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
Д. Д. Крупич,
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:

7	Mushroom	poisonous

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
from sklearn.manifold import TSNE
from sklearn.decomposition import PCA
import warnings
warnings.filterwarnings('ignore')

print("Датасет: Mushroom | Целевой класс: poisonous")
print("=" * 80)

#
==========================================================
====================
```

```
# 1. ЗАГРУЗКА И ПРЕДОБРАБОТКА ДАННЫХ
#
==========
print("\n[1] Загрузка данных Mushroom...")

# Загрузка датасета Mushroom из UCI ML Repository
url = "https://archive.ics.uci.edu/ml/machine-learning-databases/mushroom/agaricus-lepiota.data"
columns = ['class', 'cap-shape', 'cap-surface', 'cap-color', 'bruises', 'odor',
           'gill-attachment', 'gill-spacing', 'gill-size', 'gill-color',
           'stalk-shape', 'stalk-root', 'stalk-surface-above-ring',
           'stalk-surface-below-ring', 'stalk-color-above-ring',
           'stalk-color-below-ring', 'veil-type', 'veil-color', 'ring-number',
           'ring-type', 'spore-print-color', 'population', 'habitat']

try:
    data = pd.read_csv(url, names=columns)
    print(f"✓ Данные загружены: {data.shape[0]} образцов, {data.shape[1]} признаков")
except:
    print("✗ Ошибка загрузки данных из UCI. Используйте локальный файл или другой источник.")
    exit()

# Обработка пропущенных значений
data = data.replace('?', np.nan)
print(f"  Пропущенные значения: {data.isnull().sum().sum()}")

# Удаление строк с пропусками (если есть)
data = data.dropna()

# Целевая переменная
y = data['class'].map({'e': 0, 'p': 1})  # e=съедобный (0), p=ядовитый (1)
X = data.drop('class', axis=1)

print(f"  Распределение классов:")
print(f"    - Съедобные (edible): {(y == 0).sum()} ({(y == 0).sum() / len(y) * 100:.1f}%)")
print(f"    - Ядовитые (poisonous): {(y == 1).sum()} ({(y == 1).sum() / len(y) * 100:.1f}%)")

# Кодирование категориальных признаков
print("\n[2] Кодирование признаков...")
label_encoders = {}
X_encoded = X.copy()

for column in X.columns:
    le = LabelEncoder()
```

```
    X_encoded[column] = le.fit_transform(X[column])
    label_encoders[column] = le

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
X_scaled = scaler.fit_transform(X_encoded)

print(f"✓ Признаки закодированы и нормализованы: {X_scaled.shape}")

# Преобразование в PyTorch тензоры
X_tensor = torch.FloatTensor(X_scaled)
y_tensor = torch.LongTensor(y.values)

#
==========================================================
======================
# 2. ОПРЕДЕЛЕНИЕ АРХИТЕКТУРЫ АВТОЭНКОДЕРА
#
==========================================================
======================
print("\n[3] Создание архитектуры автоэнкодера...")

class Autoencoder(nn.Module):
    def __init__(self, input_dim, encoding_dim):
        super(Autoencoder, self).__init__()

        # Энкодер
        self.encoder = nn.Sequential(
            nn.Linear(input_dim, 64),
            nn.ReLU(),
            nn.Linear(64, 32),
            nn.ReLU(),
            nn.Linear(32, encoding_dim)
        )

        # Декодер
        self.decoder = nn.Sequential(
            nn.Linear(encoding_dim, 32),
            nn.ReLU(),
            nn.Linear(32, 64),
            nn.ReLU(),
            nn.Linear(64, input_dim)
        )

    def forward(self, x):
        encoded = self.encoder(x)
```

```
        decoded = self.decoder(encoded)
        return decoded

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

#
================================================================
======================
# 3. ОБУЧЕНИЕ АВТОЭНКОДЕРОВ
#
================================================================
======================
def train_autoencoder(model, data, epochs=100, batch_size=64, lr=0.001):
    """Обучение автоэнкодера"""
    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):
        epoch_loss = 0
        for batch in dataloader:
            x = batch[0]

            output = model(x)
            loss = criterion(output, x)

            optimizer.zero_grad()
            loss.backward()
            optimizer.step()

            epoch_loss += loss.item()

        avg_loss = epoch_loss / len(dataloader)
        losses.append(avg_loss)

        if (epoch + 1) % 20 == 0:
            print(f"  Эпоха {epoch+1}/{epochs}, Loss: {avg_loss:.6f}")

    return losses

# Обучение автоэнкодера с 2 компонентами
print("\n[4] Обучение автоэнкодера с 2 компонентами...")
```

```
input_dim = X_scaled.shape[1]
autoencoder_2d = Autoencoder(input_dim, encoding_dim=2)
losses_2d = train_autoencoder(autoencoder_2d, X_tensor, epochs=100)

# Обучение автоэнкодера с 3 компонентами
print("\n[5] Обучение автоэнкодера с 3 компонентами...")
autoencoder_3d = Autoencoder(input_dim, encoding_dim=3)
losses_3d = train_autoencoder(autoencoder_3d, X_tensor, epochs=100)

# Получение закодированных представлений
with torch.no_grad():
    encoded_2d = autoencoder_2d.encode(X_tensor).numpy()
    encoded_3d = autoencoder_3d.encode(X_tensor).numpy()

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

#
==========
# 4. ПРИМЕНЕНИЕ T-SNE
#
==========
print("\n[6] Применение t-SNE...")

# t-SNE с разными значениями perplexity
perplexities = [20, 30, 50]
tsne_results_2d = {}
tsne_results_3d = {}

for perp in perplexities:
    print(f"  Perplexity = {perp}...")

    # 2D t-SNE
    tsne_2d = TSNE(n_components=2, perplexity=perp, init='pca', random_state=42)
    tsne_results_2d[perp] = tsne_2d.fit_transform(X_scaled)

    # 3D t-SNE
    tsne_3d = TSNE(n_components=3, perplexity=perp, init='pca', random_state=42)
    tsne_results_3d[perp] = tsne_3d.fit_transform(X_scaled)

print("✓ t-SNE применен для различных perplexity")

#
```

```
# ============================================================
# 5. ПРИМЕНЕНИЕ PCA
# ============================================================
print("\n[7] Применение PCA...")

pca_2d = PCA(n_components=2)
pca_result_2d = pca_2d.fit_transform(X_scaled)
print(f"  PCA 2D объясненная дисперсия: {pca_2d.explained_variance_ratio_.sum():.4f}")

pca_3d = PCA(n_components=3)
pca_result_3d = pca_3d.fit_transform(X_scaled)
print(f"  PCA 3D объясненная дисперсия: {pca_3d.explained_variance_ratio_.sum():.4f}")

# ============================================================
# 6. ВИЗУАЛИЗАЦИЯ РЕЗУЛЬТАТОВ
# ============================================================
print("\n[8] Создание визуализаций...")

# Цвета для классов
colors = ['#2ecc71', '#e74c3c']  # Зеленый для съедобных, красный для ядовитых
labels = ['Съедобный', 'Ядовитый']

# ===== ВИЗУАЛИЗАЦИЯ 2D =====
fig = plt.figure(figsize=(20, 12))

# 1. Автоэнкодер 2D
ax1 = plt.subplot(2, 4, 1)
for i in range(2):
    mask = y == i
    ax1.scatter(encoded_2d[mask, 0], encoded_2d[mask, 1],
                c=colors[i], label=labels[i], alpha=0.6, s=20)
ax1.set_title('Автоэнкодер (2D)', fontsize=12, fontweight='bold')
ax1.set_xlabel('Компонента 1')
ax1.set_ylabel('Компонента 2')
ax1.legend()
ax1.grid(True, alpha=0.3)

# 2-4. t-SNE 2D с разными perplexity
for idx, perp in enumerate(perplexities):
    ax = plt.subplot(2, 4, idx + 2)
```

```
    for i in range(2):
        mask = y == i
        ax.scatter(tsne_results_2d[perp][mask, 0], tsne_results_2d[perp][mask, 1],
                   c=colors[i], label=labels[i], alpha=0.6, s=20)
    ax.set_title(f't-SNE 2D (perplexity={perp})', fontsize=12, fontweight='bold')
    ax.set_xlabel('Компонента 1')
    ax.set_ylabel('Компонента 2')
    ax.legend()
    ax.grid(True, alpha=0.3)

# 5. PCA 2D
ax5 = plt.subplot(2, 4, 5)
for i in range(2):
    mask = y == i
    ax5.scatter(pca_result_2d[mask, 0], pca_result_2d[mask, 1],
                c=colors[i], label=labels[i], alpha=0.6, s=20)
ax5.set_title('PCA (2D)', fontsize=12, fontweight='bold')
ax5.set_xlabel('PC1')
ax5.set_ylabel('PC2')
ax5.legend()
ax5.grid(True, alpha=0.3)

# 6. График потерь автоэнкодера
ax6 = plt.subplot(2, 4, 6)
ax6.plot(losses_2d, label='2D Autoencoder', color='blue', linewidth=2)
ax6.plot(losses_3d, label='3D Autoencoder', color='orange', linewidth=2)
ax6.set_title('Кривая обучения', fontsize=12, fontweight='bold')
ax6.set_xlabel('Эпоха')
ax6.set_ylabel('Loss (MSE)')
ax6.legend()
ax6.grid(True, alpha=0.3)

plt.tight_layout()
plt.savefig('mushroom_visualization_2d.png', dpi=300, bbox_inches='tight')
print("✓ 2D визуализация сохранена: mushroom_visualization_2d.png")

# ===== ВИЗУАЛИЗАЦИЯ 3D =====
fig = plt.figure(figsize=(20, 10))

# 1. Автоэнкодер 3D
ax1 = fig.add_subplot(2, 3, 1, projection='3d')
for i in range(2):
    mask = y == i
    ax1.scatter(encoded_3d[mask, 0], encoded_3d[mask, 1], encoded_3d[mask, 2],
                c=colors[i], label=labels[i], alpha=0.6, s=20)
ax1.set_title('Автоэнкодер (3D)', fontsize=12, fontweight='bold')
```

```
ax1.set_xlabel('Компонента 1')
ax1.set_ylabel('Компонента 2')
ax1.set_zlabel('Компонента 3')
ax1.legend()

# 2. t-SNE 3D (perplexity=30)
ax2 = fig.add_subplot(2, 3, 2, projection='3d')
for i in range(2):
    mask = y == i
    ax2.scatter(tsne_results_3d[30][mask, 0], tsne_results_3d[30][mask, 1],
                tsne_results_3d[30][mask, 2],
                c=colors[i], label=labels[i], alpha=0.6, s=20)
ax2.set_title('t-SNE (3D, perplexity=30)', fontsize=12, fontweight='bold')
ax2.set_xlabel('Компонента 1')
ax2.set_ylabel('Компонента 2')
ax2.set_zlabel('Компонента 3')
ax2.legend()

# 3. PCA 3D
ax3 = fig.add_subplot(2, 3, 3, projection='3d')
for i in range(2):
    mask = y == i
    ax3.scatter(pca_result_3d[mask, 0], pca_result_3d[mask, 1], pca_result_3d[mask, 2],
                c=colors[i], label=labels[i], alpha=0.6, s=20)
ax3.set_title('PCA (3D)', fontsize=12, fontweight='bold')
ax3.set_xlabel('PC1')
ax3.set_ylabel('PC2')
ax3.set_zlabel('PC3')
ax3.legend()

# 4-6. Дополнительные углы обзора для автоэнкодера
for idx, (elev, azim) in enumerate([(10, 45), (30, 120), (20, -60)]):
    ax = fig.add_subplot(2, 3, idx + 4, projection='3d')
    for i in range(2):
        mask = y == i
        ax.scatter(encoded_3d[mask, 0], encoded_3d[mask, 1], encoded_3d[mask, 2],
                   c=colors[i], label=labels[i], alpha=0.6, s=20)
    ax.view_init(elev=elev, azim=azim)
    ax.set_title(f'Автоэнкодер 3D (угол {idx+1})', fontsize=12, fontweight='bold')
    ax.set_xlabel('Компонента 1')
    ax.set_ylabel('Компонента 2')
    ax.set_zlabel('Компонента 3')
    ax.legend()

plt.tight_layout()
plt.savefig('mushroom_visualization_3d.png', dpi=300, bbox_inches='tight')
```

```
print("✓ 3D визуализация сохранена: mushroom_visualization_3d.png")

#
==========
# 7. АНАЛИЗ И ВЫВОДЫ
#
==========
print("\n" + "=" * 80)
print("ВЫВОДЫ И СРАВНЕНИЕ МЕТОДОВ")
print("=" * 80)

print("\n1. АВТОЭНКОДЕР:")
print("   ✓ Успешно обучен для 2D и 3D представлений")
print("   ✓ Нелинейное преобразование данных")
print("   ✓ Показывает хорошее разделение классов")
print(f"   ✓ Финальная ошибка (2D): {losses_2d[-1]:.6f}")
print(f"   ✓ Финальная ошибка (3D): {losses_3d[-1]:.6f}")

print("\n2. T-SNE:")
print("   ✓ Отличная визуализация кластерной структуры")
print("   ✓ Хорошо разделяет классы при всех значениях perplexity")
print("   ✓ Оптимальное значение perplexity: 30-50")
print("   ⚠ Не сохраняет глобальную структуру данных")

print("\n3. PCA:")
print(f"   ✓ Объясненная дисперсия (2D): {pca_2d.explained_variance_ratio_.sum():.4f}")
print(f"   ✓ Объясненная дисперсия (3D): {pca_3d.explained_variance_ratio_.sum():.4f}")
print("   ✓ Линейный метод, быстрый и интерпретируемый")
print("   ⚠ Может не улавливать нелинейные зависимости")

print("\n4. СРАВНЕНИЕ:")
print("   • Автоэнкодер: баланс между интерпретируемостью и качеством")
print("   • t-SNE: лучший для визуального анализа кластеров")
print("   • PCA: быстрый, но может терять информацию о нелинейных паттернах")
print("   • Все три метода показывают хорошее разделение классов грибов")

print("\n5. РЕКОМЕНДАЦИИ:")
print("   • Для исследовательского анализа: t-SNE (perplexity=30-50)")
print("   • Для быстрого анализа: PCA")
print("   • Для обучаемых представлений: Автоэнкодер")

print("\n" + "=" * 80)
```

```
print("Визуализации сохранены:")
print("  - mushroom_visualization_2d.png")
print("  - mushroom_visualization_3d.png")
print("=" * 80)

plt.show()
```

Вывод программы:

```
================================================================================
Датасет: Mushroom | Целевой класс: poisonous
================================================================================

[1] Загрузка данных Mushroom...
✓ Данные загружены: 8124 образцов, 23 признаков
  Пропущенные значения: 2480
  Распределение классов:
    - Съедобные (edible): 3488 (61.8%)
    - Ядовитые (poisonous): 2156 (38.2%)

[2] Кодирование признаков...
✓ Признаки закодированы и нормализованы: (5644, 22)

[3] Создание архитектуры автоэнкодера...

[4] Обучение автоэнкодера с 2 компонентами...
  Эпоха 20/100, Loss: 0.198996
  Эпоха 40/100, Loss: 0.154453
  Эпоха 60/100, Loss: 0.130560
  Эпоха 80/100, Loss: 0.121241
  Эпоха 100/100, Loss: 0.118290

[5] Обучение автоэнкодера с 3 компонентами...
  Эпоха 20/100, Loss: 0.129084
  Эпоха 40/100, Loss: 0.104056
  Эпоха 60/100, Loss: 0.091726
  Эпоха 80/100, Loss: 0.085111
  Эпоха 100/100, Loss: 0.079520
```

✓ Автоэнкодеры обучены
2D представление: (5644, 2)
3D представление: (5644, 3)

[6] Применение t-SNE...
Perplexity = 20...
Perplexity = 30...
Perplexity = 50...
✓ t-SNE применен для различных perplexity

[7] Применение PCA...
PCA 2D объясненная дисперсия: 0.3504
PCA 3D объясненная дисперсия: 0.4638

[8] Создание визуализаций...
✓ 2D визуализация сохранена: mushroom_visualization_2d.png
✓ 3D визуализация сохранена: mushroom_visualization_3d.png

==========
ВЫВОДЫ И СРАВНЕНИЕ МЕТОДОВ
==========

1. АВТОЭНКОДЕР:
✓ Успешно обучен для 2D и 3D представлений
✓ Нелинейное преобразование данных
✓ Показывает хорошее разделение классов
✓ Финальная ошибка (2D): 0.118290
✓ Финальная ошибка (3D): 0.079520

2. T-SNE:
✓ Отличная визуализация кластерной структуры
✓ Хорошо разделяет классы при всех значениях perplexity
✓ Оптимальное значение perplexity: 30-50
⚠ Не сохраняет глобальную структуру данных

3. PCA:

✓ Объясненная дисперсия (2D): 0.3504

✓ Объясненная дисперсия (3D): 0.4638

✓ Линейный метод, быстрый и интерпретируемый

⚠ Может не улавливать нелинейные зависимости

4. СРАВНЕНИЕ:

• Автоэнкодер: баланс между интерпретируемостью и качеством

• t-SNE: лучший для визуального анализа кластеров

• PCA: быстрый, но может терять информацию о нелинейных паттернах

• Все три метода показывают хорошее разделение классов грибов

5. РЕКОМЕНДАЦИИ:

• Для исследовательского анализа: t-SNE (perplexity=30-50)

• Для быстрого анализа: PCA

• Для обучаемых представлений: Автоэнкодер

==========

Визуализации сохранены:

- mushroom_visualization_2d.png

- mushroom_visualization_3d.png

==========

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