Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №1

По дисциплине «интеллектуальный анализ данных»

Тема: “PCA”

Выполнил:

Студент 4 курса

Группы ИИ-23

Швороб В.А.

Проверила:

Андренко К.В.

Брест 2025

Цель: научиться применять метод PCA для осуществления визуализации данных

Вариант 12.

№ варианта	Выборка	Класс
12	hcv+data.zip	Category

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.

Код программы:

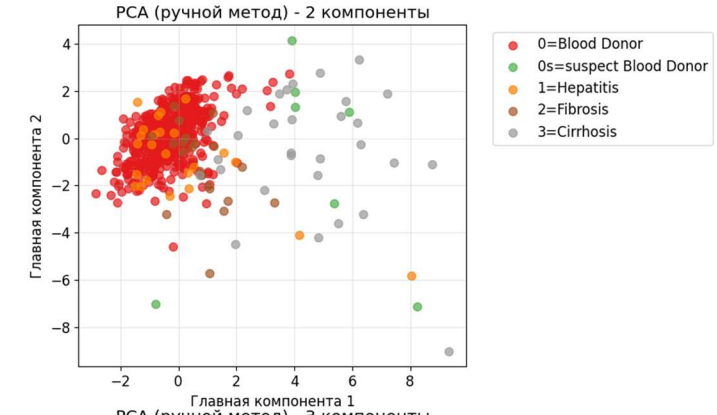
```
import torch
import torch.nn as nn
import torch.optim as optim
import torchvision
import torchvision.transforms as transforms
import matplotlib.pyplot as plt
import numpy as np
from torch.utils.data import DataLoader
import requests
from PIL import Image
import io
import os

device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
print(f"Используемое устройство: {device}")

transform = transforms.Compose([
    transforms.ToTensor(),
    transforms.Normalize((0.5,), (0.5,))
])

train_dataset = torchvision.datasets.FashionMNIST(
    root='./data', train=True, download=True, transform=transform)
test_dataset = torchvision.datasets.FashionMNIST(
    root='./data', train=False, download=True, transform=transform)

batch_size = 128
train_loader = DataLoader(train_dataset, batch_size=batch_size, shuffle=True)
test_loader = DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

classes = ['T-shirt/top', 'Trouser', 'Pullover', 'Dress', 'Coat',
           'Sandal', 'Shirt', 'Sneaker', 'Bag', 'Ankle boot']


class SimpleCNN(nn.Module):
    def __init__(self):
        super(SimpleCNN, self).__init__()
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, kernel_size=3, padding=1)

        self.pool = nn.MaxPool2d(2, 2)

        self.fc1 = nn.Linear(128 * 3 * 3, 256)
        self.fc2 = nn.Linear(256, 128)
        self.fc3 = nn.Linear(128, 10)

        self.relu = nn.ReLU()
        self.dropout = nn.Dropout(0.25)

    def forward(self, x):
        x = self.pool(self.relu(self.conv1(x)))
        x = self.pool(self.relu(self.conv2(x)))
        x = self.pool(self.relu(self.conv3(x)))

        x = x.view(-1, 128 * 3 * 3)

        x = self.relu(self.fc1(x))
        x = self.dropout(x)
        x = self.relu(self.fc2(x))
        x = self.dropout(x)
        x = self.fc3(x)
```

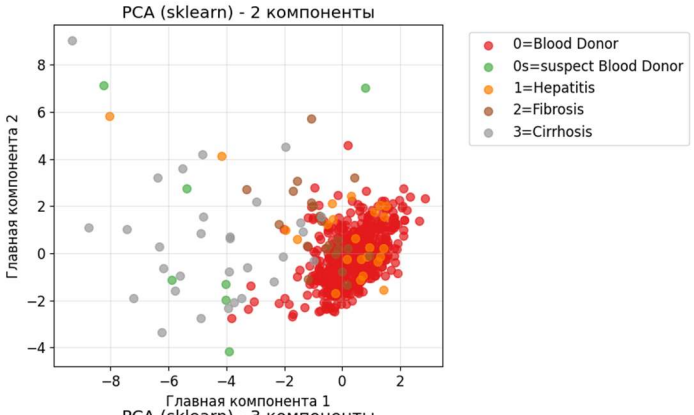
```
        return x


model = SimpleCNN().to(device)
print(model)

criterion = nn.CrossEntropyLoss()
optimizer = optim.Adadelta(model.parameters())


def train_model(model, train_loader, criterion, optimizer, num_epochs=15):
    train_losses = []
    train_accuracies = []

    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0

        for batch_idx, (data, target) in enumerate(train_loader):
            data, target = data.to(device), target.to(device)

            optimizer.zero_grad()
            output = model(data)
            loss = criterion(output, target)
            loss.backward()
            optimizer.step()

            running_loss += loss.item()
            _, predicted = torch.max(output.data, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()

            if batch_idx % 100 == 0:
                print(
                    f'Epoch: {epoch + 1}/{num_epochs} | Batch: {batch_idx}/{len(train_loader)} | Loss: {loss.item():.4f}')

        epoch_loss = running_loss / len(train_loader)
        epoch_accuracy = 100 * correct / total

        train_losses.append(epoch_loss)
        train_accuracies.append(epoch_accuracy)

        print(f'Epoch {epoch + 1}/{num_epochs} | Loss: {epoch_loss:.4f} | Accuracy: {epoch_accuracy:.2f}%')

    return train_losses, train_accuracies


print("Начало обучения...")
train_losses, train_accuracies = train_model(model, train_loader, criterion, optimizer, num_epochs=15)


def evaluate_model(model, test_loader):
    model.eval()
    correct = 0
    total = 0
    test_loss = 0

    with torch.no_grad():
        for data, target in test_loader:
            data, target = data.to(device), target.to(device)
```

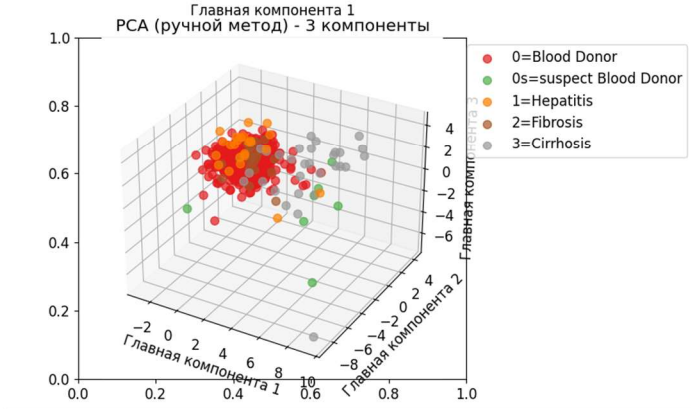
```
            output = model(data)
            test_loss += criterion(output, target).item()
            _, predicted = torch.max(output.data, 1)
            total += target.size(0)
            correct += (predicted == target).sum().item()

    accuracy = 100 * correct / total
    avg_loss = test_loss / len(test_loader)

    print(f'Test Loss: {avg_loss:.4f} | Test Accuracy: {accuracy:.2f}%')
    return accuracy, avg_loss


print("\nОценка на тестовой выборке:")
test_accuracy, test_loss = evaluate_model(model, test_loader)

plt.figure(figsize=(15, 5))

plt.subplot(1, 2, 1)
plt.plot(train_losses, 'b-', label='Training Loss')
plt.axhline(y=test_loss, color='r', linestyle='--', label=f'Test Loss: {test_loss:.4f}')
plt.title('Изменение ошибки во время обучения')
plt.xlabel('Эпоха')
plt.ylabel('Loss')
plt.legend()
plt.grid(True)

plt.subplot(1, 2, 2)
plt.plot(train_accuracies, 'g-', label='Training Accuracy')
plt.axhline(y=test_accuracy, color='orange', linestyle='--', label=f'Test Accuracy: {test_accuracy:.2f}%')
plt.title('Изменение точности во время обучения')
plt.xlabel('Эпоха')
plt.ylabel('Accuracy (%)')
plt.legend()
plt.grid(True)

plt.tight_layout()
plt.savefig('training_results.png', dpi=300, bbox_inches='tight')
plt.show()


def visualize_predictions(model, test_loader, num_images=12):
    model.eval()
    data_iter = iter(test_loader)
    images, labels = next(data_iter)
    images, labels = images.to(device), labels.to(device)

    with torch.no_grad():
        outputs = model(images[:num_images])
        _, predicted = torch.max(outputs, 1)

    images = images.cpu()

    fig, axes = plt.subplots(3, 4, figsize=(15, 10))
    for i, ax in enumerate(axes.flat):
        if i < num_images:
            img = images[i].squeeze() * 0.5 + 0.5
            ax.imshow(img, cmap='gray')
            ax.set_title(f'True: {classes[labels[i]]}\nPred: {classes[predicted[i]]}',
                         color='green' if labels[i] == predicted[i] else 'red')
            ax.axis('off')

    plt.tight_layout()
```

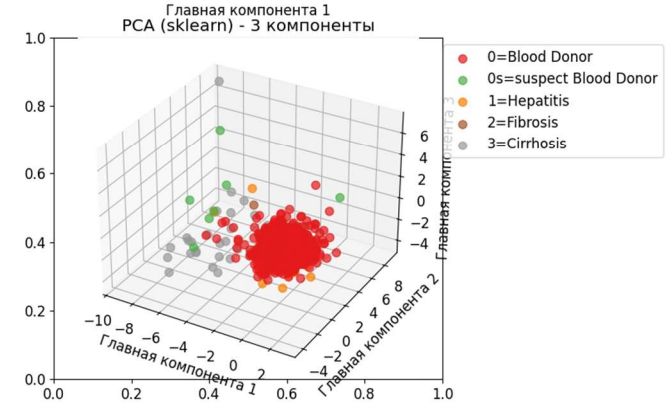
```
    plt.savefig('predictions_visualization.png', dpi=300, bbox_inches='tight')
    plt.show()


print("\nВизуализация предсказаний на тестовых изображениях:")
visualize_predictions(model, test_loader)


def predict_single_image(model, image_path=None, url=None):
    model.eval()

    if url:
        response = requests.get(url)
        image = Image.open(io.BytesIO(response.content))
    else:
        image = Image.open(image_path)

    transform_single = transforms.Compose([
        transforms.Grayscale(),
        transforms.Resize((28, 28)),
        transforms.ToTensor(),
        transforms.Normalize((0.5,), (0.5,))
    ])

    image_tensor = transform_single(image).unsqueeze(0).to(device)

    with torch.no_grad():
        output = model(image_tensor)
        probabilities = torch.nn.functional.softmax(output[0], dim=0)
        _, predicted = torch.max(output, 1)

    plt.figure(figsize=(8, 4))

    plt.subplot(1, 2, 1)
    plt.imshow(image.convert('L'), cmap='gray')
    plt.title(f'Предсказание: {classes[predicted.item()]}')
    plt.axis('off')

    plt.subplot(1, 2, 2)
    y_pos = np.arange(len(classes))
    plt.barh(y_pos, probabilities.cpu().numpy())
    plt.yticks(y_pos, classes)
    plt.xlabel('Вероятность')
    plt.title('Распределение вероятностей')
    plt.tight_layout()

    plt.savefig('single_prediction.png', dpi=300, bbox_inches='tight')
    plt.show()

    return classes[predicted.item()], probabilities.cpu().numpy()
```

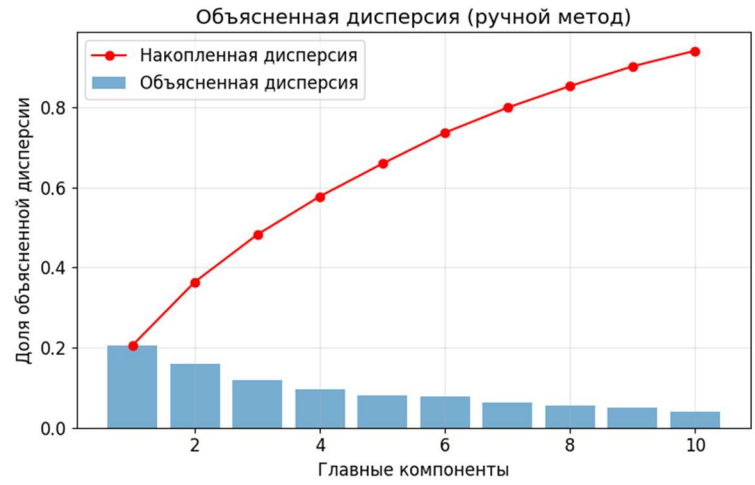
Вывод программы:

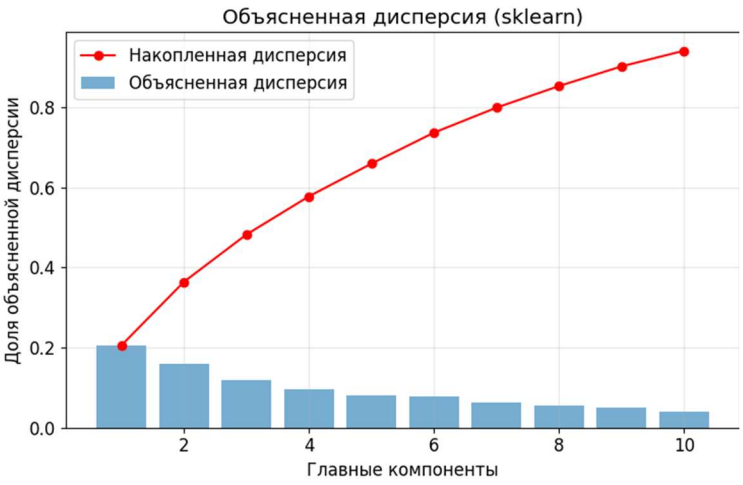












1. Загрузка и предобработка данных...
Информация о данных:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 615 entries, 0 to 614
Data columns (total 14 columns):

#	Column	Non-Null Count	Dtype
0	Unnamed: 0	615 non-null	int64
1	Category	615 non-null	object
2	Age	615 non-null	int64
3	Sex	615 non-null	object
4	ALB	614 non-null	float64
5	ALP	597 non-null	float64
6	ALT	614 non-null	float64
7	AST	615 non-null	float64

```
 8   BIL         615 non-null    float64
 9   CHE         615 non-null    float64
 10  CHOL        605 non-null    float64
 11  CREA        615 non-null    float64
 12  GGT         615 non-null    float64
 13  PROT        614 non-null    float64
dtypes: float64(10), int64(2), object(2)
memory usage: 67.4+ KB
None

Первые 5 строк:
   Unnamed: 0      Category  Age Sex   ALB  ...    CHE  CHOL   CREA   GGT  PROT
0           1  0=Blood Donor   32   m  38.5  ...   6.93  3.23  106.0  12.1  69.0
1           2  0=Blood Donor   32   m  38.5  ...  11.17  4.80   74.0  15.6  76.5
2           3  0=Blood Donor   32   m  46.9  ...   8.84  5.20   86.0  33.2  79.3
3           4  0=Blood Donor   32   m  43.2  ...   7.33  4.74   80.0  33.8  75.7
4           5  0=Blood Donor   32   m  39.2  ...   9.15  4.32   76.0  29.9  68.7

[5 rows x 14 columns]

Колонки в файле: ['Unnamed: 0', 'Category', 'Age', 'Sex', 'ALB', 'ALP', 'ALT', 'AST', 'BIL', 'CHE', 'CHOL', 'CREA', 'GGT', 'PROT']
Размерность данных: (615, 14)

Пропущенные значения:
Unnamed: 0     0
Category       0
Age            0
Sex            0
ALB            1
ALP           18
ALT            1
AST            0
BIL            0
CHE            0
CHOL          10
CREA           0
GGT            0
PROT           1
dtype: int64

Признаки для PCA: ['Age', 'Sex', 'ALB', 'ALP', 'ALT', 'AST', 'BIL', 'CHE', 'CHOL', 'CREA', 'GGT', 'PROT']
Кодируем переменную 'Sex'...
Замена пропущенных значений...
Стандартизация данных...
Размерность после предобработки: (615, 12)
Размерность данных после предобработки: (615, 12)
Количество признаков: 12
Количество наблюдений: 615
Уникальные категории: ['0=Blood Donor' '0s=suspect Blood Donor' '1=Hepatitis' '2=Fibrosis'
 '3=Cirrhosis']

2. Применение PCA...
2.1 Ручной метод с numpy.linalg.eig...
2.2 Метод с sklearn.decomposition.PCA...

3. Визуализация результатов...

4. Расчет потерь информации...
```

Результаты анализа потерь информации:
==
Ручной метод:
  - Объясненная дисперсия (2 компоненты): 0.364 (36.4%)
  - Потери информации (2 компоненты): 0.636 (63.6%)
  - Объясненная дисперсия (3 компоненты): 0.482 (48.2%)
  - Потери информации (3 компоненты): 0.518 (51.8%)

Sklearn метод:
  - Объясненная дисперсия (2 компоненты): 0.364 (36.4%)
  - Потери информации (2 компоненты): 0.636 (63.6%)
  - Объясненная дисперсия (3 компоненты): 0.482 (48.2%)
  - Потери информации (3 компоненты): 0.518 (51.8%)

Анализ важности признаков в главных компонентах:
==

Главная компонента 1:
  ALB: -0.444
  CHE: -0.415
  AST: 0.369
  GGT: 0.349
  BIL: 0.342

Главная компонента 2:
  GGT: -0.449
  ALT: -0.428
  ALP: -0.344
  PROT: -0.315
  CHE: -0.303

Главная компонента 3:
  Age: -0.454
  ALP: -0.424
  CHOL: -0.407
  Sex_m: 0.308
  PROT: 0.300

1. Оба метода (ручной и sklearn) дают схожие результаты.
2. Первые две главные компоненты объясняют значительную часть дисперсии данных.
3. Добавление третьей компоненты уменьшает потери информации.
4. Визуализация показывает возможность разделения классов в пространстве главных компонент.
5. PCA эффективно снижает размерность данных при сохранении основной информации.

Вывод: научился применять метод PCA для осуществления визуализации данных