IE0 529 Compu Hw1

Chen Yuhao

NetID:yuhaoc2

**Q0.**  
a. The original matrix is a 3\*200 sized matrix.

Conducting singular value decomposition on the original matrix gives the singular value matrix:

|  |  |
| --- | --- |
|  | 4.80431456e+01 |
|  | 7.64023841e+00 |
|  | 1.27870577e-14 |

The unitary matrix containing the left-singular vectors is in size 3\*3

U=

|  |  |  |
| --- | --- | --- |
| -6.27971189e-01 | 3.25041822e-01 | -7.07106781e-01 |
| -4.59678553e-01 | -8.88085372e-01 | -2.77555756e-17 |
| -6.27971189e-01 | 3.25041822e-01 | 7.07106781e-01 |

The first three right eigenvectors in are

|  |  |  |
| --- | --- | --- |
|  |  |  |
| -1.85953419e-02 | -3.37191547e-02 | 4.91754744e-02 |
| 2.51828637e-03 | -4.25091757e-02 | -3.53990364e-02, |
| 9.92733588e-01 | -1.13087085e-03 | -3.99746130e-03 |

Since the singular value for the 3rd variable is extremely close to 0, we can expect that the true component needed to represent the data is 2. We should set the PCA parameter to reduce the dimension of the original data to 2.

b. The mean of the three variables are

|  |
| --- |
| 1.8727525 |
| 1.48782644 |
| 1.8727525 |

Following the directions of debiasing the original matrix then calculate the covariance matrix and normalizing it with the constant 1/(3-1), we will have the normalized covariance matrix

C=

|  |  |  |
| --- | --- | --- |
| 107.46917577 | 46.08182007 | 107.46917577 |
| 46.08182007 | 45.51705296 | 46.08182007 |
| 107.46917577 | 46.08182007 | 107.46917577 |

The sorted (in descending order) eigenvalue and sorted eigenvector is recorded in the following tables:

|  |  |
| --- | --- |
| Eigenvalue | |
|  | 2.38297449e+00 |
|  | 2.34667765e-01 |
|  | 1.11972991e-16 |

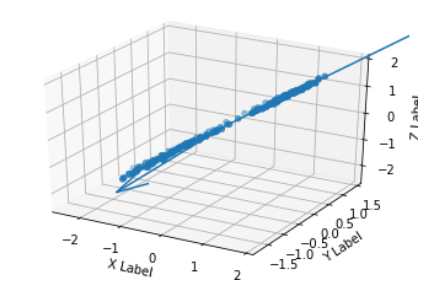
|  |  |  |  |
| --- | --- | --- | --- |
| Eigenvector |  |  |  |
| 0.66943821 | 0.22771139 | 7.07106781e-01 |
| 0.32203254 | -0.946728601 | 1.24496500e-16 |
| 0.66943821 | 0.22771139 | -7.07106781e-01 |

c. The 1st principal component is the eigenvector corresponding to the biggest eigenvalue, so the 1st and 2nd principal component equal to

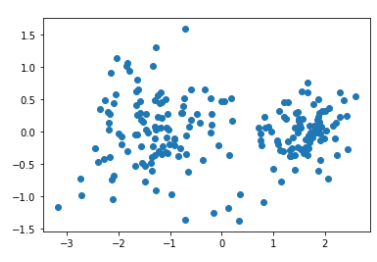
|  |  |
| --- | --- |
|  |  |
| 0.66943821 | 0.22771139 |
| 0.32203254 | -0.946728601 |
| 0.66943821 | 0.22771139 |

We can plot the debiased data with respect to the two principal components in 3D plot

The two vectors shown in the plot that will make up the new coordinate system.



We can also plot the projection of the original data points with the dimension reduced from 3 to 2, on the plane made up by the two principal components.



3. Furthermore, we can derive the eigenvalues in the covariance matrix with the singular values of the debiased original matrix. Suppose the debiased original matrix is denotes as X

Thus, and having the same eigenvalues, so the eigenvalues of covariance matrix can be related to the singular value of the origin matrix :

We run an SVD on the debiased scaled data and the predicted eigenvalue in covariance matrix to be shown in the following table

|  |  |  |  |
| --- | --- | --- | --- |
| Singular Value of debiased matrix | 2.17764075e+01 | 6.83365826e+00 | 6.83371307e-15 |
| Predicted eigenvalue in covariance matrix | 2.38E+00 | 2.35E-01 | 2.35E-31 |

Which is almost the same to the eigenvalue of the covariance matrix calculated in part 2 though the smallest value is even closer to 0, the reason will be shown in the following paragraph

The left singular vector of debiased X is the orthonormal eigenvector of , thus we expect

the same result:

Eigenvalue decomposition:

array([[ 6.69438214e-01, 2.27711392e-01, 7.07106781e-01], [ 3.22032538e-01, -9.46728601e-01, 1.10903003e-15], [ 6.69438214e-01, 2.27711392e-01, -7.07106781e-01]])

SVD Left Singular result:

array([[-6.69438214e-01, 2.27711392e-01, -7.07106781e-01], [-3.22032538e-01, -9.46728601e-01, 6.93889390e-17], [-6.69438214e-01, 2.27711392e-01, 7.07106781e-01]])

So the SVD result will set up the same plane as that by components in eigenvectors of the covariance matrix.

The difference between the choice of SVD on origin matrix and eigenvalue decomposition on the covariance matrix lies in “the fact that on a computer real numbers are represented with finite precision. Since each number is stored in a finite amount of memory there is necessarily a gap between consecutive representable numbers. For double precision floating point numbers, for instance, the relative error between a real number and its closest floating point approximation is on the order of . Algorithms that take this limitation into account are called backward stable The basic idea is that such an algorithm produces the correct output for an input value that’s within of the value you actually asked about”(<https://intoli.com/blog/pca-and-svd/>).

If we are to find relatively small singular values, we should conduct SVD directly on the origin matrix so that we will get a far more accurate result rather than conduct eigenvalue decomposition on the covariance matrix.

**Q1.**

We denote as the data for longest shot-put(meter), as the maximum lift(kg).

1. We can calculate the optimal for the polynomial regression

with the Henkel Matrix

And

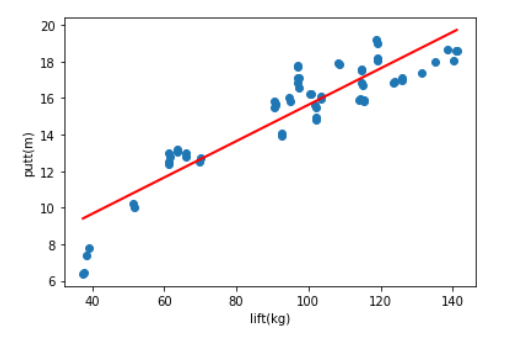
And optimal

The program in Python is attached in the Appendix

When the order : we have the following result

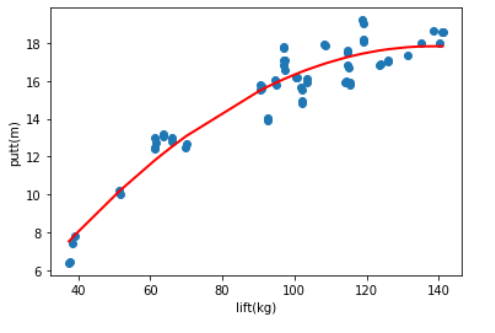
The regression result is shown in this plot

Since the SSR value is extremely big, order is not good enough.



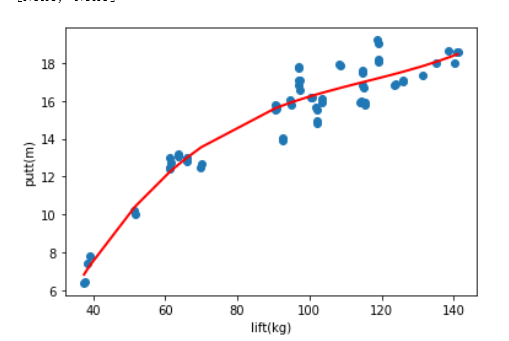
Plot for

When the order q=2,



Plot for

When the order q=3,



Plot for

If q=4, the plot will be extremely close to the one for and has the following parameters.

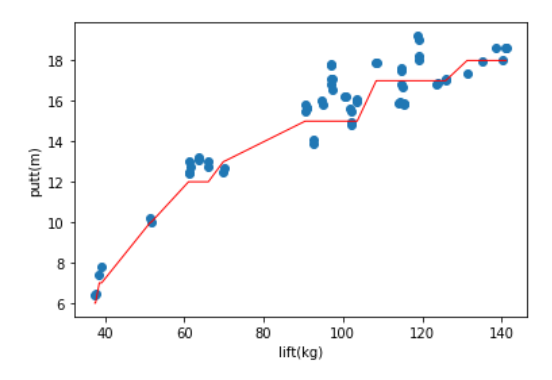
Therefore, order no smaller than 3 will have a very close goodness-of-fit but the calculation cost will increase significantly.

A reasonably well-fit order will be based on the standard of an R-squared value bigger than 0.9, we can conclude that an order q no less than 2 will give fit the data reasonably well. There will be a trade-off between goodness-of-fit and calculation cost, if we are focusing on goodness-of-fit, we choose order , else, if we would like to significantly cut down the calculation cost, we choose order .

It is not quite reasonable to use a logistic regression model on this data set. The reason is that the longest shot-put in meter is not a categorical data, but a variable with different magnitude. The major problem on using the logistic regression is on how to categorize the data to logistic results 0 and 1. It is definitely not reliable to simply categorizing the variable 0 and 1.

We are setting a multinomial logistic regression since the independent variable has several different values. Therefore, the manipulation on the data is taking the integer of them. Then use the “logisticregression” function in the “sklearn.linear\_model” Python package.

The logistic regression is better than a simple linear regression but no better than a polynomial regression.



Plot for Multinomial Logistic Regression

3/4.

My opinion simply depends on the focus of the problem, if the problem involves an extremely largest data set and requires quick calculation and efficiency, I will recommend a polynomial linear regression with order . It has offered a creditable goodness-of-fit on the data sample (R-square is bigger than 0.9). The R-square result is far better than a simple linear regression and the logistic regression. Meanwhile, it has the lowest order in the polynomial linear regression models.

Else, if we are focusing on the accuracy of the goodness-of-fit, I will prefer to use a polynomial linear regression order , which provides high goodness-of-fit and lower complexity of the system..

Python code for the polynomial regression(Written in Jupyter Notebook)

import numpy as np

import pandas as pd

from scipy.linalg import hankel

from numpy.linalg import inv

import matplotlib.pyplot as plt

data=pd.read\_csv('Comp1\_IE529.csv',header=None)

X=data[0].values

Y=data[1].values

avgy=np.mean(Y)

SStotal=sum(np.power(Y-avgy,2))

q=1 #order of the polynomial regression

H1=[]

H2=[]

Ybar=[]

i=0

for i in range(0,q+1):

x=sum(np.power(X,i))

y=sum(np.power(X,i)\*Y)

H1.append(x)

Ybar.append(y)

i+=1

for i in range(q,2\*q+1):

x=sum(np.power(X,i))

H2.append(x)#H2 is the last row of the hankel matrix

i+=1

H=hankel(H1,H2)

beta=np.matmul(inv(H),Ybar)

mid=[]

ypredict=[]

for j in range(0,len(X)):

mid=[]

for i in range(0,len(beta)):

yp=(np.power(X[j],i))\*beta[i]

mid.append(yp)

y=sum(mid)

ypredict.append(y)

SSres=sum(np.power(Y-ypredict,2))

Rsquare=1-SSres/SStotal

plt.scatter(X,Y)

plt.xlabel('lift(kg)')

plt.ylabel('putt(m)')

results=plt.plot(np.sort(X),np.sort(ypredict))

plt.setp(results, color='r', linewidth=2.0)

Python code for logistic regression

regr=LogisticRegression(solver='lbfgs', multi\_class='multinomial',max\_iter=100000,C=1000.0,tol=0.01)

Y\_int=Y.astype(int)

XX=[X]

XXX=np.array(XX).reshape(-1,1)

YY=[Y\_int]

YYY=np.array(YY).reshape(-1,1)

regr.fit(XXX,YYY)

Y\_pred=regr.predict(XXX)

Y\_pred

idx=np.argsort(X)

plt.scatter(X,Y)

plt.xlabel('lift(kg)')

plt.ylabel('putt(m)')

#results=plt.plot(XXX[idx],(Y\_pred[idx]))

results=plt.plot(XXX[idx],(Y\_pred[idx]))

plt.setp(results, color='r', linewidth=1.0)

SSres=sum(np.power(Y-Y\_pred,2))

Rsquare=1-SSres/SStotal

print(SSres,Rsquare)

**Q2.**

1. The package used is sklearn.decomposition.PCA

The PCA process starts with preprocessing the data. In the first part, we debias the data with the column mean. In part 4, we do this by standardizing the data, using the Function “StandardScaler” from the package “sklearn.preprocessing”. The function works like: StandardScaler().fit\_transform(data) and the input is the data read from the csv file. It standardize the data feature onto unit scale (mean=0, standard deviation=1, like standard normal distribution) as the output.

Then we use the function sklearn.decomposition.PCA. For example, we can set the desired reduced dimension to 2. Use the function pca.fit(data).transform(data) to output the projection on the first two principal components.

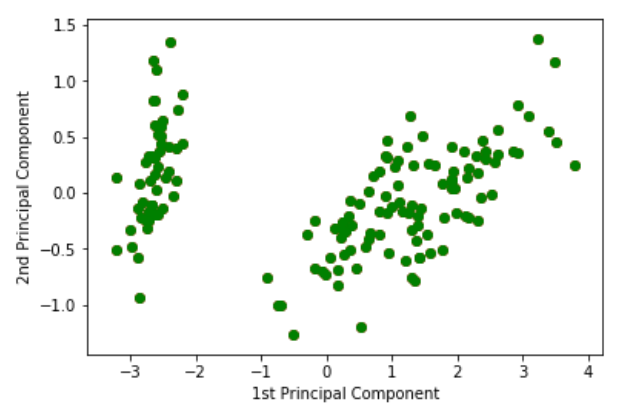
We can use the function PCA(components desired). explained\_variance\_ratio\_ to describe the portion of the variance contained in these components. We will use explained\_variance\_ to the corresponding variance respect to that principal component.

1. We use the explained\_variance\_ratio and explained\_variance\_ attribute in PCA, the result is shown in the following table:

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | No.1 PC | No.2 PC | No.3 PC | No.4 PC |
| Ratio explained | 92.461621% | 5.301557% | 1.718514% | 0.518309% |
| Corresponding Variance | 4.22484077 | 0.24224357 | 0.07852391 | 0.02368303 |

Since 1st and 2nd principal component can explain more than 97% of the variance, we believe 2 true components are needed to explain the data.

3. We calculate the projection onto the plane formed by the first two principal components with the attribute pca.fit(debiased).transform(debiased). And use the matplotlib.pyplot.scatter function to generate the plot shown below:



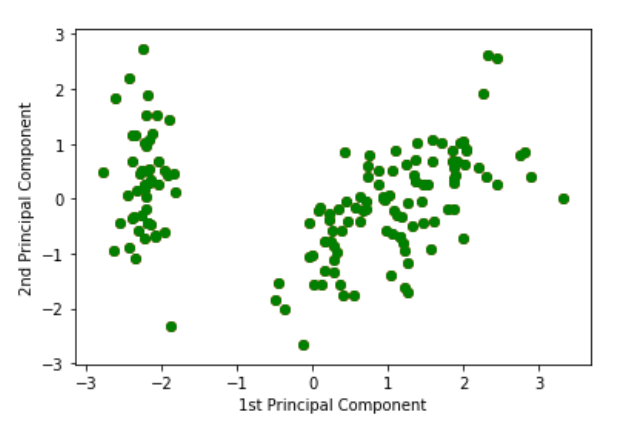
There is a visually obvious cluster in the plot, such as using a vertical line on as separation, to the left of the line is one cluster, and to the right another. The big cluster on the right may be further divided, but we have no more information currently. Therefore at least two types of flowers in the dataset.

We input the standardized data with StandardScaler().fit\_transform(data)

We don’t reduce any dimension, and use the explained\_variance\_ratio attribute to decide how many true components do we need to keep. The variance portion explained in the standardized data case is different from the debiased data case.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | No.1 PC | No.2 PC | No.3 PC | No.4 PC |
| Ratio explained | 72.770452% | 23.030523% | 3.6838325% | 0.515193% |
| Corresponding Variance | 2.93035378 | 0.92740362 | 0.14834223 | 0.02074601 |

Since the variance explained by the 3rd and 4th principal component is less than 4.5%, we believe that 1st and 2nd principal component will be needed to represent data. Although the portion value and corresponding variance value changes from the debiased input case, the conclusion is the same.



As we can see in the plot, the value of the projected value changed from the debiased input case.

Again, there is only 1 visually extremely obvious cluster, such as the cluster to the left of the line between . We still cannot further divide the big cluster on the right. So, there are at least two species of flowers in this dataset.

Python code:

import numpy as np

import pandas as pd

from scipy.linalg import hankel

from sklearn.decomposition import PCA

from numpy.linalg import inv

import matplotlib.pyplot as plt

from sklearn.preprocessing import StandardScaler

data=pd.read\_csv('PCA\_comp1.csv',header=None)

mean=pd.array(data.mean(axis=0))

np.transpose(mean)

meantominus=pd.DataFrame(np.outer(np.ones((data.shape[0],)),mean))

debiased=data-meantominus

debiased=np.array(debiased)

pca1=PCA(n\_components=4)#the pca fitter for the standardize data

standard=StandardScaler().fit\_transform(data)

data\_lowdimension=pca1.fit(standard).transform(standard)

ax=plt.figure()

for c,i in zip('rgb',[0,1]):

plt.scatter(data\_lowdimension[:,0],data\_lowdimension[:,1],c=c)

plt.xlabel('1st Principal Component')

plt.ylabel('2nd Principal Component')

pca1.explained\_variance\_ratio\_

pca1.explained\_variance\_

pca2=PCA(n\_components=2)#debiased bata input

data\_lowdimension=pca2.fit(debiased).transform(debiased)

ax=plt.figure()

for c,i in zip('rgb',[0,1]):

plt.scatter(data\_lowdimension[:,0],data\_lowdimension[:,1],c=c)

plt.xlabel('1st Principal Component')

plt.ylabel('2nd Principal Component')

pca2.explained\_variance\_ratio\_

pca2.explained\_variance\_