Viridiana Spíndola Esquivel
Rogelio Pulido Silva

PCA Algebra Steps (iris)

PCA

In this work we will explain the algebra steps of PCA applied to iris data set. In general what it is done is to reduce the dimensions of the original data set (150x4) to a dimension of (150x3), but this is not done arbitrary. We must select the component with less significance to discriminate.

PCA help us to identify patterns in data and to highlight similarities and differences, compressing the data without losing much information.

Step 1:

The data we use in this example is the iris data, this data set has dimensions 150 x 40 as shown in fig1. We can plot the data at this step, so at the end we can compare both graphs.

	X	float64	(150L, 4L)

	0	1	2	3
0	5.100	3.500	1.400	0.200
1	4.900	3.000	1.400	0.200
2	4.700	3.200	1.300	0.200
3	4.600	3.100	1.500	0.200
4	5.000	3.600	1.400	0.200
5	5.400	3.900	1.700	0.400
6	4.600	3.400	1.400	0.300
7	5.000	3.400	1.500	0.200
8	4.400	2.900	1.400	0.200
9	4.900	3.100	1.500	0.100
10	5.400	3.700	1.500	0.200

	0	1	2	3
138	6.000	3.000	4.800	1.800
139	6.900	3.100	5.400	2.100
140	6.700	3.100	5.600	2.400
141	6.900	3.100	5.100	2.300
142	5.800	2.700	5.100	1.900
143	6.800	3.200	5.900	2.300
144	6.700	3.300	5.700	2.500
145	6.700	3.000	5.200	2.300
146	6.300	2.500	5.000	1.900
147	6.500	3.000	5.200	2.000
148	6.200	3.400	5.400	2.300
149	5.900	3.000	5.100	1.800

Fig1. Iris Data set

In the code this data set corresponds to the variable X:

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

from sklearn import datasets

np.random.seed(10)

centers = [[1, 1], [-1, -1], [1, -1]]
iris = datasets.load_iris()
X = iris.data
```

Fig2. Iris Data

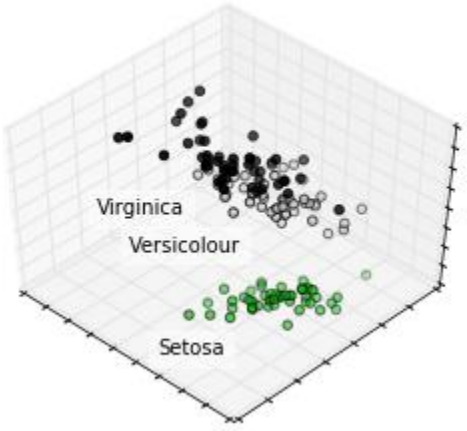


Fig3. Plot of the data

And the variable Y corresponds to the target, we can see we have 50 samples with target 0 (Setosa), 50 samples with target 1(Versicolour), and 50 samples with target 2(Viriginica).

Y	float64	(150L,)

```
Y = iris.target
```

	0
43	1.000
44	1.000
45	1.000
46	1.000
47	1.000
48	1.000
49	1.000
50	2.000
51	2.000
52	2.000
53	2.000
54	2.000

Fig4. Target

Step 2:

After we have the data, we calculate the mean of each column of the data:

```
mean_x = np.mean(X,axis=0)
```

	0
0	5.843
1	3.054
2	3.759
3	1.199

Fig5. Mean of iris data

The next step is to subtract the mean from the data and have a new variable with the new data set with a mean equal to zero. This step will allow us to do the next steps.

```
nX = X - mean_x
nXT = nX.T
```

Fig6. Calculation of the mean and its transpose.

Step 3:

The third step is to calculate the covariance matrix. Since we are multiplying a matrix (4x150) and a matrix (150x4). We expect a matrix of dimensions (4x4)

nXT	float64	(4L, 150L)
nX	float64	(150L, 4L)

```
cov_mtx = np.dot(nXT,nX)/(X.shape[0]-1)
```

	0	1	2	3
0	0.686	-0.039	1.274	0.517
1	-0.039	0.188	-0.322	-0.118
2	1.274	-0.322	3.113	1.296
3	0.517	-0.118	1.296	0.582

Fig7. Covariance matrix

Step 4:

From the covariance matrix we calculate its correspondent eigen values and eigen vectors, we can do this step since the covariance matrix is square. The eigen values and eigen vectors will allows us to know useful information about our data.

```
eigen_val_mtx, eigen_vect_mtx = np.linalg.eig(cov_mtx)
```

Fig8. Calculate eigen values and eigen vectors from covariance matrix

	0
0	4.225
1	0.242
2	0.079
3	0.024

Fig9. Eigen values

eig_vecs	float64	(4L, 4L)

	0	1	2	3
0	0.362	-0.657	-0.581	0.317
1	-0.082	-0.730	0.596	-0.324
2	0.857	0.176	0.073	-0.480
3	0.359	0.075	0.549	0.751

Fig10. Eigen vectors

Once we have done this, we are going to proceed to reduce the dimensions of the data. In order to do this, we must order the eigen values, from higher to lower, so we can discriminate the component with less significance. Doing this will help us to have a final data set with less dimensions than the original one.

```
cov_mtx2 = eigen_vect_mtx[:,[0,1,2]]
```

ncov_mat	float64	(4L, 3L)

	0	1	2
0	0.362	-0.657	-0.581
1	-0.082	-0.730	0.596
2	0.857	0.176	0.073
3	0.359	0.075	0.549

Fig11. One dimension less.

Step 5:

Finally, we are creating the final data set that is the multiplication of the transpose of the new matrix of one dimension less (3x4) and the transpose of the original data set (4x150). What we expect is a matrix of dimensions 3x150 that will correspond to our final data set.

```
new_data = np.dot(cov_mtx2.T,nXT)
```

new_data	float64	(3L, 150L)

	0	1	2	3	4	...	145	146	147	148	149
0	-2.684	-2.715	-2.890	-2.746	-2.729		1.944	1.526	1.764	1.902	1.390
1	-0.327	0.170	0.137	0.311	-0.334		-0.187	0.375	-0.079	-0.116	0.283
2	-0.022	-0.204	0.025	0.038	0.096		0.179	-0.121	0.131	0.723	0.362

Fig12. New data set

In order to plot this data, we have to obtain its transpose to have the correct dimensions (150x3).

```
new_dataT = new_data.T
```

new_dataT	float64	(150L, 3L)

	0	1	2
0	-2.684	-0.327	-0.022
1	-2.715	0.170	-0.204
2	-2.890	0.137	0.025
3	-2.746	0.311	0.038
4	-2.729	-0.334	0.096
5	-2.280	-0.748	0.174
6	-2.821	0.082	0.264
7	-2.626	-0.170	-0.016
8	-2.888	0.571	0.027
9	-2.674	0.107	-0.192
10	-2.507	-0.652	-0.069

...

140	2.314	-0.183	0.323
141	1.922	-0.409	0.115
142	1.414	0.575	0.296
143	2.563	-0.276	0.291
144	2.419	-0.304	0.504
145	1.944	-0.187	0.179
146	1.526	0.375	-0.121
147	1.764	-0.079	0.131
148	1.902	-0.116	0.723
149	1.390	0.283	0.362

Fig13. Transpose of new data set.

Finally we obtain the plot

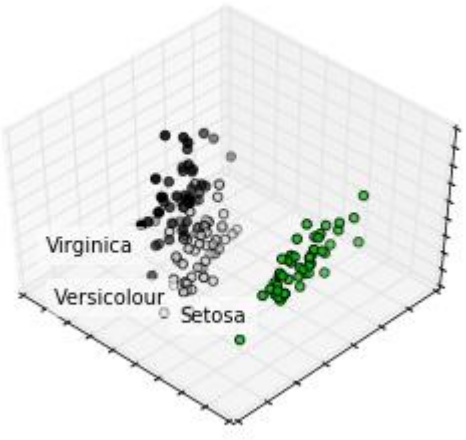
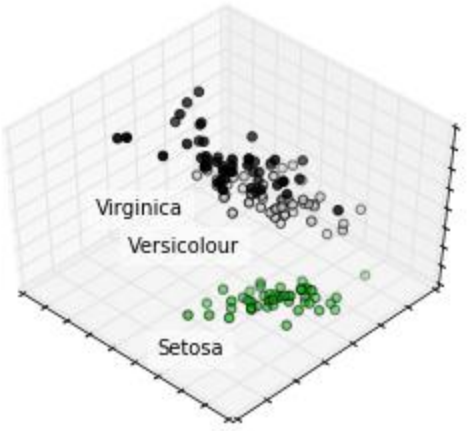
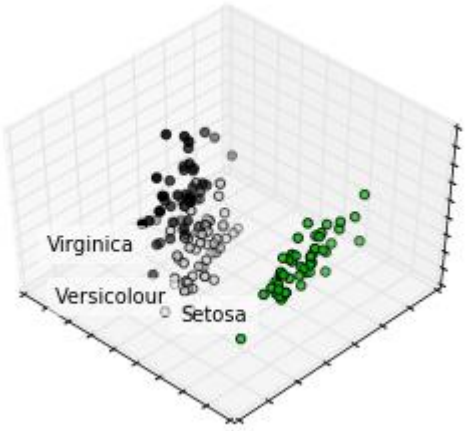


Fig14. Plot of the new data

Conclusions



a)



b)

Fig15. Comparison between a) plot of original data set and b) plot of new data set after PCA

From both scatter plots we can observe the difference before applying the PCA and after applying it. The main difference, we can see is that Virginica and Versicolour are more separate from each other after applying PCA. We can conclude that after reducing dimensions, we don't lose a lot of information and we can still interpret data.

APPENDIX A

Complete code

```
# -*- coding: utf-8 -*-
"""
Created on Tue May 03 20:30:01 2016

@author: Spindola
"""

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

from sklearn import datasets

np.random.seed(10)

centers = [[1, 1], [-1, -1], [1, -1]]
iris = datasets.load_iris()
X = iris.data
Y = iris.target

fig = plt.figure(1, figsize=(4, 3))
plt.clf()
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)

plt.cla()
mean_x = np.mean(X,axis=0)
nX = X - mean_x
nXT = nX.T

cov_mtx = np.dot(nXT,nX)/(X.shape[0]-1)
eigen_val_mtx, eigen_vect_mtx = np.linalg.eig(cov_mtx)

cov_mtx2 = eigen_vect_mtx[:,[0,1,2]]

new_data = np.dot(cov_mtx2.T,nXT)

new_dataT = new_data.T


for name, label in [('Setosa', 0), ('Versicolour', 1), ('Virginica', 2)]:
    ax.text3D(new_dataT[Y == label, 0].mean(),
              new_dataT[Y == label, 1].mean() + 1.5,
              new_dataT[Y == label, 2].mean(), name,
              horizontalalignment='center',
              bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))

Y = np.choose(Y, [1, 2, 0]).astype(np.float)
ax.scatter(new_dataT[:, 0], new_dataT[:, 1], new_dataT[:, 2], c=Y, cmap=plt.cm.spectral)

#for name, label in [('Setosa', 0), ('Versicolour', 1), ('Virginica', 2)]:
   # ax.text3D(X[Y == label, 0].mean(),
              #X[Y == label, 1].mean() + 1.5,
             # X[Y == label, 2].mean(), name,
              #horizontalalignment='center',
              #bbox=dict(alpha=.5, edgecolor='w', facecolor='w'))

#Y = np.choose(Y, [1, 2, 0]).astype(np.float)
#ax.scatter(X[:, 0], X[:, 1], X[:, 2], c=Y, cmap=plt.cm.spectral)

ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])
plt.figure(1)

plt.show()
```

References

Smith, Lindsay I. "A tutorial on principal components analysis." Cornell University, USA 51.52 (2002).

Pedregosa, F. (2014) "Sklearn.decomposition.PCA." Scikit Learn De Scikit. Available: http://scikit-learn.org/stable/modules/generated/sklearn.decomposition.PCA.html.

Principal Component Analysis, Available:http://support.sas.com/publishing/publicat/chaps/55)