МИНОБРНАУКИ РОССИИ
САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ
ЭЛЕКТРОТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ
«ЛЭТИ» ИМ. В.И. УЛЬЯНОВА (ЛЕНИНА)
Кафедра МОЭВМ

ОТЧЕТ

по лабораторной работе №2

по дисциплине «Машинное обучение»

Студенты гр. 6304 _____ Тимофеев А.А.

Преподаватель _____ Жангиров Т.Р.

Санкт-Петербург

2020

Цель работы

Ознакомиться с методами предобработки данных из библиотеки Scikit Learn

Ход работы

Загрузка данных

1. Был создан датафрейм Pandas на основе загруженного датасета (https://www.kaggle.com/uciml/glass)
2. Данные были разделены на описательные признаки и признак отображающий класс, а затем нормированы.
3. Были построены диаграммы рассеивания для пар признаков. Результат представлен на рис. 1.

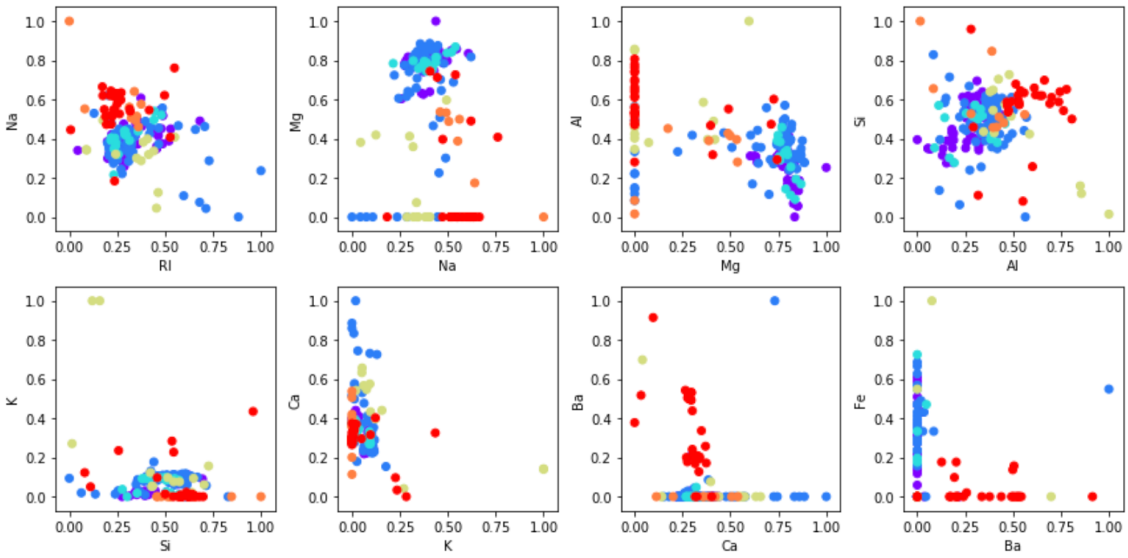


Рисунок 1 – Диаграммы рассеивания для пар признаков

4. Соответствие цвета на диаграмме и класса на датасете показано на рис. 2.

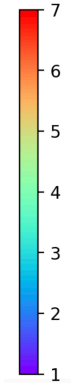


Рисунок 2 – Соответствие класса и цвета

Метод главных компонент

1. Было проведено понижение размерности до двух компонент. Соответствующая диаграмма рассеивания приведена на рис. 3.

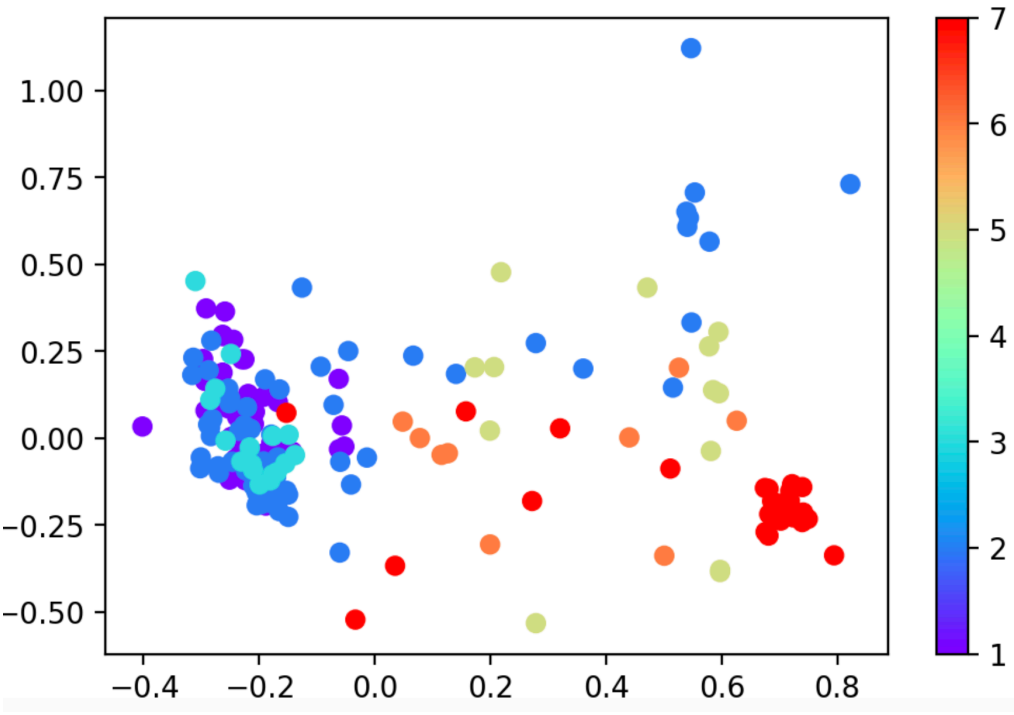


Рисунок 3 – Диаграмма рассеивания после применения PCA

Соответствующие значения объясненной дисперсии и сингулярных чисел:

Компонент	Дисперсия	Сингулярное число
PC1	0.45429569	5.1049308
PC2	0.17990097	3.21245688

Компоненты объясняют ~63% дисперсии данных. Также по диаграмме рассеивания видно, что между данными отсутствует явная линейная зависимость.

2. Путем изменения количества компонентов было установлено, что значение объясненной дисперсии равное ~85% достигается при количестве компонентов равном 4.

3. Было выполнено восстановление исходных данных при помощи метода inverse_transform. Сравнение диаграмм рассеивания исходных и восстановленных данных представлено на рис. 4 (сверху диаграммы исходных данных, ниже - восстановленных).

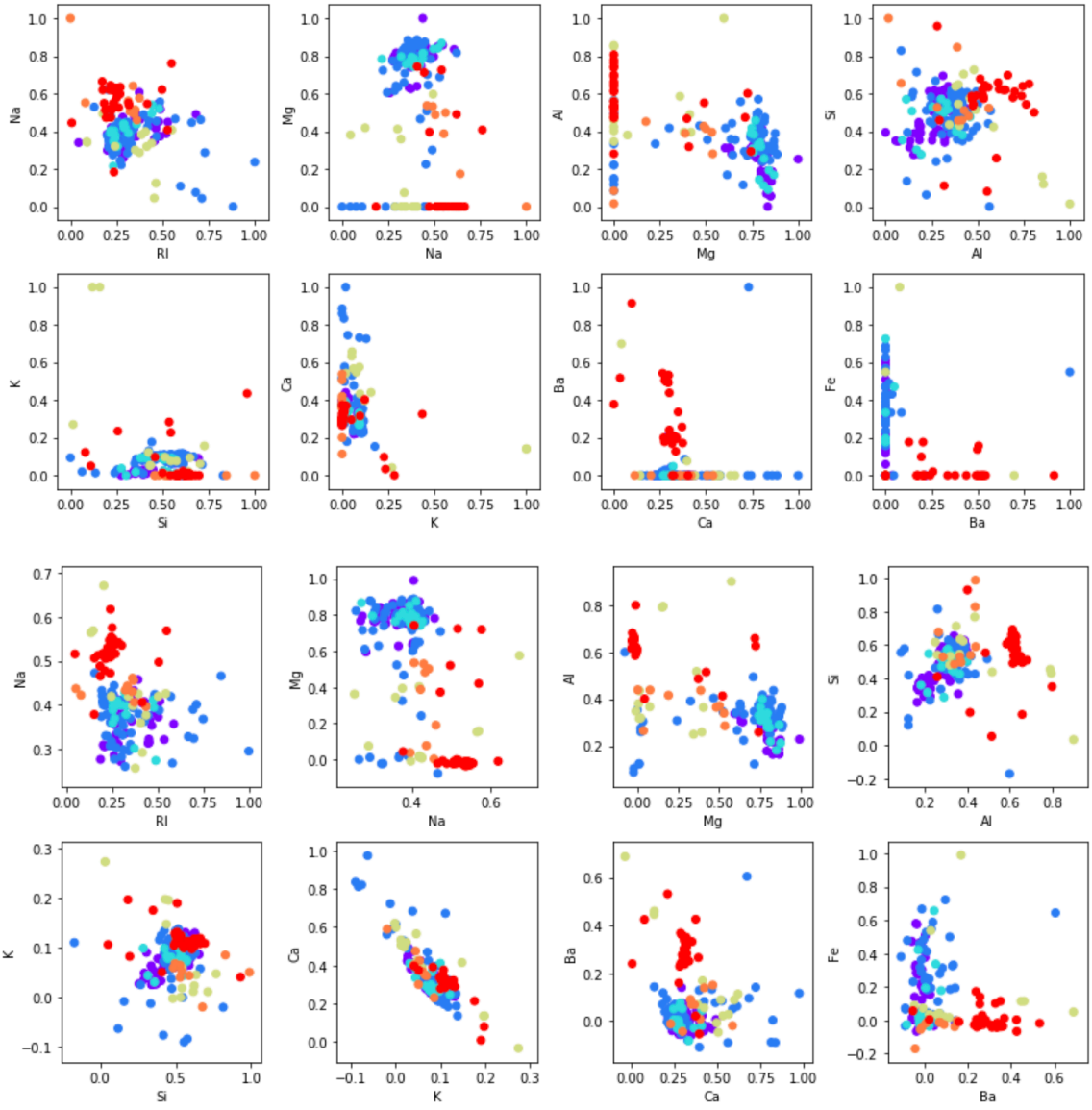


Рисунок 4 – Сравнение диаграмм рассеивания

На основании получившихся диаграмм можно сделать вывод, что данные были восстановлены с потерями, что объясняется размером объясненной дисперсии ~86%.

4. Было выполнено исследование PCA при различных значениях svd_solver, соответствующие диаграммы приведены на рис. 5. При значении ‘full’ выполняется полное разложение алгоритмом LAPACK, а затем определяется количество компонент. При значении ‘arpack’

выполняется усеченное разложение методом ARPACK для заданного числа компонент. При значении ‘randomized’ выполняется рандомизированное разложение по методу Халко. Если значение параметра не указано, то на основании размера выборки применяется либо полное, либо рандомизированное разложение.

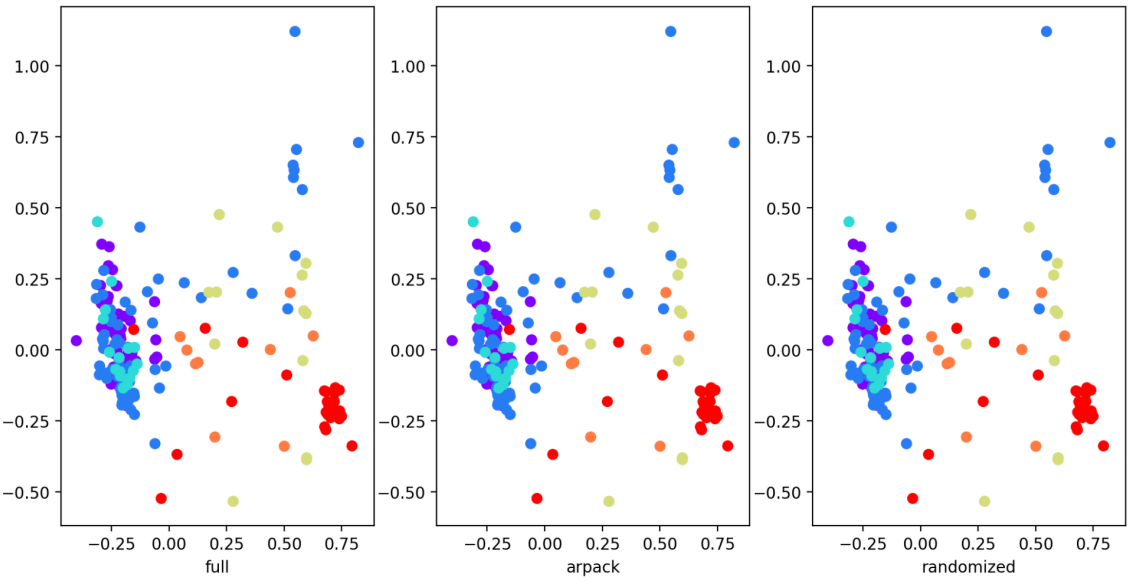
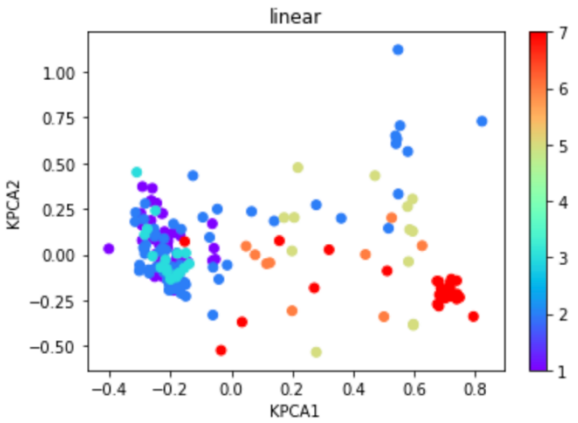


Рисунок 5 – Диаграммы рассеивания при различных значениях svd_solver

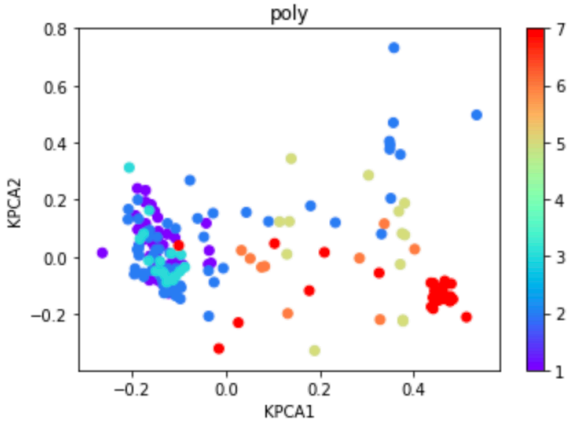
Из диаграмм следует, что на данном наборе данных метод сингулярного разложения не влияет на конечный результат.

Модификации метода главных компонент

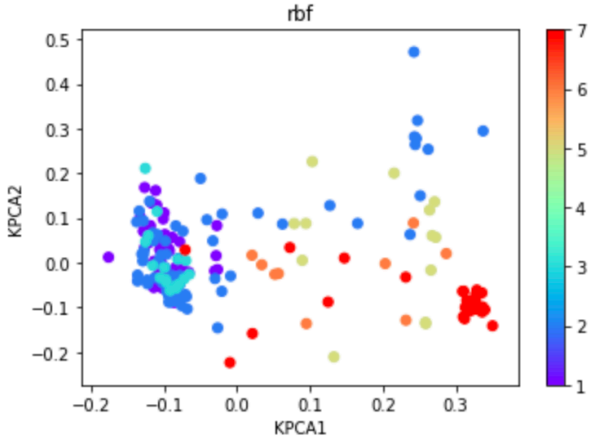
1. Было проведено исследование KernelPCA при различных параметрах kernel. Соответствующие диаграммы представлены на рис.6. В результате исследования можно сделать вывод, что собственные числа отличаются в виду разных функций ядра, но значения объясненной дисперсии различаются в пределах 1-2%.



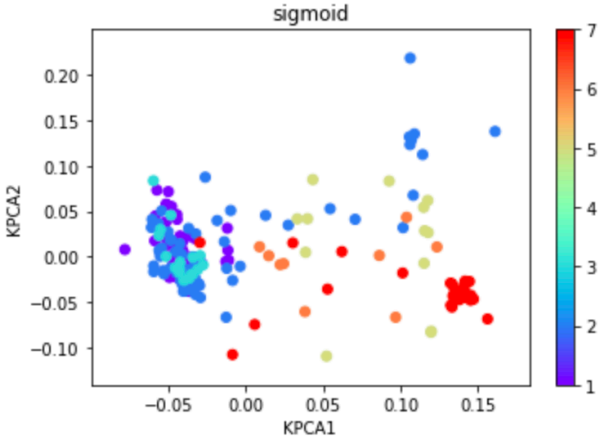

```
lambdas: [26.06031845 10.31987923]
explained variance: 0.6341966621042778
```



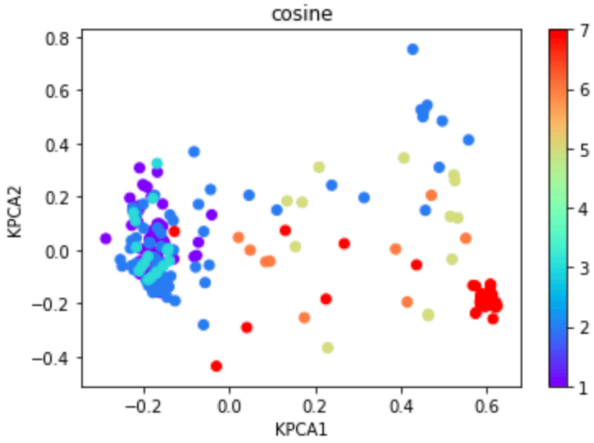

```
lambdas: [10.9181964   4.31937695]
explained variance: 0.6242506448537448
```




```
lambdas: [5.35145251 2.0180542 ]
explained variance: 0.6278600006634807
```




```
lambdas: [1.00618101 0.39983752]
explained variance: 0.6390542671618413
```




```
lambdas: [18.31403041  6.47538495]
explained variance: 0.6447433552468603
```

Рисунок 6 – Диаграммы рассеивания при различных значениях kernel

2. Было выполнено исследование SparcePCA. Разреженный PCA вычисляет разреженные линейные комбинации данных. В отличии от обычного PCA, данный метод применим в условиях, когда количество признаков больше либо равно числу примеров в выборке.

Разреженность в методе библиотеки sklearn контролируется параметром alpha: чем он выше, тем она больше. Диаграммы при различных значениях данного параметра представлены на рис. 7.

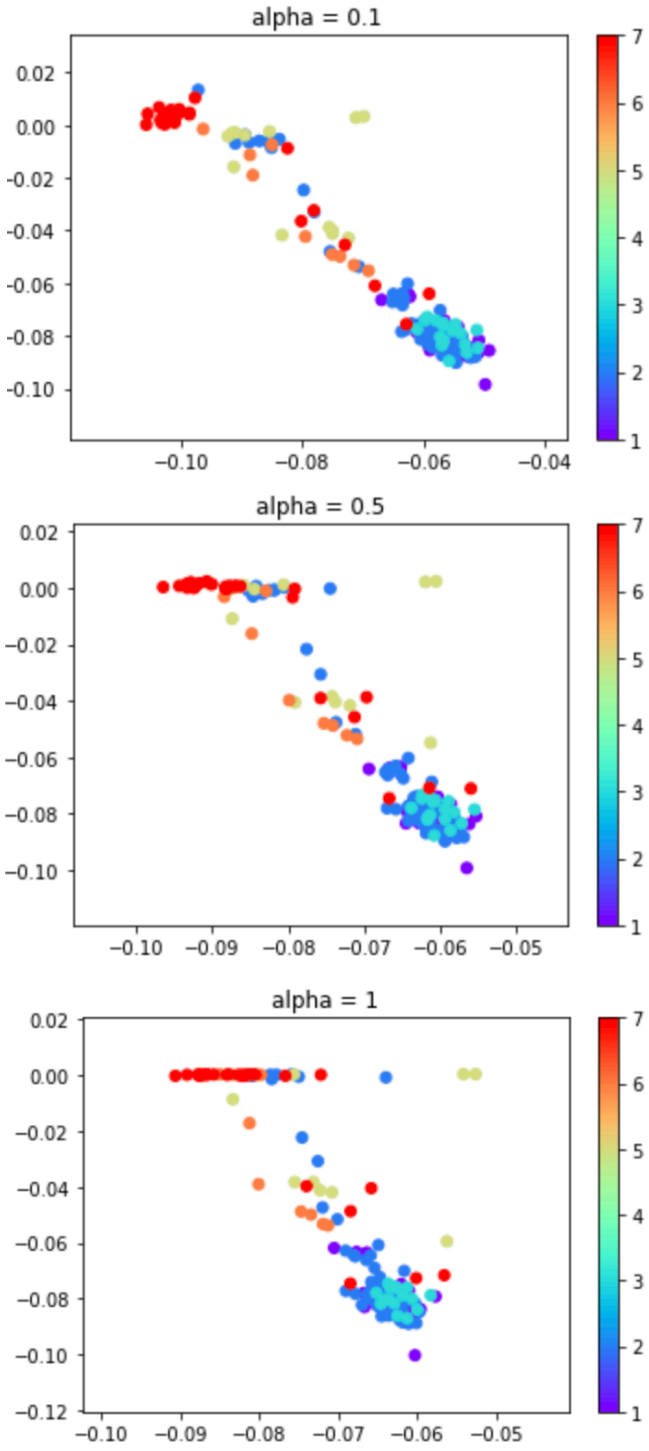


Рисунок 7 – Диаграммы рассеивания при различных значениях alpha

Доступно два метода решения задачи: наименьшая угловая регрессия и координатный спуск. Первый работает быстрее, если предполагаемые компоненты разрежены. Было установлено, что данные в выборке не

являются разреженными, а также проведено 10 опытов анализа с применением обоих методов. На рис. 8 представлены результаты опытов по времени. Метод координатного спуска отработал быстрее.

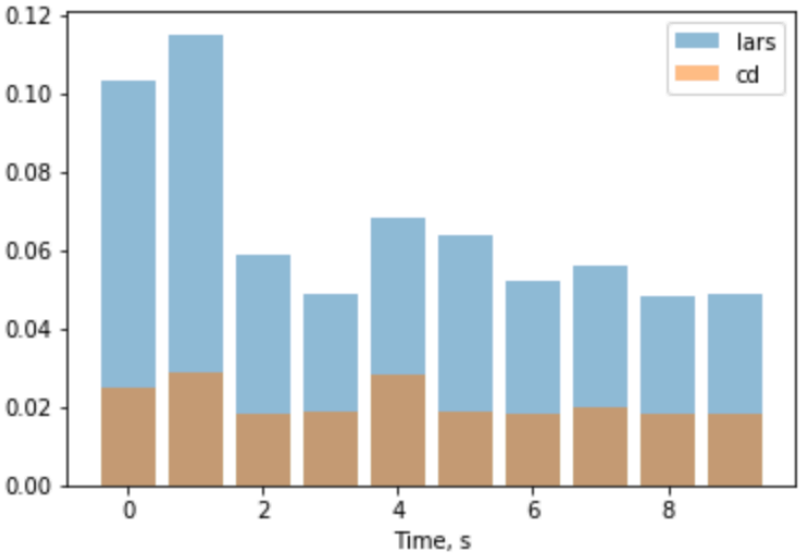


Рисунок 8 – Результаты применение методов lars и cd

Факторный анализ

1. Было выполнено понижение размерности с помощью факторного анализа. На рис. 9 представлено сравнение полученной диаграммы с диаграммой после PCA. Как видно из диаграммы, применение FA позволило увидеть зависимость между компонентами.

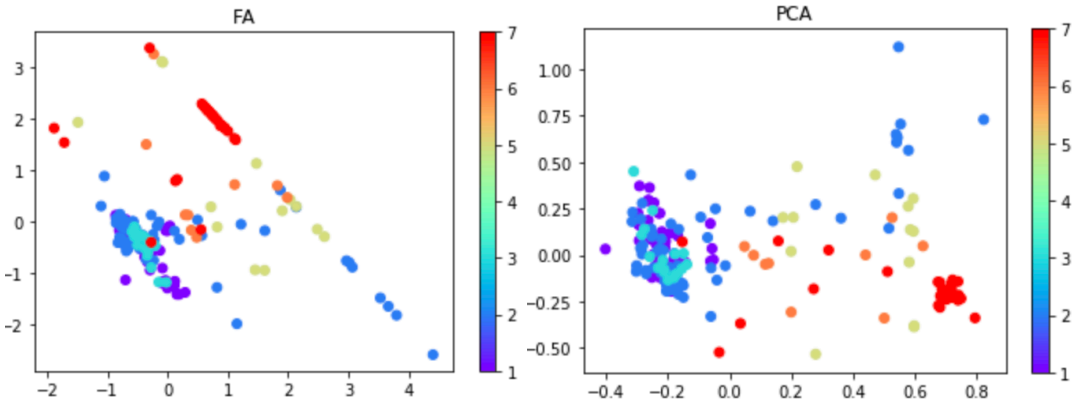


Рисунок 9 – Сравнение диаграмм после FA и PCA

Различия FA и PCA:

- FA производится для поиска связей между переменными, их классификации, PCA для уменьшения размерности

- В FA нет ограничения на направление при поиске дисперсии, в PCA компоненты должны быть ортогональны друг другу

Выводы

В ходе выполнения данной лабораторной работы были следующие изучены методы понижения размерности из библиотеки Scikit Learn: PCA, KernelPCA и SparsePCA. Также было выполнено сравнение FA и PCA при решении задачи понижения размерности.

ПРИЛОЖЕНИЕ А

Исходный код

```
import pandas as pd
import numpy as np
import matplotlib as mpl
import matplotlib.pyplot as plt

mpl.rcParams['figure.dpi'] = 200

df = pd.read_csv('glass.csv')

var_names = list(df.columns) #получение имен признаков

labels = df.to_numpy('int')[:,-1] #метки классов
data = df.to_numpy('float')[:,:-1] #описательные признаки
df.describe()

from sklearn import preprocessing

data = preprocessing.minmax_scale(data)

fig, axs = plt.subplots(2,4, figsize=(12, 6))

for i in range(data.shape[1]-1):
    axs[i // 4, i % 4].scatter(data[:,i], data[:,(i+1)], c=labels, cmap='rainbow')
    axs[i // 4, i % 4].set_xlabel(var_names[i])
    axs[i // 4, i % 4].set_ylabel(var_names[i+1])

plt.tight_layout()
plt.show()

from sklearn.decomposition import PCA

pca = PCA(n_components = 2)

pca_data = pca.fit(data).transform(data)

print(pca.explained_variance_ratio_)
print(pca.singular_values_)

plt.scatter(pca_data[:,0],pca_data[:,1],c=labels,cmap='rainbow')
plt.colorbar()
plt.title('PCA')
plt.show()

pca = PCA(n_components = 4)

pca_data = pca.fit(data).transform(data)

print(pca.explained_variance_ratio_)
print(pca.singular_values_)
print(np.sum(pca.explained_variance_ratio_))

restored_data = pca.inverse_transform(pca_data)

fig, axs = plt.subplots(2,4, figsize=(12, 6))

for i in range(data.shape[1]-1):
    axs[i // 4, i % 4].scatter(restored_data[:,i], restored_data[:,(i+1)], c=labels,
cmap='rainbow')
```

```
    axs[i // 4, i % 4].set_xlabel(var_names[i])
    axs[i // 4, i % 4].set_ylabel(var_names[i+1])

plt.tight_layout()
plt.show()

pca_full = PCA(n_components = 2, svd_solver='full')
pca_data_full = pca_full.fit(data).transform(data)

pca_arpack = PCA(n_components = 2, svd_solver='arpack')
pca_data_arpack = pca_arpack.fit(data).transform(data)

pca_randomized = PCA(n_components = 2, svd_solver='randomized')
pca_data_randomized = pca_randomized.fit(data).transform(data)

fig, axs = plt.subplots(1,3, figsize=(12, 6))
axs[0].scatter(pca_data_full[:,0], pca_data_full[:,1], c=labels, cmap='rainbow')
axs[0].set_xlabel('full')

axs[1].scatter(pca_data_arpack[:,0], pca_data_arpack[:,1], c=labels, cmap='rainbow')
axs[1].set_xlabel('arpack')

axs[2].scatter(pca_data_randomized[:,0], pca_data_randomized[:,1], c=labels,
cmap='rainbow')
axs[2].set_xlabel('randomized')

plt.show()

from sklearn.decomposition import KernelPCA

kernels = ['linear', 'poly', 'rbf', 'sigmoid', 'cosine']

for kernel in kernels:
    k_pca = KernelPCA(n_components=2, kernel=kernel)
    k_pca_max_components = KernelPCA(n_components=data.shape[1],
kernel=kernel).fit(data)
    k_pca_data = k_pca.fit_transform(data)
    plt.scatter(k_pca_data[:,0], k_pca_data[:,1], c=labels,cmap='rainbow')
    plt.colorbar()
    plt.title(kernel)
    plt.xlabel('KPCA1')
    plt.ylabel('KPCA2')
    plt.show()
    print('lambdas: {}\nexplained variance: {}\n'.format(k_pca.lambdas_,
sum(k_pca.lambdas_)/sum(k_pca_max_components.lambdas_)))

kernel_pca_precomp = KernelPCA(n_components=2, kernel='precomputed')
k_pca_max_precomp_components = KernelPCA(n_components=data.shape[1],
kernel=kernel).fit(data)
kernel_pca_precomp_data = kernel_pca_precomp.fit_transform(data.dot(data.T))
plt.scatter(kernel_pca_precomp_data[:,0], kernel_pca_precomp_data[:,1],
c=labels,cmap='rainbow')
plt.colorbar()
plt.title('precomputed')
plt.xlabel('KPCA1')
plt.ylabel('KPCA2')
plt.show()
print('lambdas: {}\nexplained variance: {}\n'.format(kernel_pca_precomp.lambdas_,
sum(kernel_pca_precomp.lambdas_)/sum(k_pca_max_precomp_components.lambdas_)))

from sklearn.decomposition import SparsePCA
```

```
methods = ['lars', 'cd']

for method in methods:
    s_pca = SparsePCA(n_components=2, method=method)
    s_pca_data = s_pca.fit_transform(data)
    plt.scatter(s_pca_data[:,0], s_pca_data[:,1], c=labels,cmap='rainbow')
    plt.colorbar()
    plt.title(method)
    plt.show()
    print('components: {}\n'.format(s_pca.components_))

from scipy.sparse import isspmatrix
import time

print('Is data sparce:', isspmatrix(data))

lars_times = []
cd_times = []
for _ in range(10):
    start_time = time.time()
    s_pca = SparsePCA(n_components=2, alpha=1, method='lars')
    s_pca_data = s_pca.fit_transform(data)
    lars_times.append(round(time.time() - start_time, 3))
    start_time = time.time()
    s_pca = SparsePCA(n_components=2, alpha=alpha, method='cd')
    s_pca_data = s_pca.fit_transform(data)
    cd_times.append(round(time.time() - start_time, 3))
print('lars:', lars_times)
print('cd:', cd_times)

bins = np.arange(0, 10, 1)
plt.bar(bins, lars_times, alpha=0.5, label='lars')
plt.bar(bins, cd_times, alpha=0.5, label='cd')
plt.xlabel('Time, s')
plt.legend(loc='upper right')
plt.show()

alphas = [0.1, 0.5, 1]

for alpha in alphas:
    s_pca = SparsePCA(n_components=2, alpha=alpha, method='cd')
    s_pca_data = s_pca.fit_transform(data)
    plt.scatter(s_pca_data[:,0], s_pca_data[:,1], c=labels,cmap='rainbow')
    plt.colorbar()
    plt.title('alpha = {}'.format(alpha))
    plt.show()
    print('components: \n{}'.format(s_pca.components_))

from sklearn.decomposition import FactorAnalysis

fa = FactorAnalysis(n_components=2)
fa_data = fa.fit_transform(data)
plt.scatter(fa_data[:,0], fa_data[:,1], c=labels,cmap='rainbow')
plt.colorbar()
plt.title('FA')
plt.show()
```