924, -0.54472099, -0.29332133,
                 0.40605066,  1.1134493 ,  0.96524152],
               [ 0.19687903,  0.02123125,  1.10933436, -0.2687382 ,  0.08835836,
                 0.80899739,  1.21553297, -0.49840699,  2.13596773,  0.26901965,
                 0.31830389,  0.78858745,  1.39514818]])
```

Step 2: Constructing the covariance matrix

We will use cov function from numpy module to find the covariance matrix.

```
In [9]: cov_matrix = np.cov(X_std.transpose())
        cov_matrix

Out[9]: array([[ 1.00564972,  0.09493026,  0.21273976, -0.31198788,  0.27232816,
                 0.29073446,  0.23815287, -0.15681042,  0.13747022,  0.549451  ,
                -0.07215255,  0.07275191,  0.64735687],
               [ 0.09493026,  1.00564972,  0.16497228,  0.29013035, -0.05488343,
                -0.3370606 , -0.41332866,  0.29463237, -0.22199334,  0.25039204,
                -0.56446685, -0.37079354, -0.19309537],
               [ 0.21273976,  0.16497228,  1.00564972,  0.44587209,  0.28820583,
                 0.12970824,  0.11572743,  0.1872826 ,  0.00970647,  0.2603499 ,
                -0.07508874,  0.00393333,  0.22488969],
               [-0.31198788,  0.29013035,  0.44587209,  1.00564972, -0.0838039 ,
                -0.32292752, -0.353355  ,  0.36396647, -0.19844168,  0.01883781,
                -0.27550299, -0.27833221, -0.44308618],
               [ 0.27232816, -0.05488343,  0.28820583, -0.0838039 ,  1.00564972,
                 0.21561254,  0.19688989, -0.25774204,  0.23777643,  0.20107967,
                 0.05571118,  0.06637684,  0.39557317],
               [ 0.29073446, -0.3370606 ,  0.12970824, -0.32292752,  0.21561254,
                 1.00564972,  0.86944804, -0.45247731,  0.61587304, -0.05544792,
                 0.43613151,  0.70390388,  0.50092909],
               [ 0.23815287, -0.41332866,  0.11572743, -0.353355  ,  0.19688989,
                 0.86944804,  1.00564972, -0.54093859,  0.65637929, -0.17335329,
                 0.54654907,  0.79164133,  0.49698518],
               [-0.15681042,  0.29463237,  0.1872826 ,  0.36396647, -0.25774204,
                -0.45247731, -0.54093859,  1.00564972, -0.36791202,  0.13984265,
                -0.26412347, -0.50611293, -0.31314443],
```

Step 3: Decomposing the covariance matrix into its eigenvectors and eigenvalues

To do this we will use linalg.eig from numpy module.

```
In [12]: # Converting to eigen values and eigen vectors
         eig_vals, eig_vecs = np.linalg.eig(cov_matrix)
```

```
In [13]: eigen_pairs = [(np.abs(eig_vals[i]), eig_vecs[ :, i]) for i in range(len(eig_vals))]

         # Sort the (eigenvalue, eigenvector) tuples from high to low
         eigen_pairs.sort()
         eigen_pairs.reverse()
```

```
In [14]: eigen_pairs
```

```
Out[14]: [(4.7324369775835953,
           array([-0.1443294 ,  0.24518758,  0.00205106,  0.23932041, -0.14199204,
                  -0.39466085, -0.4229343 ,  0.2985331 , -0.31342949,  0.0886167 ,
                  -0.29671456, -0.37616741, -0.28675223])),
          (2.5110809296451233,
           array([ 0.48365155,  0.22493093,  0.31606881, -0.0105905 ,  0.299634  ,
                   0.06503951, -0.00335981,  0.02877949,  0.03930172,  0.52999567,
                  -0.27923515, -0.16449619,  0.36490283])),
          (1.4542418678464692,
           array([-0.20738262,  0.08901289,  0.6262239 ,  0.61208035,  0.13075693,
                   0.14617896,  0.1506819 ,  0.17036816,  0.14945431, -0.13730621,
                   0.08522192,  0.16600459, -0.12674592])),
          (0.92416586682487556,
```

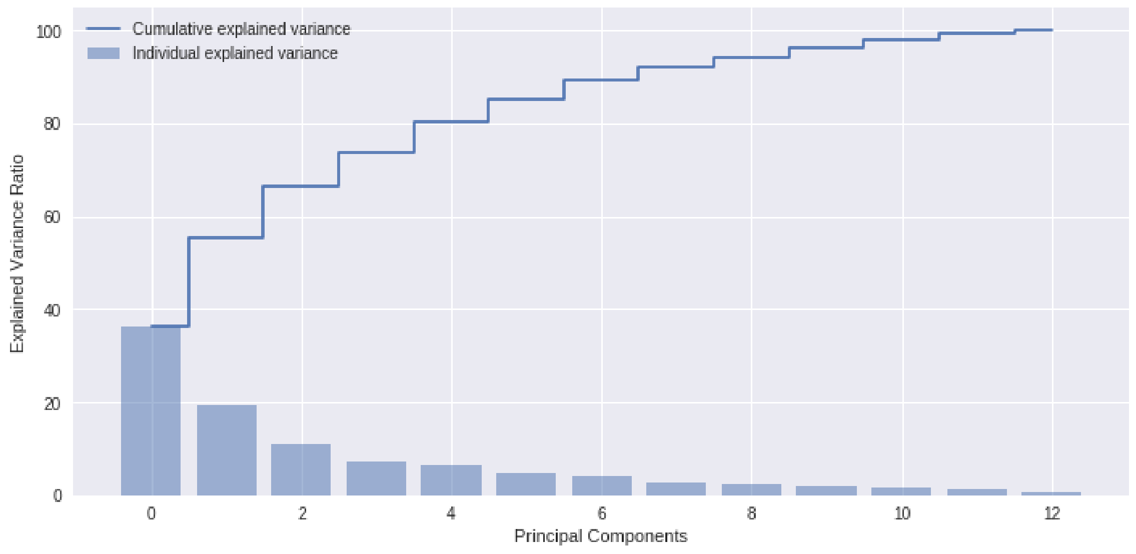
Step 4 & 5: Selecting k eigenvectors

Select k eigenvectors that correspond to the k largest eigenvalues, where k is the dimensionality of the new feature subspace (k ≤ d)

```
In [19]: 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("Cumulative Variance Explained", cum_var_exp)
```

```
Cumulative Variance Explained [  36.1988481    55.40633836   66.52996889   73.59899908   80.16229276
   85.09811607   89.3367954    92.01754435   94.23969775   96.16971684
   97.90655253   99.20478511  100.        ]
```

Cumulative graph is below:



Step 6: Projection matrix W

Constructing a projection matrix W from the "top" 7 eigenvectors.

```
In [22]: # Projection matrix
         W = eig_vecs[:, :7]
```

```
In [23]: # Display the eigen Vectors
         print("First 7 Eigen Vectors:")
         pd.DataFrame(W)
```

```
First 7 Eigen Vectors:
```

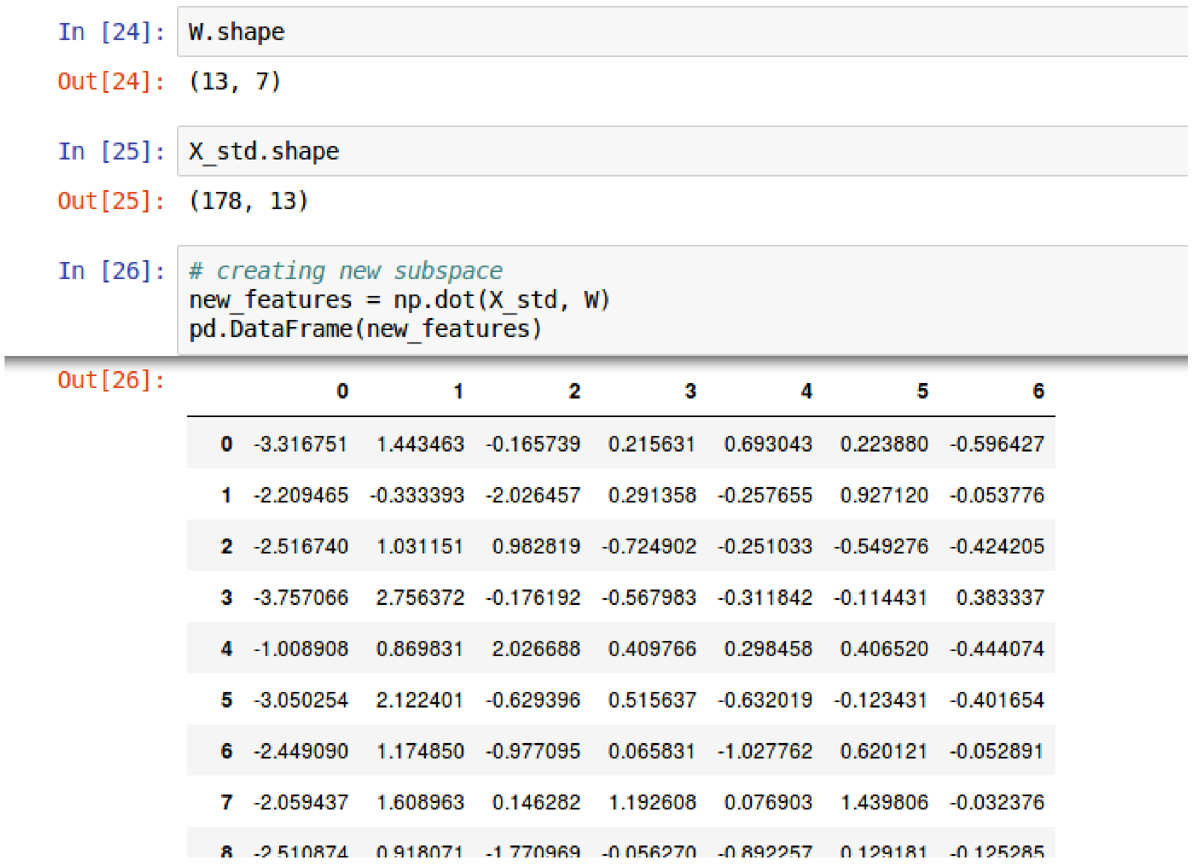
Out[23]:

	0	1	2	3	4	5	6
0	-0.144329	0.483652	-0.207383	0.017856	-0.265664	0.213539	0.056396
1	0.245188	0.224931	0.089013	-0.536890	0.035214	0.536814	-0.420524
2	0.002051	0.316069	0.626224	0.214176	-0.143025	0.154475	0.149171
3	0.239320	-0.010591	0.612080	-0.060859	0.066103	-0.100825	0.286969
4	-0.141992	0.299634	0.130757	0.351797	0.727049	0.038144	-0.322883
5	-0.394661	0.065040	0.146179	-0.198068	-0.149318	-0.084122	0.027925
6	-0.422934	-0.003360	0.150682	-0.152295	-0.109026	-0.018920	0.060685
7	0.298533	0.028779	0.170368	0.203301	-0.500703	-0.258594	-0.595447
8	-0.313429	0.039302	0.149454	-0.399057	0.136860	-0.533795	-0.372139

Here, we are reducing the 13-dimensional feature space to a 7-dimensional feature subspace, by choosing the "top 7" eigenvectors with the highest eigenvalues to construct our d×k-dimensional eigenvector matrix W.

Step 7: Transform the dataset

Transforming the 13-dimensional input dataset x using the projection matrix W to obtain the new 7-dimensional feature subspace.



```
In [24]: W.shape
Out[24]: (13, 7)

In [25]: X_std.shape
Out[25]: (178, 13)

In [26]: # creating new subspace
         new_features = np.dot(X_std, W)
         pd.DataFrame(new_features)
Out[26]:
```

	0	1	2	3	4	5	6
0	-3.316751	1.443463	-0.165739	0.215631	0.693043	0.223880	-0.596427
1	-2.209465	-0.333393	-2.026457	0.291358	-0.257655	0.927120	-0.053776
2	-2.516740	1.031151	0.982819	-0.724902	-0.251033	-0.549276	-0.424205
3	-3.757066	2.756372	-0.176192	-0.567983	-0.311842	-0.114431	0.383337
4	-1.008908	0.869831	2.026688	0.409766	0.298458	0.406520	-0.444074
5	-3.050254	2.122401	-0.629396	0.515637	-0.632019	-0.123431	-0.401654
6	-2.449090	1.174850	-0.977095	0.065831	-1.027762	0.620121	-0.052891
7	-2.059437	1.608963	0.146282	1.192608	0.076903	1.439806	-0.032376
8	-2.510874	0.918071	-1.770969	-0.056270	-0.892257	0.129181	-0.125285

Conclusion:

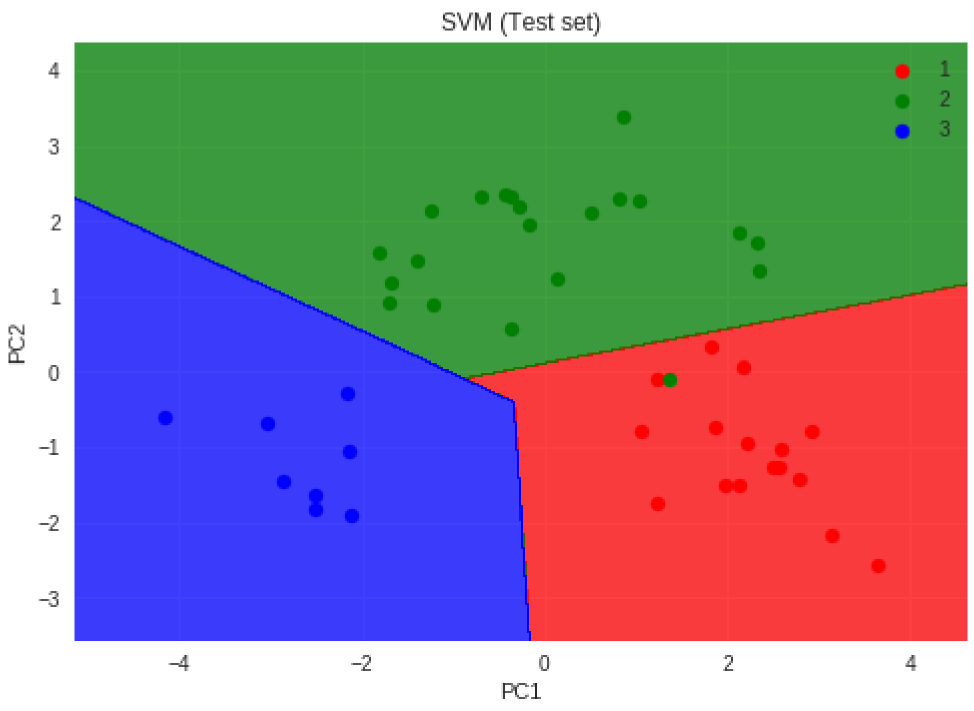
Using PCA, we were able to reduce the huge 13 dimensional dataset to 7 dimensional subspace, by retaining 89.33% of the variance of the original dataset.

We used 6 steps to apply Principal Component Analysis on our dataset.

This can also accomplished using PCA implementation from scikit-learn. Ex: from sklearn.decomposition import PCA as sklearnPCA

Constructing a classification model

- Using SVM to classify the wine dataset to 3 classes
- Applied PCA and reducing the dimentionality to just 2
- These 2 Principal Components explained total variance of 56.02% of the total variance in the dataset
- The SVM model was able to get a accuracy of 97.77%


SVM (Test set)
PC2
PC1
● 1
● 2
● 3