```
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
```