Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине «Интеллектуальный анализ данных»

Тема: «PCA»

Выполнила:

Студентка 4 курса

Группы ИИ-24

Лящук А. В.

Проверила:

Якимук А. В.

Брест 2025

Цель: научиться применять метод PCA для осуществления визуализации данных.

Общее задание

1. Используя выборку по варианту, осуществить проецирование данных на плоскость первых двух и трех главных компонент (двумя способами: 1. вручную через использование numpy.linalg.eig для вычисления собственных значений и собственных векторов и 2. с помощью sklearn.decomposition.PCA для непосредственного применения метода PCA – два независимых варианта решения);

2. Выполнить визуализацию полученных главных компонент с использованием средств библиотеки matplotlib, обозначая экземпляры разных классов с использованием разных цветовых маркеров;

3. Используя собственные значения, рассчитанные на этапе 1, вычислить потери, связанные с преобразованием по методу PCA. Сделать выводы;

4. Оформить отчет по выполненной работе, загрузить исходный код и отчет в соответствующий репозиторий на github.

Задание по вариантам

№ варианта	Выборка	Класс
10	wholesale+customers.zip	Region

Код:

```
import pandas as pd
import numpy as np
from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt

# Загружаем данные seeds (без заголовка)
df = pd.read_csv('seeds_dataset.txt', header=None, delim_whitespace=True)

# Добавляем названия колонок как в примере
cols = ["area", "perimeter", "compactness", "length of kernel",
        "width of kernel", "asymmetry coefficient", "length of kernel
groove", "class"]
df.columns = cols

# Смотрим на первые строки и информацию о данных
print("Первые 5 строк данных:")
print(df.head())
print("\nИнформация о данных:")
```

```
print(df.info())
print("\nПроверка на пропуски:")
print(df.isnull().sum())

# Проверяем уникальные классы
classes = df['class'].unique()
print(f"\nУникальные классы: {classes}")
print("Количество записей по классам:")
print(df['class'].value_counts())

# Сохраняем метки класса для будущей визуализации (преобразуем в строки для
удобства)
class_labels = df['class'].astype(str)  # Целевая переменная для раскраски

# Создаем копию DataFrame без столбца 'class'
# PCA применяется только к числовым данным
data = df.drop(['class'], axis=1)

# Стандартизируем данные! Это ВАЖНЫЙ шаг для PCA.
scaler = StandardScaler()
data_scaled = scaler.fit_transform(data)

# 1. Рассчитываем ковариационную матрицу
cov_matrix = np.cov(data_scaled.T)  # Важно: транспонируем, так как строки -
это наблюдения

# 2. Вычисляем собственные значения и собственные векторы
eig_vals, eig_vecs = np.linalg.eig(cov_matrix)

# 3. Сортируем собственные векторы по убыванию собственных значений
eig_pairs = [(np.abs(eig_vals[i]), eig_vecs[:, i]) for i in
range(len(eig_vals))]
eig_pairs.sort(key=lambda x: x[0], reverse=True)

# 4. Формируем матрицу проекции W из первых 2 и первых 3 собственных векторов
matrix_w_2d = np.hstack((eig_pairs[0][1].reshape(7, 1),
eig_pairs[1][1].reshape(7, 1)))
matrix_w_3d = np.hstack((matrix_w_2d, eig_pairs[2][1].reshape(7, 1)))

# 5. Проецируем стандартизированные данные на новые главные компоненты
manual_pca_2d = data_scaled.dot(matrix_w_2d)
manual_pca_3d = data_scaled.dot(matrix_w_3d)

# Создаем объект PCA для 2 и 3 компонент
pca_2d = PCA(n_components=2)
pca_3d = PCA(n_components=3)

# Применяем PCA к стандартизированным данным
sklearn_pca_2d = pca_2d.fit_transform(data_scaled)
sklearn_pca_3d = pca_3d.fit_transform(data_scaled)

# Создаем фигуру с 4 subplot'ами: 2D ручной, 2D sklearn, 3D ручной, 3D
sklearn
fig = plt.figure(figsize=(18, 12))

# Определяем цвета для каждого класса
classes_unique = sorted(class_labels.unique())
colors = ['r', 'g', 'b', 'c', 'm', 'y', 'k']
color_map = {cls: colors[i] for i, cls in enumerate(classes_unique)}

# 1. Ручной метод 2D
```

```
ax1 = fig.add_subplot(2, 2, 1)
for cls in classes_unique:
    idx = class_labels == cls
    ax1.scatter(manual_pca_2d[idx, 0], manual_pca_2d[idx, 1],
                c=color_map[cls], label=f'Class {cls}', alpha=0.7)
ax1.set_title('Ручной PCA (2D)')
ax1.set_xlabel('PC1')
ax1.set_ylabel('PC2')
ax1.legend()

# 2. Sklearn метод 2D
ax2 = fig.add_subplot(2, 2, 2)
for cls in classes_unique:
    idx = class_labels == cls
    ax2.scatter(sklearn_pca_2d[idx, 0], sklearn_pca_2d[idx, 1],
                c=color_map[cls], label=f'Class {cls}', alpha=0.7)
ax2.set_title('Sklearn PCA (2D)')
ax2.set_xlabel(f'PC1 ({pca_2d.explained_variance_ratio_[0]*100:.2f}%)')
ax2.set_ylabel(f'PC2 ({pca_2d.explained_variance_ratio_[1]*100:.2f}%)')
ax2.legend()

# 3. Ручной метод 3D
ax3 = fig.add_subplot(2, 2, 3, projection='3d')
for cls in classes_unique:
    idx = class_labels == cls
    ax3.scatter(manual_pca_3d[idx, 0], manual_pca_3d[idx, 1],
manual_pca_3d[idx, 2],
                c=color_map[cls], label=f'Class {cls}', alpha=0.7)
ax3.set_title('Ручной PCA (3D)')
ax3.set_xlabel('PC1')
ax3.set_ylabel('PC2')
ax3.set_zlabel('PC3')
ax3.legend()

# 4. Sklearn метод 3D
ax4 = fig.add_subplot(2, 2, 4, projection='3d')
for cls in classes_unique:
    idx = class_labels == cls
    ax4.scatter(sklearn_pca_3d[idx, 0], sklearn_pca_3d[idx, 1],
sklearn_pca_3d[idx, 2],
                c=color_map[cls], label=f'Class {cls}', alpha=0.7)
ax4.set_title('Sklearn PCA (3D)')
ax4.set_xlabel(f'PC1 ({pca_3d.explained_variance_ratio_[0]*100:.2f}%)')
ax4.set_ylabel(f'PC2 ({pca_3d.explained_variance_ratio_[1]*100:.2f}%)')
ax4.set_zlabel(f'PC3 ({pca_3d.explained_variance_ratio_[2]*100:.2f}%)')
ax4.legend()

plt.tight_layout()
plt.show()

# Расчет объясненной дисперсии
total_variance = sum(eig_vals)  # Сумма всех собственных значений
explained_variance_ratio_manual = [eig_pair[0] / total_variance for eig_pair
in eig_pairs]

print("\n" + "="*50)
print("АНАЛИЗ ОБЪЯСНЕННОЙ ДИСПЕРСИИ")
print("="*50)

print("\nРучной метод:")
```

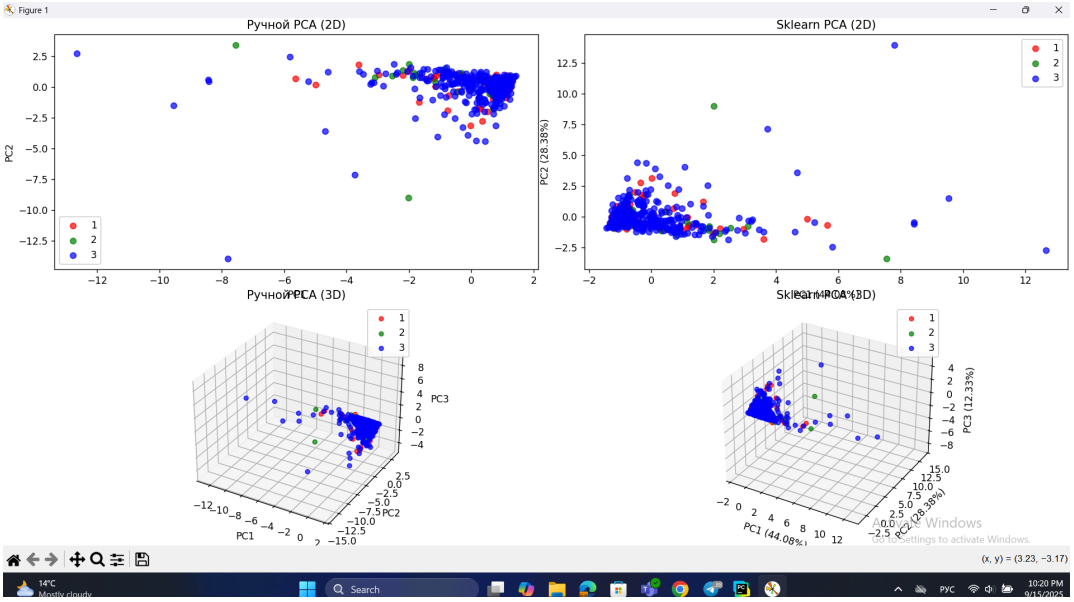
```
print(f"Объясненная дисперсия 2 главных компонент:
{sum(explained_variance_ratio_manual[:2]):.4f} или
{sum(explained_variance_ratio_manual[:2])*100:.2f}%")
print(f"Объясненная дисперсия 3 главных компонент:
{sum(explained_variance_ratio_manual[:3]):.4f} или
{sum(explained_variance_ratio_manual[:3])*100:.2f}%")
print(f"Потери при переходе к 2D: {(1 -
sum(explained_variance_ratio_manual[:2]))*100:.2f}%")

print("\nSklearn метод (2D):")
print(f"Объясненная дисперсия каждой компоненты:
{pca_2d.explained_variance_ratio_}")
print(f"Суммарная объясненная дисперсия:
{sum(pca_2d.explained_variance_ratio_):.4f} или
{sum(pca_2d.explained_variance_ratio_)*100:.2f}%")

print("\nSklearn метод (3D):")
print(f"Объясненная дисперсия каждой компоненты:
{pca_3d.explained_variance_ratio_}")
print(f"Суммарная объясненная дисперсия:
{sum(pca_3d.explained_variance_ratio_):.4f} или
{sum(pca_3d.explained_variance_ratio_)*100:.2f}%")

# Дополнительно: график объясненной дисперсии (Scree plot)
plt.figure(figsize=(10, 6))
plt.plot(range(1, len(explained_variance_ratio_manual) + 1),
         explained_variance_ratio_manual, 'o-', linewidth=2)
plt.title('Scree Plot (График объясненной дисперсии)')
plt.xlabel('Номер главной компоненты')
plt.ylabel('Объясненная дисперсия')
plt.grid(True)
plt.show()
```

Вывод:



Вывод: научился применять метод PCA для осуществления визуализации данных.