Homework 7.1

import pandas as pd

import numpy as np

import sklearn.preprocessing as pre

from sklearn.discriminant\_analysis import LinearDiscriminantAnalysis,QuadraticDiscriminantAnalysis

from sklearn.linear\_model import LogisticRegression

from sklearn.metrics import confusion\_matrix

from sklearn.neighbors import KNeighborsClassifier

from sklearn.model\_selection import train\_test\_split, cross\_val\_score

from sklearn.naive\_bayes import GaussianNB

from sklearn.preprocessing import scale

from sklearn.decomposition import PCA

import matplotlib.pyplot as plt

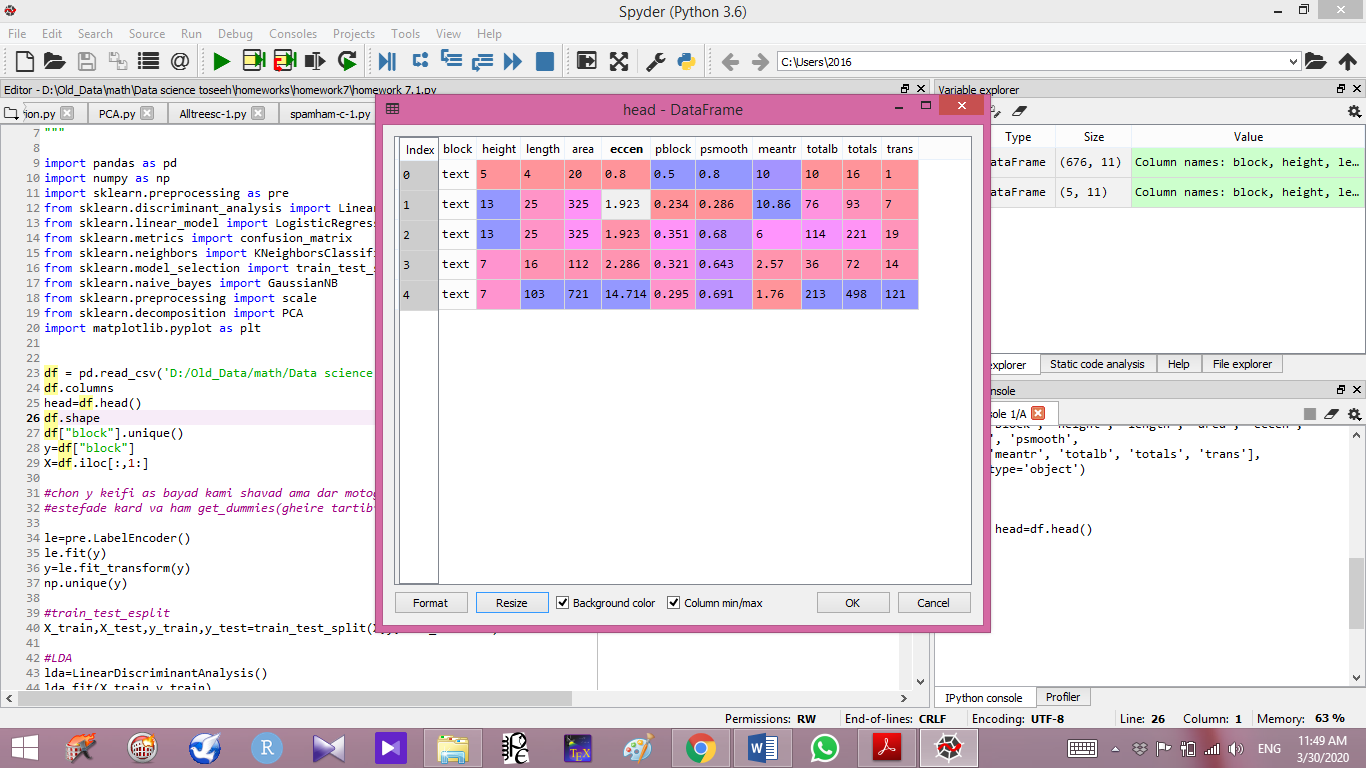
df = pd.read\_csv('D:/Old\_Data/math/Data science toseeh/Files/blocks.csv')

df.columns

output: 'block', 'height', 'length', 'area', 'eccen', 'pblock', 'psmooth',

'meantr', 'totalb', 'totals', 'trans

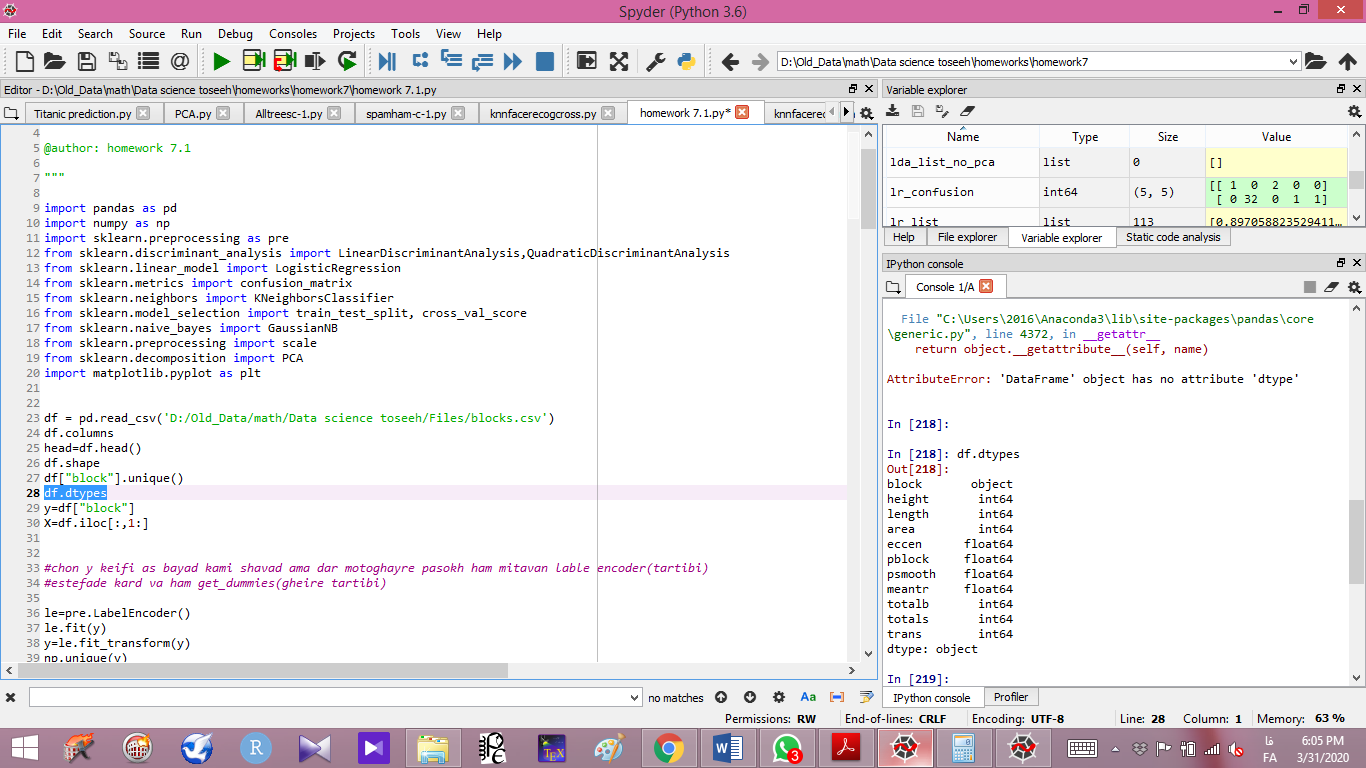
head=df.head()



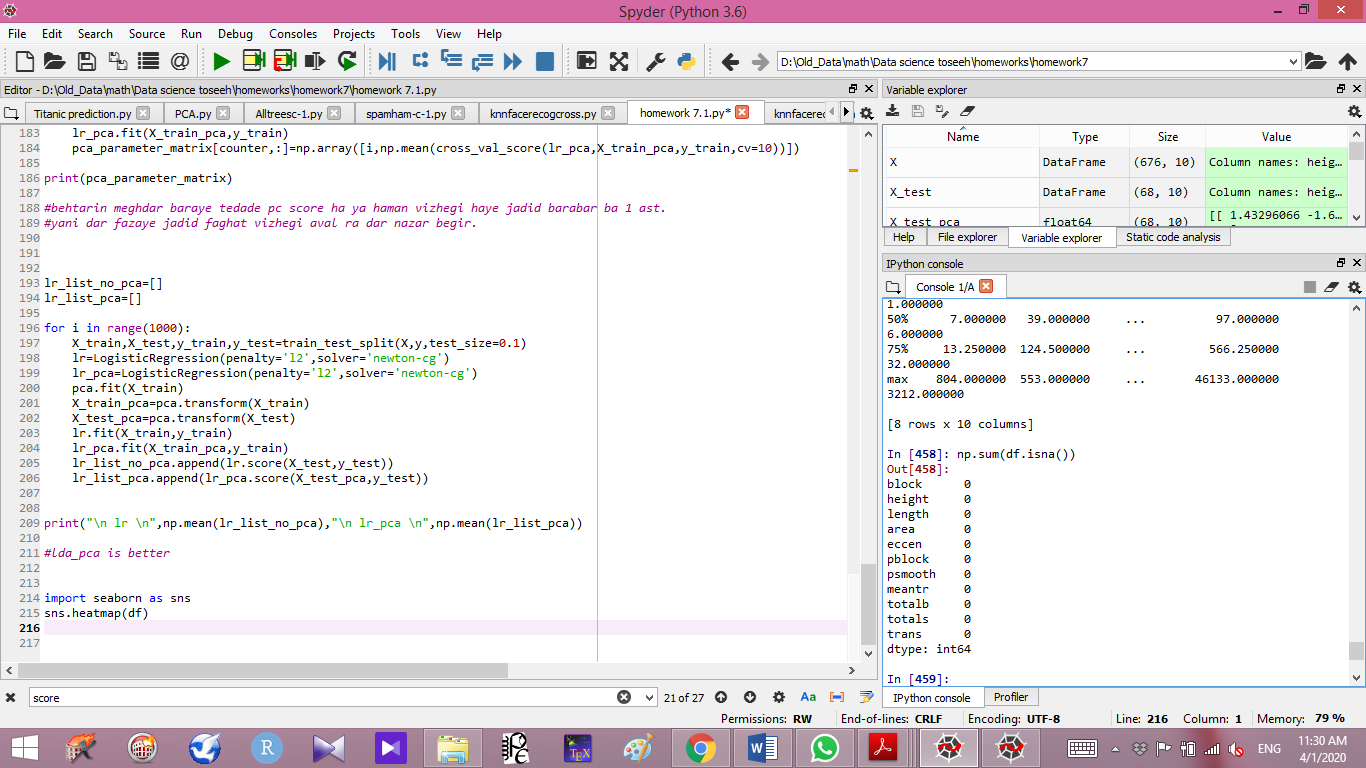
df.shape

output: (676,11)

df.dtypes



np.sum(df.isna())



تنها ویژگی که کیفی است ویژگی اول است که باید تغییر لبیل بشه.

y=df["block"]

X=df.iloc[:,1:]

چون متغیر پاسخ کیفی است باید مشخص کنیم که با یک متغیر کیفی داریم کار میکنیم. برای متغیر های کیفی که دارای ترتیب هستند باید از lableencoder استفاده کنیم و برای غیر ترتیبی ها از get\_dummies استفاده میکنیم.

اما در مورد متیغر پاسخ فرقی ندارن که کدام را به کار بریم.چون فقط میخواهیم این متغیر را پیش بینی کنیم اما متغیر های مستقل برای توصیف به کار میروند.

le=pre.LabelEncoder()

le.fit(y)

y=le.fit\_transform(y)

np.unique(y)

output: array([0, 1, 2, 3, 4], dtype=int64)

a)

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.1)

#LDA

lda=LinearDiscriminantAnalysis()

lda.fit(X\_train,y\_train)

#QDA

qda=QuadraticDiscriminantAnalysis()

qda.fit(X\_train,y\_train)

#Naive Bayes

nb=GaussianNB()

nb.fit(X\_train,y\_train)

#logistic regression

lr=LogisticRegression(penalty='l2',solver='newton-cg')

lr.fit(X\_train,y\_train)

روش knn دارای پارامتر تعداد همسایه هاست که برای یافتن این پارمتر از کراس ولیدیشن استفاده میکنیم. یعنی داده اموزشی را به 10 تکه تقسیم میکنیم. یک قسمت را ولیدیشن در نظر گرفته و بقیه را اموزشی( این کار برای این است که تا میتوانیم دچار اورفیت شدن نشیم. چون اور فیت یعنی خطای پیش بینی روی اموزشی تقریبا صفر باشد اما روی داده تست خطا داشته باشیم. اینجا چون هر بار ویلدیشن از خود داده اموزشی انتخاب میشه و در پایان خطا میشه میانگین کل خطا ها پس داریم از اورفیت شدن دوری میکنیم).

#KNN

knn=KNeighborsClassifier(n\_neighbors=1,metric='minkowski',p=2)

knn.fit(X\_train,y\_train)

در تمرین گفته شده برای یافتن پارامتر فقط یک بار حلقه for را انجام دهیم پس داریم:

knn=KNeighborsClassifier(n\_neighbors=1,metric='minkowski',p=2)

np.mean(cross\_val\_score(knn,X\_train,y\_train,cv=10))])

output: 0.8750333874104814

این میزان دقت خوب است و میتوان با همین تعداد همسایه پیش رفت اما برای اطمینان که بهترین تعداد همسایه برابر با یک است، حلقه را برای تعداد همسایه ها بین یک تا جذر تعداد نمونه ها انجام میدهم.

knnmatrix=np.empty((26,2))

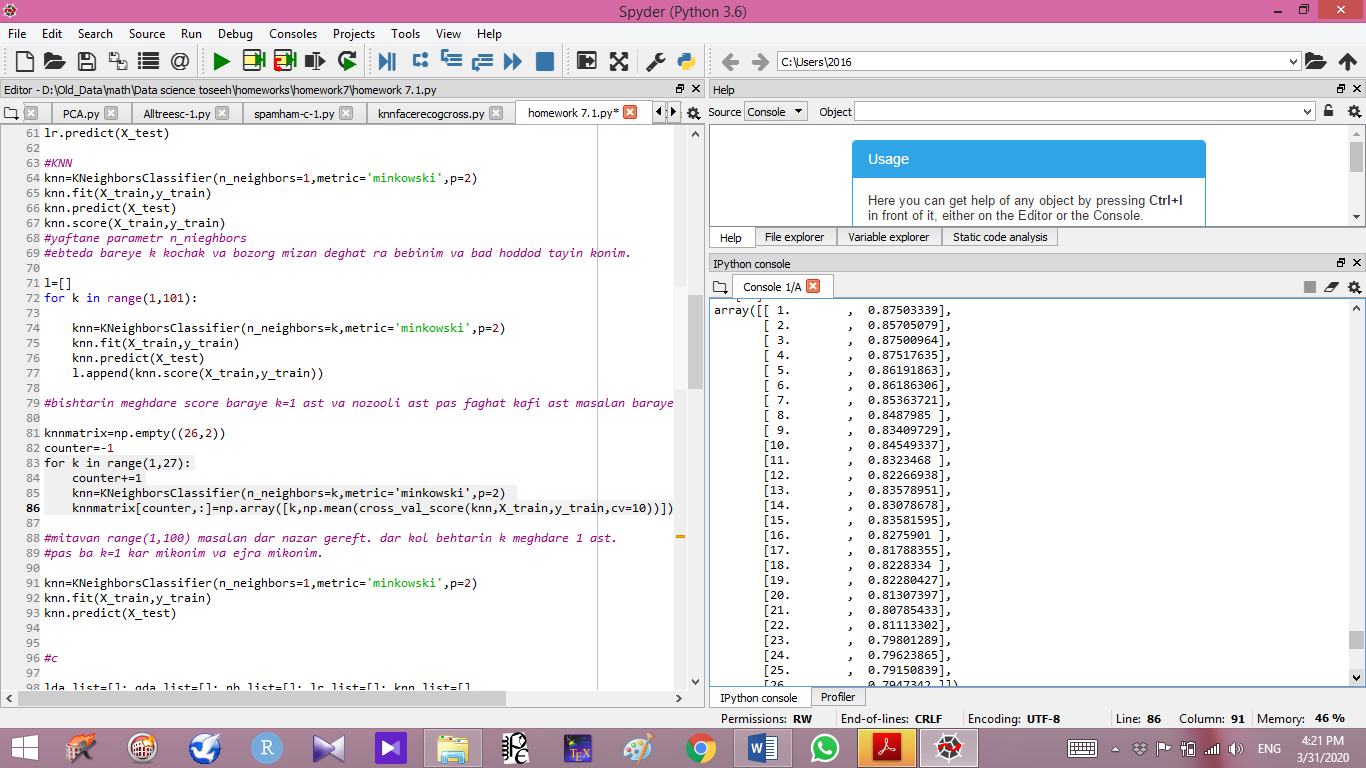
counter=-1

for k in range(1, 26):

counter+=1

knn=KNeighborsClassifier(n\_neighbors=1,metric='minkowski',p=2)

knnmatrix[counter,:]=np.array([k,np.mean(cross\_val\_score(knn,X\_train,y\_train,cv=10))])



بهترین میزان دقت کراس مربوط به k=1 است.

knn=KNeighborsClassifier(n\_neighbors=1,metric='minkowski',p=2)

C)

در این قسمت توجه شود تمام روش ها قبلا فراخوانی شده اند (knn با k=1 فراخوانی شده است).

ابتدا برای هر یک از روش ها یک لیست خالی تعریف میکنیم. یک حلقه 1000 تایی مینویسیم و هر بار داده را به دو قسمت اموزشی و تست تقسیم میکنیم و روی داده اموزشی روش ها را فیت میکنیم. میزان دقت روی داده تست (یعنی چه درصد از داده تست درست پیش بینی شده) ا محاسبه میکنیم و به لیست مربوط به ان روش اضافه میکنیم. در پایان (خارج از حلقه) میزان میزان این دقت ها را برای هر روش حساب میکنیم و با هم مقایسه میکنیم.

lda\_list=[]; qda\_list=[]; nb\_list=[]; lr\_list=[]; knn\_list=[]

for i in range(1000):

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.1)

lda.fit(X\_train,y\_train)

qda.fit(X\_train,y\_train)

nb.fit(X\_train,y\_train)

lr.fit(X\_train,y\_train)

knn.fit(X\_train,y\_train)#tavajoh k=1 farze shode.

lda\_list.append(lda.score(X\_test,y\_test))

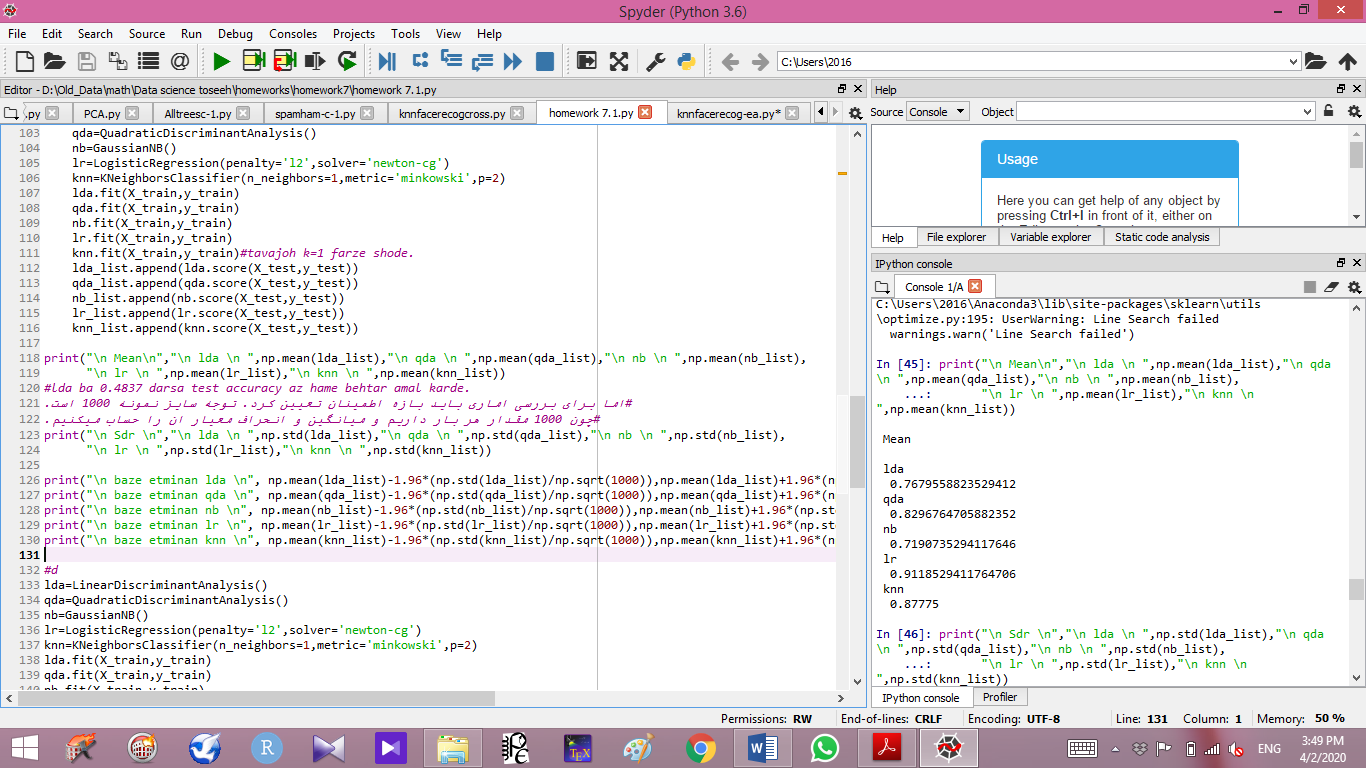
qda\_list.append(qda.score(X\_test,y\_test))

nb\_list.append(nb.score(X\_test,y\_test))

lr\_list.append(lr.score(X\_test,y\_test))

knn\_list.append(knn.score(X\_test,y\_test))

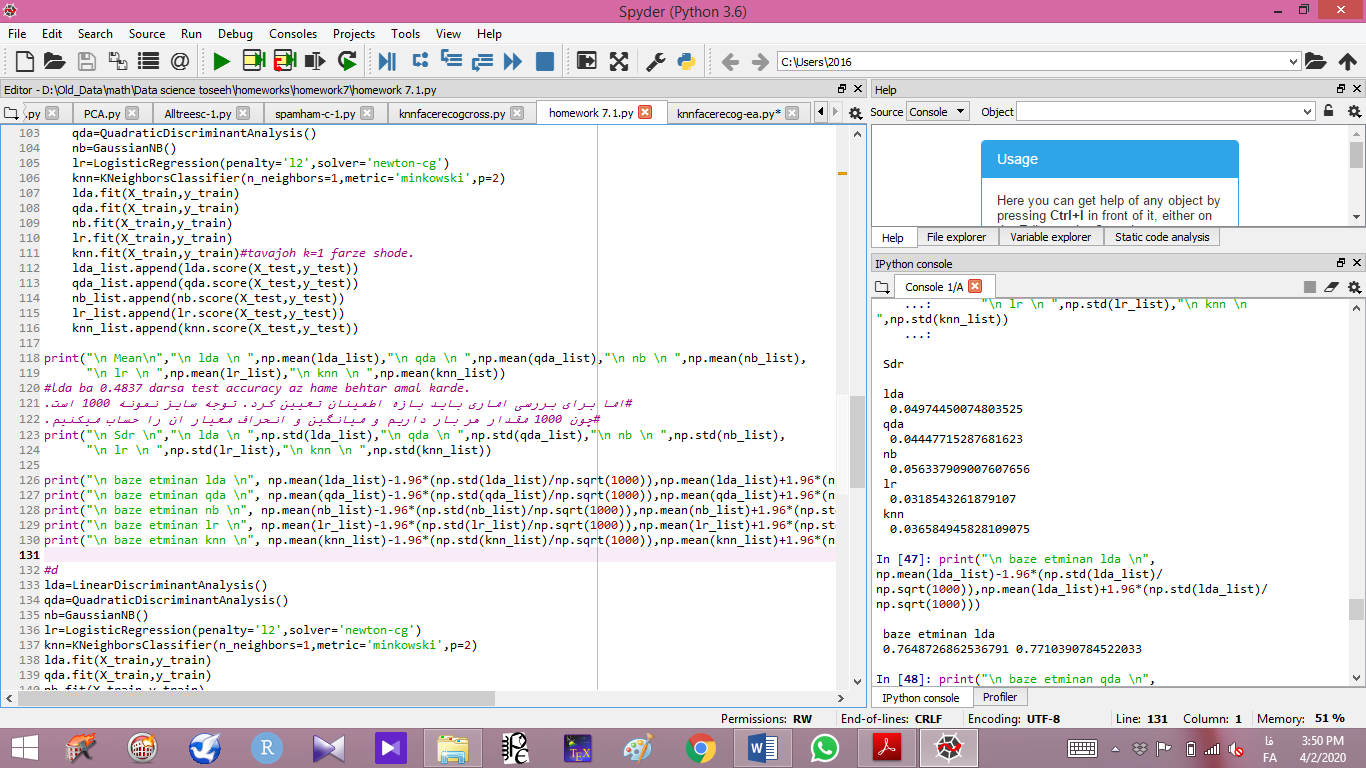
print("\n lda \n ",np.mean(lda\_list),"\n qda \n ",np.mean(qda\_list),"\n nb \n ",np.mean(nb\_list),"\n lr \n ",np.mean(lr\_list),"\n knn \n ",np.mean(knn\_list)



روش رگرسیون لاجیستیک از همه بهتر عمل میکند و دقت تست خوبی دارد ولی برای اطمینان باید بازه اطمینان تعیین کنیم تا از نظر اماری این مقادیر را بررسی کنیم

.

print("\n baze etminan lda \n", np.mean(lda\_list)+1.96\*(np.std(lda\_list)/np.sqrt(1000)),np.mean(lda\_list)+1.96\*(np.std(lda\_list)/np.sqrt(1000)))



baze etminan lda

0.7648726862536791

0.7710390784522033

به همین صورت بازه اطمینان بقیه روشها را حساب میکنیم.

baze etminan qda

0.8269197480924023

0.8324331930840682

baze etminan nb

0.7155816696346876

0.7225653891888416

baze etminan lr

0.9098785895844292

0.913827292768512

baze etminan knn

0.8754824415649424

0.8800175584350577

همانطور که میبینیم هیچ یک از بازه ها اشتراک ندارند پس میتوان گفت تفاوت ان ها معنادار است و روش رگرسیون لاجیستیک از همه بهتر عمل میکند.

d)

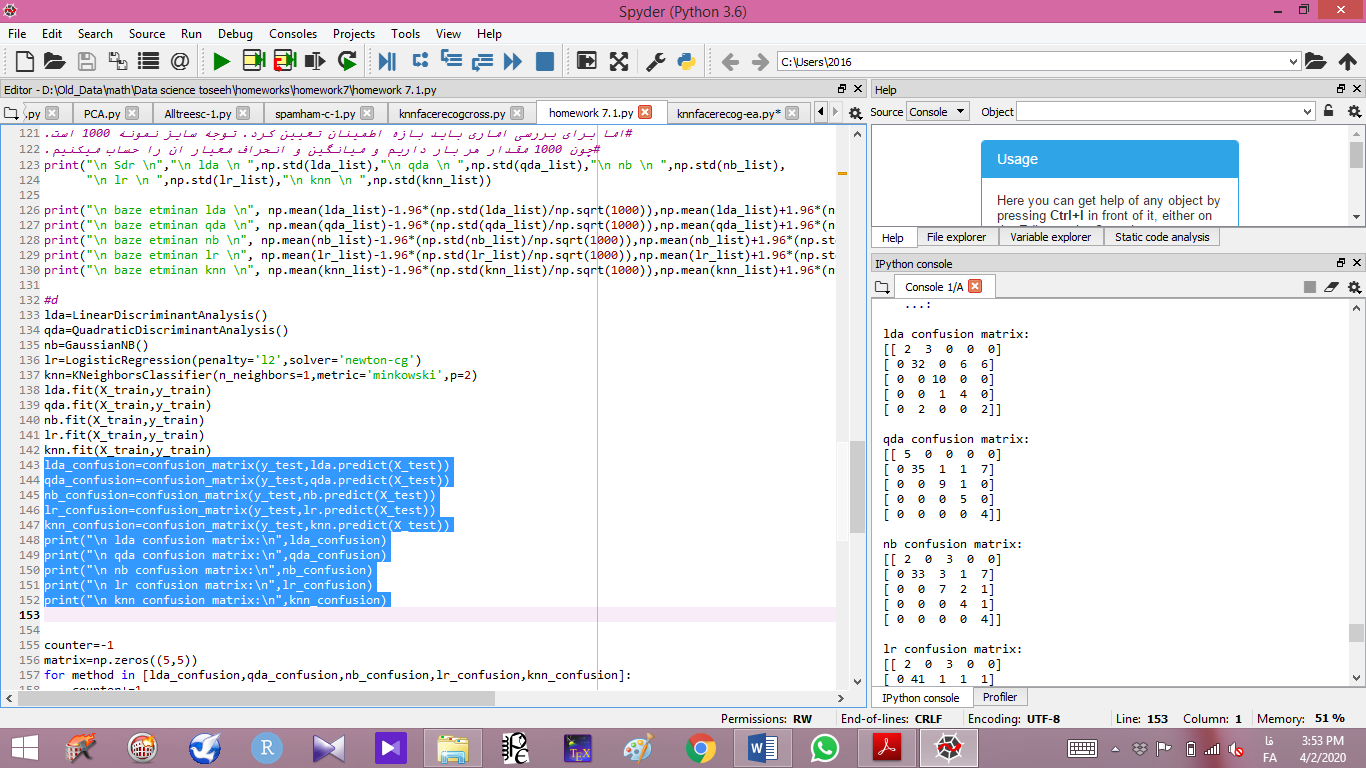
lda\_confusion=confusion\_matrix(y\_test,lda.predict(X\_test))

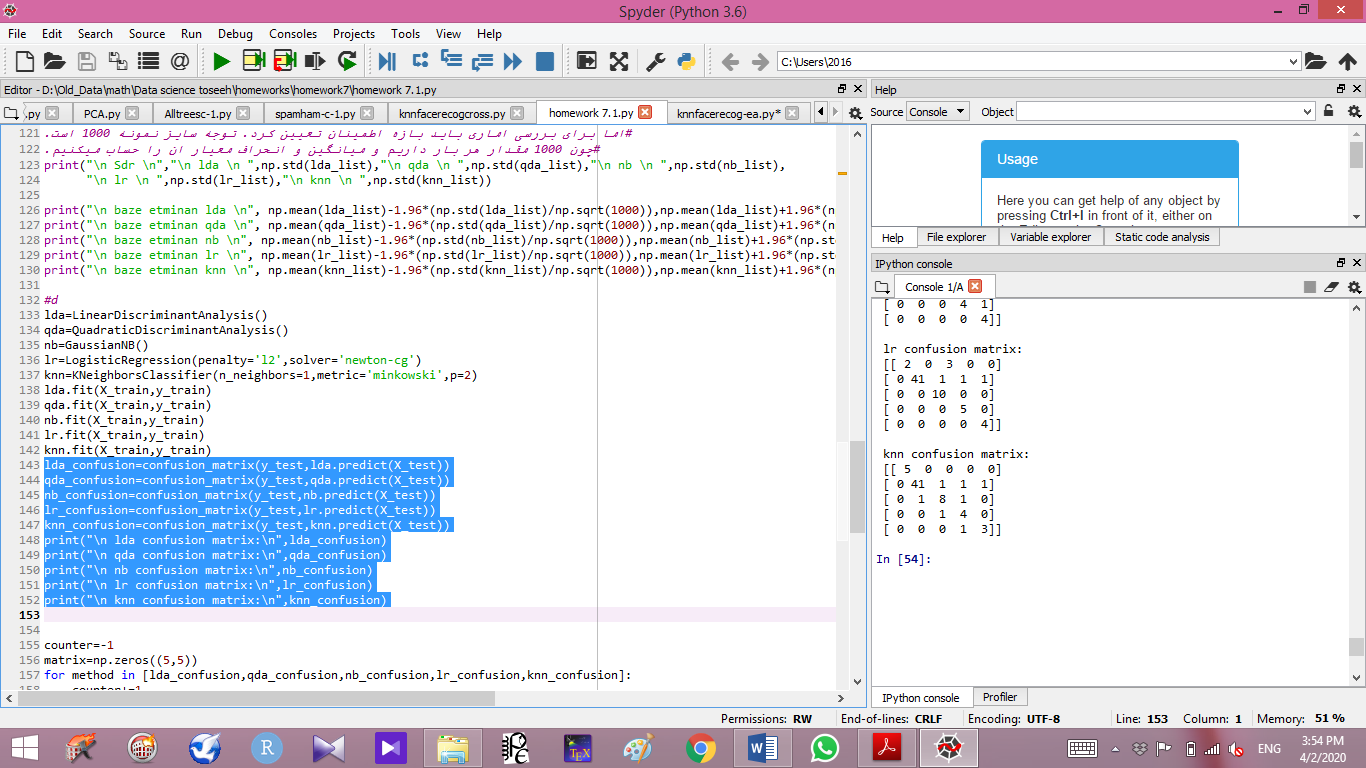
qda\_confusion=confusion\_matrix(y\_test,qda.predict(X\_test))

nb\_confusion=confusion\_matrix(y\_test,nb.predict(X\_test))

lr\_confusion=confusion\_matrix(y\_test,lr.predict(X\_test))

knn\_confusion=confusion\_matrix(y\_test,knn.predict(X\_test))



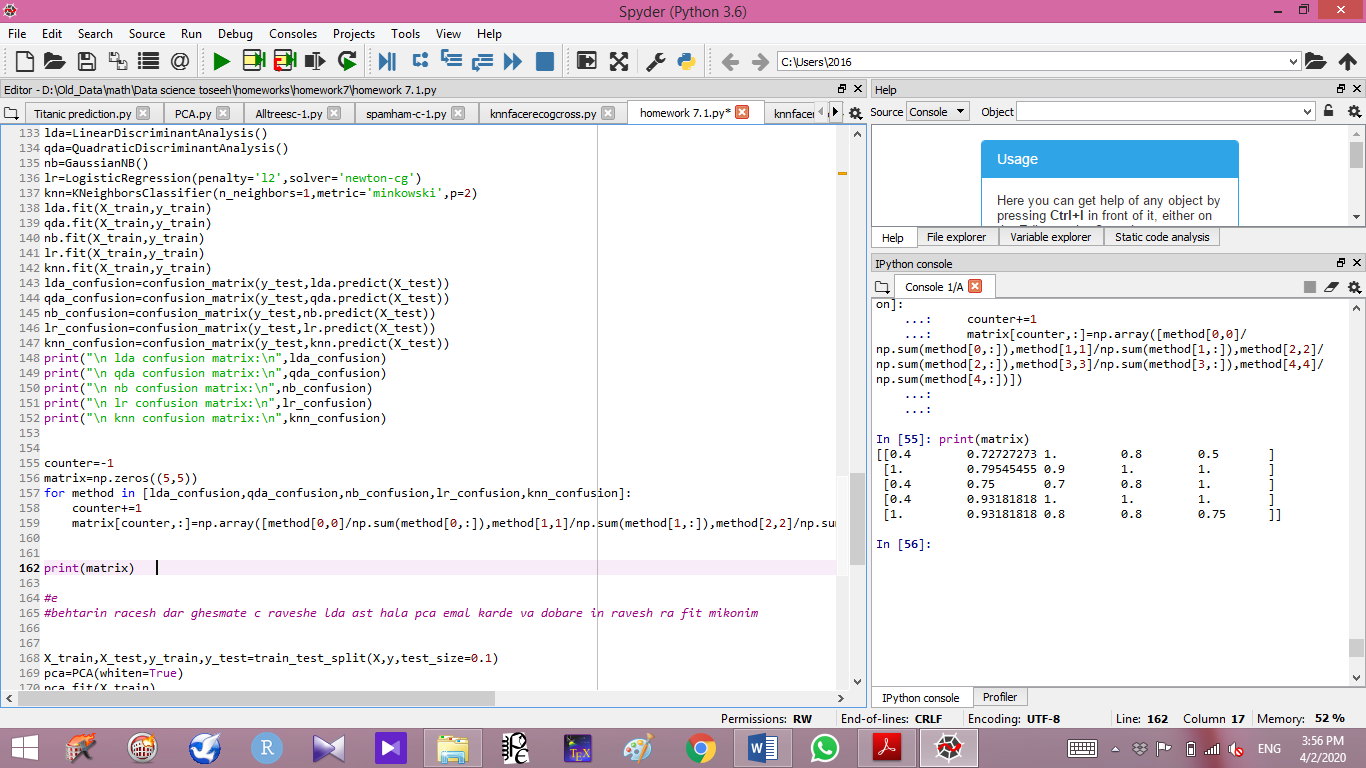


counter=-1

matrix=np.zeros((5,5))

for method in [lda\_confusion,qda\_confusion,nb\_confusion,lr\_confusion,knn\_confusion]:

counter+=1 matrix[counter,:]=np.array([method[0,0]/np.sum(method[0,:]),method[1,1]/np.sum(method[1,:]),method[2,2]/np.sum(method[2,:]),method[3,3]/np.sum(method[3,:]),method[4,4]/np.sum(method[4,:])])



به دنبال روشی هستیم که در تمام کلاسها خوب عمل کرده باشد.

میانگین میزان درستی کلاس ها در هر روش :

Lda=( 0.4+0.72+1+0.8+0.8)/5=0.74 Qda:0.938

Nb:0.73 lr: 0.866 knn:0.85

البته اگر بازه اطمینان حساب کنیم دو روش qda و knn که در همه کلاسها تقریبا بهتر عمل کرده اند اشتراک بازه اطمینان دارند.

درایه (0و0) ماتریس نشان دهنده این است که در روش ال دی ای 0.4 از داده های اولین کلاس از ستون بلوک درست پیش بینی شده اند و درایه مثلا (3,3)نشان دهنده این است که تمام داده های کلاس 4 در روش رگرسیون لاجیستیک درست پیش بینی شده است. ان روشی از همه بهتر است که در تمام کلاس ها خوب عمل کرده باشد. روش qda از lda بهتر عمل کرده اما این روش در مقایسه با knn در بعضی کلاس ها مثل کلاس اول خیلی خوب عمل کرده اما بعضی کلاس ها را هم بد پیش بینی کرده. به نطر من روش qda از همه بهتر عمل کرده.

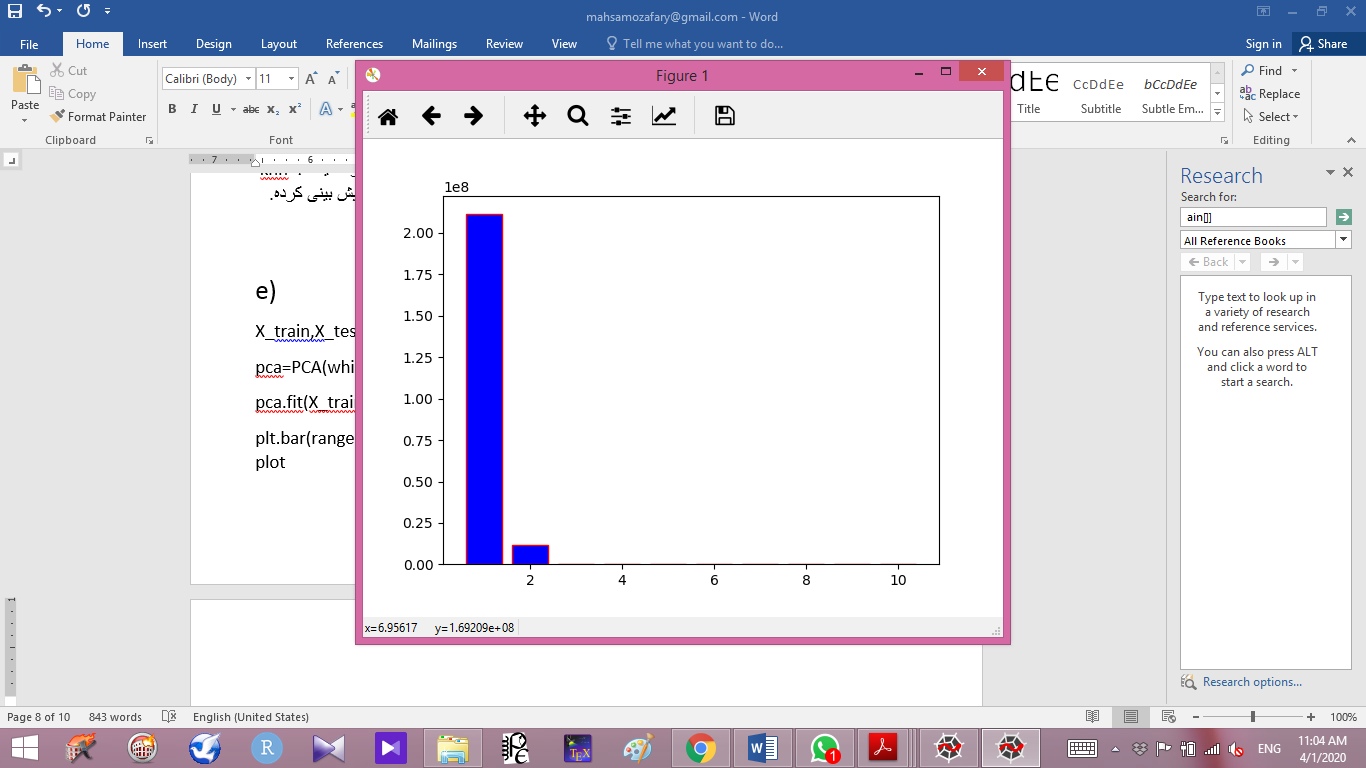
e)

ابتدا مجموعه اموزشی و تست را جدا میکنیم (که قبلا جدا شده) و بعد استاندارد کردن را انجام میدیم. چون در غیر این صورت وقتی pca را روی مجموعه اموزشی فیت میکنیم استاندارد سازی مجموعه اموزشی با میانگین کل داده ها (اموزشی و تست) انجام میشه در حالیکه باید با میانگین و انحراف معیار خود مجموعه استاندارد سازی انجام شود.

pca=PCA(whiten=True)

pca.fit(X\_train)

plt.bar(range(1,11),pca.explained\_variance\_,color="blue",edgecolor="Red")#scree plot



با توجه به نمودار اگر 2 ویژگی اول بیشترین تغییر پذیری را دارند اما ما به دنبال تعداد ویژگی هایی هستیم که بتوانند تفکیک پذیری خوبی از پاسخ دهند به همین دلیل از **کراس** استفاده میکنیم.

بازه را 1 تا 11 میگیرم چون 10 ویژگی داریم.

lr\_pca=LogisticRegression(penalty='l2',solver='newton-cg')

pca\_parameter\_matrix=np.zeros((10,2))

counter=-1

for i in range(1,11):

counter+=1

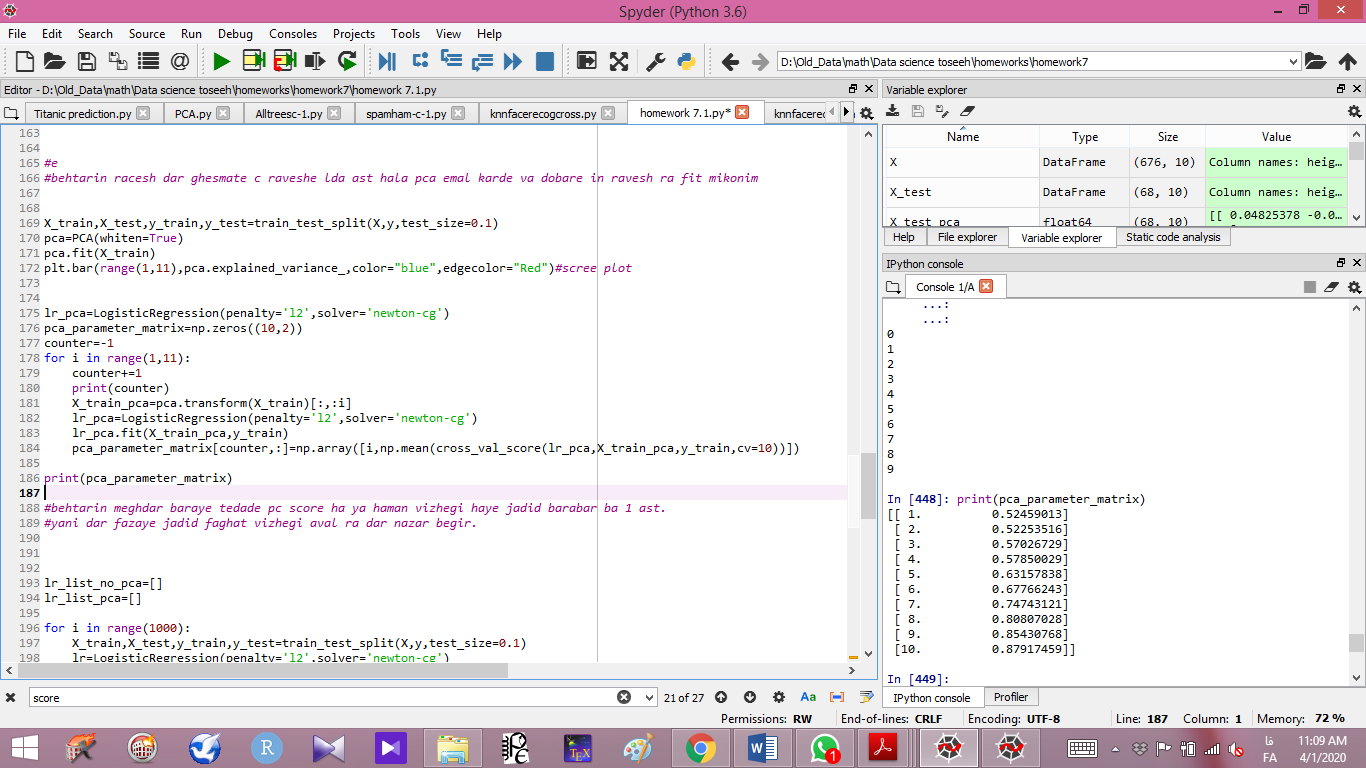
X\_train\_pca=pca.transform(X\_train)[:,:i]

lr\_pca=LogisticRegression(penalty='l2',solver='newton-cg')

lr\_pca.fit(X\_train\_pca,y\_train)

pca\_parameter\_matrix[counter,:]=np.array([i,np.mean(cross\_val\_score(lr\_pca,X\_train\_pca,y\_train,cv=10))])

print(pca\_parameter\_matrix)



بنابر کراس بهتر است تمام ویژگی ها را در نظر بگیریم.

lr\_list\_no\_pca=[]

lr\_list\_pca=[]

for i in range(1000):

X\_train,X\_test,y\_train,y\_test=train\_test\_split(X,y,test\_size=0.1)

lr=LogisticRegression(penalty='l2',solver='newton-cg')

lr\_pca=LogisticRegression(penalty='l2',solver='newton-cg')

pca.fit(X\_train)

X\_train\_pca=pca.transform(X\_train)

X\_test\_pca=pca.transform(X\_test)

lr.fit(X\_train,y\_train)

lr\_pca.fit(X\_train\_pca,y\_train)

lr\_list\_no\_pca.append(lr.score(X\_test,y\_test))

lr\_list\_pca.append(lr\_pca.score(X\_test\_pca,y\_test))

print("\n lr \n",np.mean(lr\_list\_no\_pca),"\n \r\_pca \n",np.mean(lr\_list\_pca))

lr

0.9094117647058825

lr\_pca

0.871014705882353

روش رگرسیون بدون انجام پی سی ای دقت تست بیشتری دارد.