**import** scipy.io **as** sio  
**import** sklearn.model\_selection **as** sms  
**import** sklearn.naive\_bayes **as** snb  
**import** sklearn.neighbors **as** sn  
**import** numpy **as** np  
**import** matplotlib.pyplot **as** plt  
**import** matplotlib.colors **as** mc  
**import** math  
  
  
*#1*olivetti = datasets.fetch\_olivetti\_faces()  
data = olivetti.data  
target = olivetti.target max = 0  
**for** i **in** range(1, 6):  
 pca = PCA(n\_components=i)  
 Xr = pca.fit(data).transform(data)  
 print(**'wsp. wyj. war dla '**,i,**' komp. : '**, pca.explained\_variance\_ratio\_.sum())  
 **if** max < pca.explained\_variance\_ratio\_.sum():  
 max = pca.explained\_variance\_ratio\_.sum()  
 max\_index = i  
print(**'Najlepszy wynik uzyskano dla'**, max\_index, **'komponentów.'**)  
*#1  
  
  
#2*mnist = datasets.load\_digits()  
train, test, train\_targets, test\_targets = train\_test\_split(mnist.data, mnist.target.ravel(), test\_size=0.50, random\_state=42)  
*#2  
  
  
#3*max = 0  
max\_index = 0  
**for** i **in** range(1, 10):  
 lda = LDA(n\_components=i)  
 X\_r = lda.fit(train, train\_targets).transform(train)  
 Y\_r = lda.fit(test, test\_targets).transform(test)  
 clf = neighbors.KNeighborsClassifier(round(math.sqrt(len(train))), weights=**'uniform'**, metric=**'euclidean'**)  
 clf.fit(X\_r, train\_targets)  
 print(**'Wynik dla '**,i,**' komp: '**, clf.score(Y\_r, test\_targets))  
 **if** max < clf.score(Y\_r, test\_targets):  
 max = clf.score(Y\_r, test\_targets)  
 max\_index = i  
print(**'najlepszy wynik dla'**, max\_index, **'cech.'**)  
*#3  
  
  
#4*dataSet = sklearn.datasets.load\_digits()  
data = dataSet[**"data"**]  
target = dataSet[**"target"**] plsca = PLSC(n\_components = 2)  
plsca.fit(data,target) X\_train\_r,Y\_train\_r = plsca.transform(data,target)  
knn = math.sqrt(len(X\_train\_r))  
knn = KNC(n\_neighbors = int(knn))  
Y\_train\_r = [int(Y\_train\_r[i])foriinrange(0,len( Y\_train\_r))]  
k = knn.fit(X\_train\_r,Y\_train\_r)  
print(k.score(X\_train\_r,Y\_train\_r))  
*#4  
  
  
#5*knn = KNeighborsClassifier(n\_neighbors = 4)  
sfs = SFS(knn, k\_features = 3, forward = **True**, floating = **False**, verbose = 2, scoring = **'accuracy'**, cv = 0)  
*#5  
  
  
#6***with** open(**'arcene\_train.data'**) **as** f: raw\_data = f.read()  
data = np.loadtxt(**'arcene\_train.data'**)  
random.shuffle(data)  
train = data[int(0.7\*len(data)):]  
test = data[:int(0.3\*len(data))]  
*#6  
  
  
#7*train = data[int(0.7\*len(data)):]  
test = data[:int(0.3\*len(data))]  
train = numpy.array(data[int(0.7\*len(data)):])  
train\_labels = numpy.array(labels[int(0.7\*len(data)):])  
test = numpy.array(data[:int(0.3\*len(data))])  
knn = KNeighborsClassifier(n\_neighbors = 4)  
sfs = SFS(knn, k\_features = math.sqrt(len(train)), forward = **True**, floating = **False**, scoring = **'accuracy'**, cv = 4, n\_jobs = −1)  
sfs = sfs.fit(train,train\_labels)  
*#7  
  
  
#8*knn = KNeighborsClassifier(n\_neighbors = 4)  
sffs = SFS(knn, k\_features = 3, forward = **True**,  
floating = **True**, scoring = **'accuracy'**, cv = 4, n\_jobs = −1)  
*#8  
  
  
#9***with** open(**'arcene\_train.data'**) **as** f:  
 raw\_data = f.read()  
data = np.loadtxt(**'arcene\_train.data'**)  
labels = np.loadtxt(**'arcene\_train.labels'**)  
train = data[int(0.7\*len(data)):]  
test = data[:int(0.3\*len(data))]  
knn = KNeighborsClassifier(n\_neighbors = 5)  
sffs = SFS(knn, k\_features = 10, forward = **True**, floating = **True**, scoring = **'accuracy'**, cv = 4, n\_jobs = −1)  
T = sffs.fit(train,labels[int(0.7\*len(data)):])  
print(T.k\_score\_)  
*#9  
  
  
#10***with** open(**'arcene\_train.data'**) **as** f:  
 raw\_data = f.read()  
data = np.loadtxt(**'arcene\_train.data'**)  
labels = np.loadtxt(**'arcene\_train.labels'**)  
train = data[int(0.7\*len(data)):]  
test = data[:int(0.3\*len(data))]  
labels = labels[int(0.7\*len(data)):]  
knn = KNeighborsClassifier(n\_neighbors = 5)  
sbs = SFS(knn, k\_features = 20, forward = **False**, floating = **False**, scoring = **'accuracy'**, cv = 4, n\_jobs = −1)  
sbs = sbs.fit(train,labels)  
print(sbs.k\_score\_)  
*#10  
  
  
#11*knn = KNeighborsClassifier(n\_neighbors = 4)  
sfbs = SFS(knn, k\_features = 3, forward = **False**, floating = **True**,  
scoring = **'accuracy'**, cv = 4, n\_jobs = −1)  
*#11  
  
  
#12*data = np.loadtxt(**'arcene\_train.data'**)  
labels = np.loadtxt(**'arcene\_train.labels'**)  
train = data[int(0.7\*len(data)):]  
test = data[:int(0.3\*len(data))]  
labels = labels[int(0.7\*len(data)):]  
knn = KNeighborsClassifier(n\_neighbors = 5)  
knn = KNeighborsClassifier(n\_neighbors = 4)  
sfbs = SFS(knn, k\_features = 15, forward = **False**, floating = **True**, scoring = **'accuracy'**, cv = 4, n\_jobs = −1)  
sfbs = sbfs.fit(train,labels)  
print(sfbs.k\_score\_)  
*#12*