МИНИСТЕРСТВО ОБРАЗОВАНИЯ РЕСПУБЛИКИ БЕЛАРУСЬ

УЧРЕЖДЕНИЕ ОБРАЗОВАНИЯ

«БРЕСТСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

ФАКУЛЬТЕТ ЭЛЕКТРОННО-ИНФОРМАЦИОННЫХ СИСТЕМ

Кафедра интеллектуальных информационных технологий

Отчет по лабораторной работе №2

Специальность ИИ(з)

Выполнил
А. Ю. Кураш,
студент группы ИИ-24

Проверил
Андренко К.В,
Преподаватель-стажер кафедры ИИТ,
«___k _____2025 г.

Брест 2025

Цель: научиться применять автоэнкодеры для осуществления визуализации данных и их анализа

Общее задание

1. Используя выборку по варианту, осуществить проецирование данных на плоскость первых двух и трех главных компонент с использованием нейросетевой модели автоэнкодера (с двумя и тремя нейронами в среднем слое);

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

3. Реализовать метод t-SNE для визуализации данных (использовать также 2 и 3 компонента), построить соответствующую визуализацию;

4. Применить к данным метод PCA (2 и 3 компонента), реализованный в ЛР №1, сделать выводы;

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

Вариант:

8	Rice (Cammeo and Osmancik)	Class

Код программы:

```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
from sklearn.preprocessing import LabelEncoder, StandardScaler
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings('ignore')

# ============================================================
# 1. ЗАГРУЗКА И ПРЕДОБРАБОТКА
# ============================================================
print("Датасет: Rice (Cammeo and Osmancik) | Целевой класс: Class")
print("=" * 80)

# Рабочий источник CSV
url = "https://download.mlcc.google.com/mledu-datasets/Rice_Cammeo_Osmancik.csv"
```

```
data = pd.read_csv(url)

print(f"✓ Данные загружены: {data.shape[0]} образцов, {data.shape[1]} признаков")

# Разделяем на X и y
X = data.drop("Class", axis=1)
y = data["Class"]

# Кодируем метки классов
le = LabelEncoder()
y_encoded = le.fit_transform(y)  # Cammeo=0, Osmancik=1

print(f"  Классы: {list(le.classes_)}")
print(f"  Размер X: {X.shape}, y: {y_encoded.shape}")

# Нормализация признаков
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X)

# PyTorch тензоры
X_tensor = torch.FloatTensor(X_scaled)
y_tensor = torch.LongTensor(y_encoded)

# ============================================================
# 2. МОДЕЛЬ АВТОЭНКОДЕРА
# ============================================================
class Autoencoder(nn.Module):
    def __init__(self, input_dim, encoding_dim):
        super(Autoencoder, self).__init__()
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 32),
            nn.ReLU(),
            nn.Linear(32, encoding_dim)
        )
        self.decoder = nn.Sequential(
            nn.Linear(encoding_dim, 32),
            nn.ReLU(),
            nn.Linear(32, input_dim)
        )

    def forward(self, x):
        encoded = self.encoder(x)
        decoded = self.decoder(encoded)
        return decoded

    def encode(self, x):
        return self.encoder(x)

def train_autoencoder(model, data, epochs=100, batch_size=64, lr=0.001):
    criterion = nn.MSELoss()
    optimizer = optim.Adam(model.parameters(), lr=lr)
    dataset = TensorDataset(data)
```

```
    dataloader = DataLoader(dataset, batch_size=batch_size, shuffle=True)
    losses = []

    for epoch in range(epochs):
        total_loss = 0
        for batch in dataloader:
            x = batch[0]
            output = model(x)
            loss = criterion(output, x)
            optimizer.zero_grad()
            loss.backward()
            optimizer.step()
            total_loss += loss.item()
        losses.append(total_loss / len(dataloader))
        if (epoch + 1) % 20 == 0:
            print(f"  Эпоха {epoch+1}/{epochs} | Loss = {losses[-1]:.6f}")
    return losses

# ============================================================================
# 3. ОБУЧЕНИЕ
# ============================================================================
input_dim = X_scaled.shape[1]
print(f"\nРазмерность признаков: {input_dim}")

print("\n[1] Обучение автоэнкодера 2D...")
autoencoder_2d = Autoencoder(input_dim, 2)
losses_2d = train_autoencoder(autoencoder_2d, X_tensor, epochs=100)

print("\n[2] Обучение автоэнкодера 3D...")
autoencoder_3d = Autoencoder(input_dim, 3)
losses_3d = train_autoencoder(autoencoder_3d, X_tensor, epochs=100)

# Кодированные представления
with torch.no_grad():
    encoded_2d = autoencoder_2d.encode(X_tensor).numpy()
    encoded_3d = autoencoder_3d.encode(X_tensor).numpy()

print("✓ Автоэнкодеры обучены")
print(f"  2D: {encoded_2d.shape} | 3D: {encoded_3d.shape}")

# ============================================================================
# 4. PCA и t-SNE
# ============================================================================
print("\nПрименение PCA и t-SNE...")

# PCA
pca_2d = PCA(n_components=2).fit_transform(X_scaled)
pca_3d = PCA(n_components=3).fit_transform(X_scaled)

# t-SNE
tsne_2d = TSNE(n_components=2, perplexity=30, random_state=42).fit_transform(X_scaled)
tsne_3d = TSNE(n_components=3, perplexity=30, random_state=42).fit_transform(X_scaled)
```

```
print("✓ Готово.")

# ============================================================
# 5. ВИЗУАЛИЗАЦИЯ
# ============================================================
print("\nСоздание графиков...")

colors = ['#2ecc71', '#e74c3c']
labels = list(le.classes_)

fig = plt.figure(figsize=(20, 10))

# Autoencoder 2D
ax1 = plt.subplot(2, 3, 1)
for i in range(2):
    mask = y_encoded == i
    ax1.scatter(encoded_2d[mask, 0], encoded_2d[mask, 1],
                c=colors[i], label=labels[i], alpha=0.6)
ax1.set_title("Autoencoder (2D)")
ax1.legend()
ax1.grid(True)

# PCA 2D
ax2 = plt.subplot(2, 3, 2)
for i in range(2):
    mask = y_encoded == i
    ax2.scatter(pca_2d[mask, 0], pca_2d[mask, 1],
                c=colors[i], label=labels[i], alpha=0.6)
ax2.set_title("PCA (2D)")
ax2.legend()
ax2.grid(True)

# t-SNE 2D
ax3 = plt.subplot(2, 3, 3)
for i in range(2):
    mask = y_encoded == i
    ax3.scatter(tsne_2d[mask, 0], tsne_2d[mask, 1],
                c=colors[i], label=labels[i], alpha=0.6)
ax3.set_title("t-SNE (2D)")
ax3.legend()
ax3.grid(True)

# 3D Autoencoder
ax4 = fig.add_subplot(2, 3, 4, projection='3d')
for i in range(2):
    mask = y_encoded == i
    ax4.scatter(encoded_3d[mask, 0], encoded_3d[mask, 1], encoded_3d[mask, 2],
                c=colors[i], label=labels[i], alpha=0.6)
ax4.set_title("Autoencoder (3D)")
ax4.legend()
```

```
# 3D PCA
ax5 = fig.add_subplot(2, 3, 5, projection='3d')
for i in range(2):
    mask = y_encoded == i
    ax5.scatter(pca_3d[mask, 0], pca_3d[mask, 1], pca_3d[mask, 2],
                c=colors[i], label=labels[i], alpha=0.6)
ax5.set_title("PCA (3D)")
ax5.legend()

# 3D t-SNE
ax6 = fig.add_subplot(2, 3, 6, projection='3d')
for i in range(2):
    mask = y_encoded == i
    ax6.scatter(tsne_3d[mask, 0], tsne_3d[mask, 1], tsne_3d[mask, 2],
                c=colors[i], label=labels[i], alpha=0.6)
ax6.set_title("t-SNE (3D)")
ax6.legend()

plt.tight_layout()
plt.savefig("rice_visualization.png", dpi=300, bbox_inches='tight')
print("✓ Визуализация сохранена: rice_visualization.png")

# ============================================================
# 6. ВЫВОДЫ
# ============================================================
print("\n" + "=" * 80)
print("ВЫВОДЫ:")
print("• Автоэнкодер успешно обучен, показывает разделение между сортами риса")
print("• PCA сохраняет большую часть дисперсии, но не всегда идеально разделяет классы")
print("• t-SNE лучше визуализирует кластеры")
print("• Все три метода показывают чёткое разделение Cammeo и Osmancik")
print("=" * 80)

plt.show()
```

Вывод: Я научился применять автоэнкодеры для осуществления визуализации данных и их анализа.