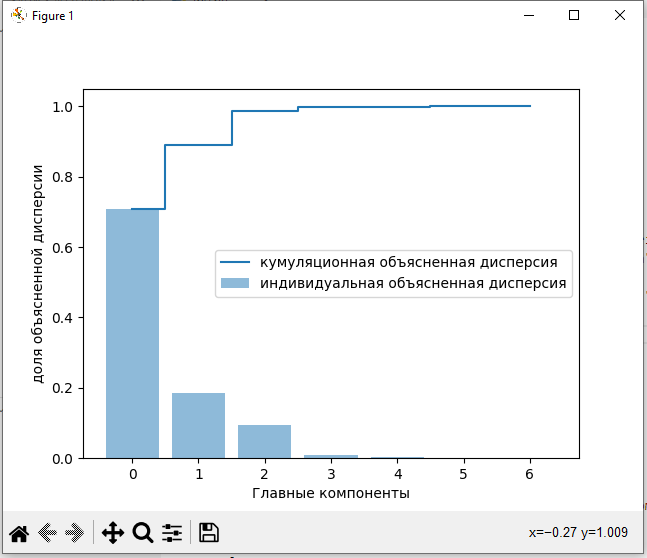
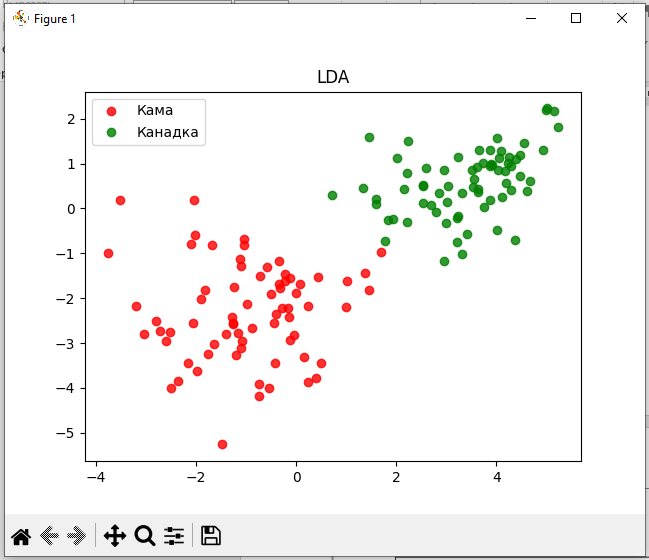




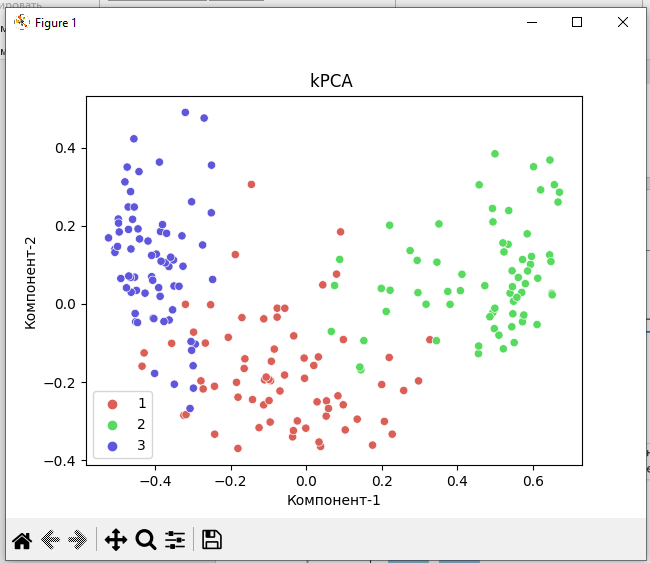
Он показывает, что первый основной компонент составляет 82,93% дисперсии, второй – 16.36 %. Можно сказать, что 82,93+ 16.36 = 99,29% информации собираются первыми и вторыми основными компонентами.



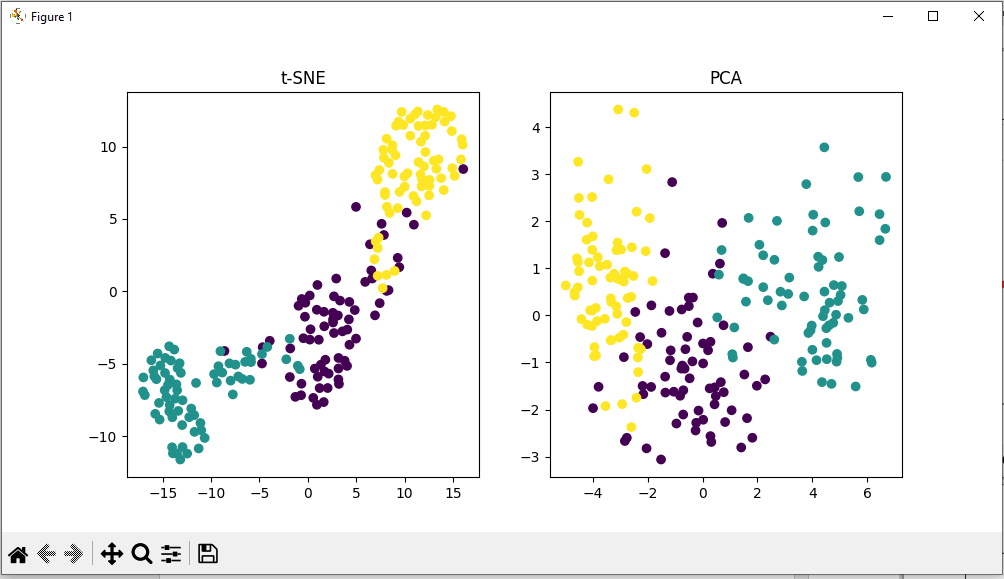
Полученный график показывает, что только первый главный компонент составляет примерно 70% дисперсии. Кроме того, мы можем видеть, что первые два главных компонента вместе объясняют почти 60% дисперсии в наборе данных.



Происхождение LDA отличается от PCA. PCA - это метод обучения без учителя, который преобразует исходные функции в набор новых функций. Если смотреть на первый рисунок можно увидеть отличия между LDA и PCA.



Ядро PCAрасширяет PCA до нелинейности. Сначала он отображает исходные данные в некоторое пространство нелинейных объектов (обычно это более высокое измерение), затем применяет PCA для извлечения основных компонентов в этом пространстве.



T-SNE алгоритм машинного обучения для визуализации, который представляет вложение многомерных данных в низкоразмерное пространство двух или трех измерений. По графику видно различия алгоритмов.

**Код программы**

from matplotlib import cm

import numpy as np

import matplotlib.pyplot as plt

from sklearn.svm import SVC

import pandas as pd

from sklearn.decomposition import PCA, KernelPCA

from sklearn import metrics

from sklearn.neighbors import KNeighborsClassifier

import seaborn as sns

from sklearn.manifold import TSNE

from sklearn. discriminant\_analysis import LinearDiscriminantAnalysis

url = "C:/Users/fox/Desktop/ii/data/seeds\_dataset.data"

names = ["площадь","периметр","компактность", "длина","ширина","асимметрия","длина канавки ядра","сорт"] #название атрибутов

target\_names = { 1:"Кама" , 2:"Роза" , 2:"Канадка" }

dataset = pd.read\_csv(url, names=names)

dataset.head()

X = dataset.iloc[:,0:7]

y = dataset['сорт'].values

print(X)

#---------------pca---------------------------

pca = PCA(n\_components=2)

X\_r = pca.fit(X).transform(X)

explained\_variance = pca.explained\_variance\_ratio\_

print("объясненный коэффициент дисперсии(первые два компонента):%s"%str(explained\_variance))

plt.figure()

colors=['r', 'g','b']

lw=1

#------------------------построить график по первым двум главным компонентам---------

for color,i,target\_name in zip(colors,[1,2,3], target\_names.values()):

    plt.scatter(X\_r[y==i,0], X\_r[y==i,1], c=color, alpha=.8, lw=lw, label=target\_name)

plt.title("PCA")

plt.legend(loc="best",shadow=False,scatterpoints=1)

plt.show()

#---------------------кумуляционная объясненная дисперсия----------------

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

X\_train, X\_test, y\_train, y\_test = train\_test\_split(

    X, y, test\_size=0.3,

    stratify=y, random\_state=0

)

sc = StandardScaler()

X\_train\_std = sc.fit\_transform(X\_train)

X\_test\_std = sc.transform(X\_test)

cov\_mat = np.cov(X\_train\_std.T)

eigen\_vals, eigen\_vecs = np.linalg.eig(cov\_mat)

print ("coбственные значения%s"% eigen\_vals)

# рассчитать совокупную сумму объясненных отклонений

tot = sum(eigen\_vals)

var\_exp = [(i / tot) for i in sorted(eigen\_vals, reverse=True)]

cum\_var\_exp = np.cumsum(var\_exp)

# вывод графика

plt.bar(range(0,7), var\_exp, alpha=0.5,

        align='center', label='индивидуальная объясненная дисперсия')

plt.step(range(0,7), cum\_var\_exp, where='mid',

         label='кумуляционная объясненная дисперсия')

plt.ylabel('доля объясненной дисперсии')

plt.xlabel('Главные компоненты')

plt.legend(loc='best')

plt.show()

#------------------------произвести снижение размерности призн прос-ва используя LDA---------

model = LinearDiscriminantAnalysis()

model. fit(X, y)

from sklearn. model\_selection import cross\_val\_score

from sklearn. model\_selection import RepeatedStratifiedKFold

#Define method to evaluate model

cv = RepeatedStratifiedKFold(n\_splits= 10 , n\_repeats= 3 , random\_state= 1 )

scores = cross\_val\_score(model, X, y, scoring='accuracy', cv=cv, n\_jobs=-1)

print(np.mean (scores))

plt.figure()

colors = ['red', 'green', 'blue']

lw = 2

data\_plot = model.fit (X, y).transform (X)

#------------------------построить график по первым двум главным компонентам---------

for color, i, target\_name in zip(colors, [1, 2, 3], target\_names.values()):

 plt.scatter (data\_plot[y == i, 0], data\_plot[y == i, 1], alpha=.8, color=color,

 label=target\_name)

plt.legend(loc='best', shadow= False , scatterpoints=1)

plt.title("LDA")

plt.show()

#------------------------произвести нелинейное снижение размерности призн прос-ва используя PCA и kpca---------

kpca = KernelPCA(kernel="rbf", n\_components=2, gamma=.01)

z = kpca.fit\_transform(X)

df = pd.DataFrame()

df["y"] = y

df["Компонент-1"] = z[:,0]

df["Компонент-2"] = z[:,1]

sns.scatterplot(x="Компонент-1", y="Компонент-2", hue=df.y.tolist(),

                palette=sns.color\_palette("hls", 3),

                data=df).set(title="kPCA ")

plt.show()

#------------------------t-SNE-----------------

X\_tsne = TSNE(learning\_rate=100, random\_state=1000).fit\_transform(X\_r)

X\_pca = PCA().fit\_transform(X)

plt.figure(figsize=(10, 5))

plt.subplot(121)

plt.title("t-SNE")

plt.scatter(X\_tsne[:, 0], X\_tsne[:, 1], c=y)

plt.subplot(122)

plt.title("PCA")

plt.scatter(X\_pca[:, 0], X\_pca[:, 1], c=y)

plt.show()