Домашнее задание 8

1. Можно ли отобрать наиболее значимые признаки с помощью PCA?

После того, как отработает метод главных компонент, матрица признаков переходит к новому базису. Причем процесс этот не обратим и обратно исходную матрицу не восстановить. Поэтому те компненты-признаки, которые образуются после применения PCA,не имеют ничего общего с исходныи признаками. Следовательно наиболее значимые признаки с помощью PCA подобрать нельзя.

2. (*) Написать свою реализацию метода главных компонент с помощью сингулярного разложения с использованием функции [numpy.linalg.svd() (https://docs.scipy.org/doc/numpy/reference/generated/numpy.linalg.svd.html)]

In [9]:

```
import numpy as np
from sklearn import datasets
import matplotlib.pyplot as plt
```

In [10]:

```
# Загрузим игрушечный датасет из sklearn
iris = datasets.load_iris()
X = iris.data
```

In [11]:

```
U, s, W = np.linalg.svd(X)
# Транспонируем матрицу W
V = W.T

# s - список диагональных элементов, его нужно привести к виду диагональной матрицы для
Sigma = np.zeros_like(X, dtype=float)
Sigma[np.diag_indices(min(X.shape))] = s
print(f'Матрица Sigma размер:{Sigma.shape} :\n{Sigma}')
print(f'Матрица U размер:{U.shape} :\n{U}')
print(f'Матрица V размер:{V.shape} :\n{V}')
```

```
Матрица Sigma размер:(150, 4) :
[[95.95991387  0.          0.          0.        ]
 [ 0.         17.76103366  0.          0.        ]
 [ 0.          0.          3.46093093  0.        ]
 [ 0.          0.          0.          1.88482631]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
 [ 0.          0.          0.          0.        ]
```

In [12]:

```
eig_sum = sum(s)
var_exp = [(i / eig_sum) * 100 for i in sorted(s, reverse=True)]
cum_var_exp = np.cumsum(var_exp)
print(f'Доля дисперсии, описвыаемая каждой из компонент \n{var_exp}')

# а теперя оценим кумулятивную (то есть накапливаемую) дисперсию при учитывании каждой
print(f'Кумулятивная доля дисперсии по компонентам \n{cum_var_exp}')
```

```
Доля дисперсии, описвыаемая каждой из компонент
[80.59340691495326, 14.916876798004958, 2.906715976729492, 1.583000310312297
2]
Кумулятивная доля дисперсии по компонентам
[ 80.59340691  95.51028371  98.41699969 100.        ]
```

Видим, что на первые два компонента приходится более 95% дисперсии. Занулим 3 и 4.

In [13]:

```python
n_elements = 2
Sigma = Sigma[:, :n_elements]
Sigma
```

Out[13]:

```
array([[95.95991387,  0.        ],
       [ 0.        , 17.76103366],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
       [ 0.        ,  0.        ],
```

In [14]:

```python
V = V[:,:n_elements]
V
```

Out[14]:

```
array([[-0.75110816,  0.2841749 ],
       [-0.38008617,  0.5467445 ],
       [-0.51300886, -0.70866455],
       [-0.16790754, -0.34367081]])
```

In [15]:

```
# Создадим новую матрицу признаков с 4 признаками, но рангом 2
B = U.dot(Sigma.dot(V.T))
B
```

Out[15]:

```
array([[5.0952927 , 3.50597743, 1.40192232, 0.20165319],
       [4.74588049, 3.19610853, 1.46136967, 0.25800276],
       [4.68667405, 3.21586325, 1.30954904, 0.19452725],
       [4.61488457, 3.08894388, 1.46347879, 0.27002699],
       [5.07488651, 3.50623125, 1.36428119, 0.1863997 ],
       [5.52598407, 3.7330351 , 1.67566825, 0.28872322],
       [4.731593  , 3.2288014 , 1.36216771, 0.21446447],
       [5.00510918, 3.39830515, 1.47931372, 0.24418439],
       [4.37933538, 2.93134058, 1.38864652, 0.25618379],
       [4.80551481, 3.23360903, 1.48569239, 0.26393296],
       [5.39533378, 3.70766642, 1.49514864, 0.2183418 ],
       [4.89451945, 3.29088668, 1.51906398, 0.27146211],
       [4.67854319, 3.16443092, 1.41000708, 0.24052709],
       [4.30090163, 3.00174374, 1.08842179, 0.12739443],
       [5.73037625, 4.07476895, 1.2813685 , 0.09359359],
       [5.90310008, 4.12548386, 1.48153824, 0.16928199],
       [5.45240789, 3.81002345, 1.36951483, 0.15684908],
       [5.09813811, 3.49356935, 1.43489086, 0.21628344],
```

In [16]:

```
# Проведем сингулярное разложение матрицы B
U, s, W = np.linalg.svd(B)
# Транспонируем матрицу W
V = W.T
print("Элементы диагональной матрицы", s)
```

```
Элементы диагональной матрицы [9.59599139e+01 1.77610337e+01 2.07617549e-14
2.38003498e-15]
```

Видим, что они практически равны 0. Сосздадим матрицу ризнаков C в новом двухмерном базисе через матрицу перехода

In [17]:

```
C = W.dot(B.T).T
C
```

Out[17]:

```
array([[-5.91274714e+00,  2.30203322e+00,  1.79717352e-15,
        -1.11022302e-16],
       [-5.57248242e+00,  1.97182599e+00,  1.86309301e-15,
        -5.55111512e-17],
       [-5.44697714e+00,  2.09520636e+00,  1.15012166e-15,
        -1.11022302e-16],
       [-5.43645948e+00,  1.87038151e+00,  2.02268757e-15,
        -1.11022302e-16],
       [-5.87564494e+00,  2.32829018e+00,  2.05564732e-15,
         0.00000000e+00],
       [-6.47759822e+00,  2.32464996e+00,  2.03656536e-15,
         1.66533454e-16],
       [-5.51597520e+00,  2.07090423e+00,  1.42941214e-15,
        -2.77555756e-17],
       [-5.85092859e+00,  2.14807482e+00,  1.60635394e-15,
        -2.77555756e-17],
       [-5.15891972e+00,  1.77506408e+00,  1.48839274e-15,
         0.00000000e+00],
```

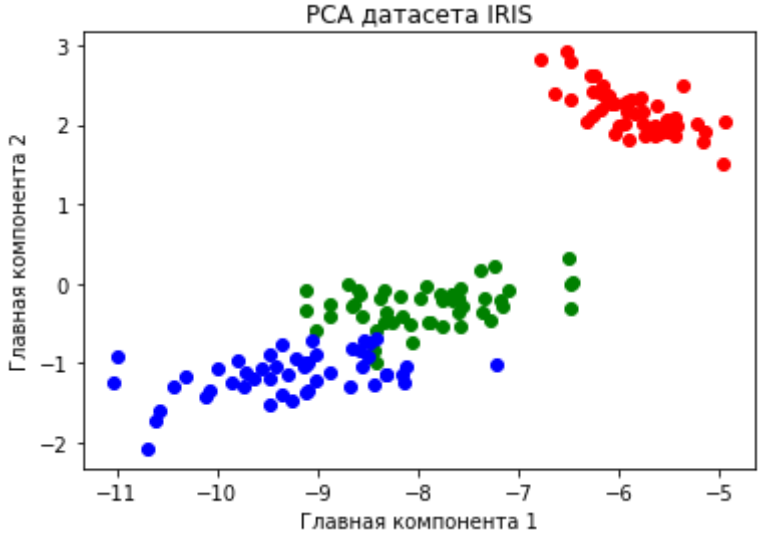
Видим, что последние два столбца признаков стали равны 0. Их можно удалить.

In [18]:

```
# Сформируем новую матрицу "объекты-признаки"
D = C[:, :2]
```

In [19]:

```
plt.figure()
y = iris.target
for c, i in zip("rgb", [0, 1, 2]):
    plt.scatter(D[y==i, 0], D[y==i, 1], c=c)
plt.xlabel('Главная компонента 1')
plt.ylabel('Главная компонента 2')
plt.title('PCA датасета IRIS')
plt.show()
```



3. (*) Обучить любую модель классификации на датасете IRIS до применения PCA и после него. Сравнить качество классификации по отложенной выборке.

In [20]:

```
from sklearn.tree import DecisionTreeClassifier
from sklearn import model_selection
from sklearn.metrics import accuracy_score
```

In [21]:

```
iris = datasets.load_iris()
X = iris.data
y = iris.target
X_train, X_test, y_train, y_test = model_selection.train_test_split(X, y, test_size=0.
```

In [22]:

```
model = DecisionTreeClassifier(max_depth=2)
model.fit(X_train, y_train)
```

Out[22]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=2, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [23]:

```
y_predict_train = model.predict(X_train)
y_predict_test = model.predict(X_test)
print("Точность на обучающей выборке", accuracy_score(y_train, y_predict_train))
print("Точность на тестовой выборке", accuracy_score(y_test, y_predict_test))
```

```
Точность на обучающей выборке 0.9619047619047619
Точность на тестовой выборке 0.9555555555555556
```

In [24]:

```
X_train_PCA, X_test_PCA, y_train_PCA, y_test_PCA = model_selection.train_test_split(D,
model = DecisionTreeClassifier(max_depth=2)
model.fit(X_train_PCA, y_train_PCA)
```

Out[24]:

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='gini',
                       max_depth=2, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=None, splitter='best')
```

In [25]:

```
y_predict_train_PCA = model.predict(X_train_PCA)
y_predict_test_PCA = model.predict(X_test_PCA)
print("Точность на обучающей выборке", accuracy_score(y_train_PCA, y_predict_train_PCA)
print("Точность на тестовой выборке", accuracy_score(y_test_PCA, y_predict_test_PCA))
```

```
Точность на обучающей выборке 0.9714285714285714
Точность на тестовой выборке 1.0
```

Вывод: Результаты работы модели после снижения размерности стали лучше