Министерство образования Республики Беларусь

Учреждение образования

«Брестский Государственный технический университет»

Кафедра ИИТ

Лабораторная работа №3

По дисциплине «Интеллектуальный анализ данных»

Тема: «Предобучение нейронных сетей с использованием автоэнкодерного подхода»

Выполнила:

Студентка 4 курса

Группы ИИ-24

Максимович А. И.

Проверила:

Андренко К. В.

Брест 2025

Цель: научиться осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода

Общее задание

1. Взять за основу любую сверточную или полносвязную архитектуру с количеством слоев более 3. Осуществить ее обучение (без предобучения) в соответствии с вариантом задания. Получить оценку эффективности модели, используя метрики, специфичные для решаемой задачи (например, MAPE – для регрессионной задачи или F1/Confusion matrix для классификационной).

2. Выполнить обучение с предобучением, используя автоэнкодерный подход, алгоритм которого изложен в лекции. Условие останова (например, по количеству эпох) при обучении отдельных слоев с использованием автоэнкодера выбрать самостоятельно.

3. Сравнить результаты, полученные при обучении с/без предобучения, сделать выводы.

4. Выполните пункты 1-3 для датасетов из ЛР 2.

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

Задание по вариантам

№ в-а	Выборка	Тип задачи	Целевая переменная
11	https://archive.ics.uci.edu/dataset/27/credit+approval	классификация	+/-

Код:

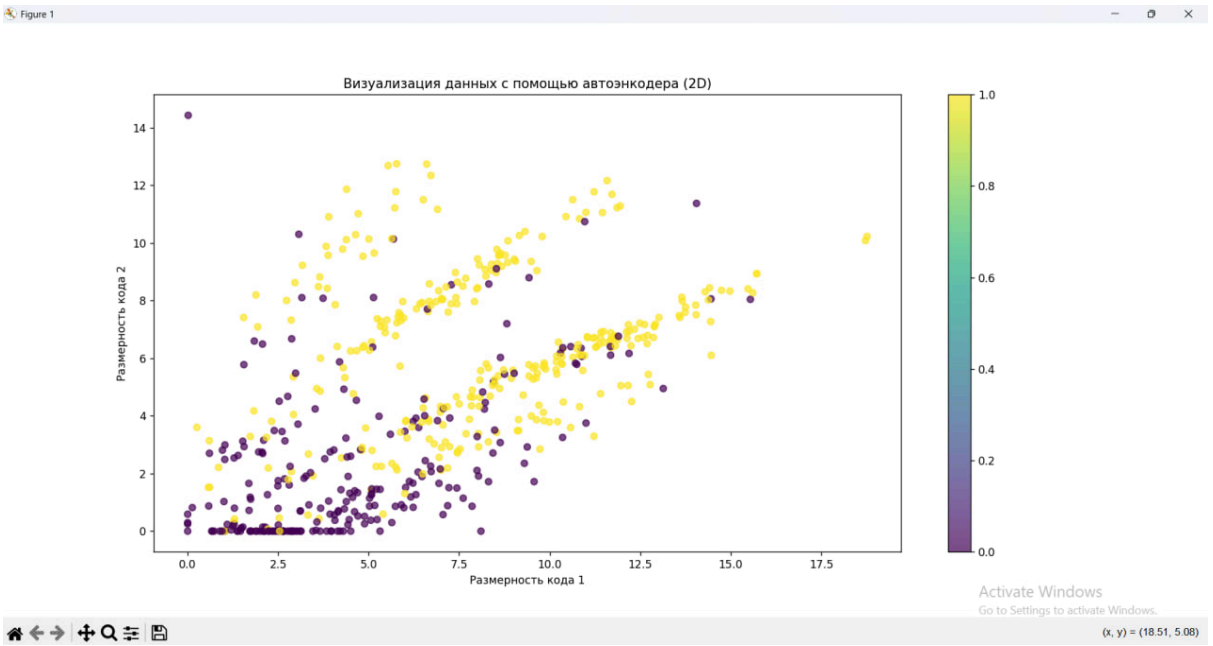
```
from ucimlrepo import fetch_ucirepo
import pandas as pd
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.model_selection import train_test_split

# Загрузка данных
credit_approval = fetch_ucirepo(id=27)
X = credit_approval.data.features
y = credit_approval.data.targets

# Предварительная обработка
# Заполнение пропущенных значений и кодирование категориальных признаков
X = X.fillna(X.mode().iloc[0])
for column in X.select_dtypes(include=['object']).columns:
    le = LabelEncoder()
    X[column] = le.fit_transform(X[column])
```

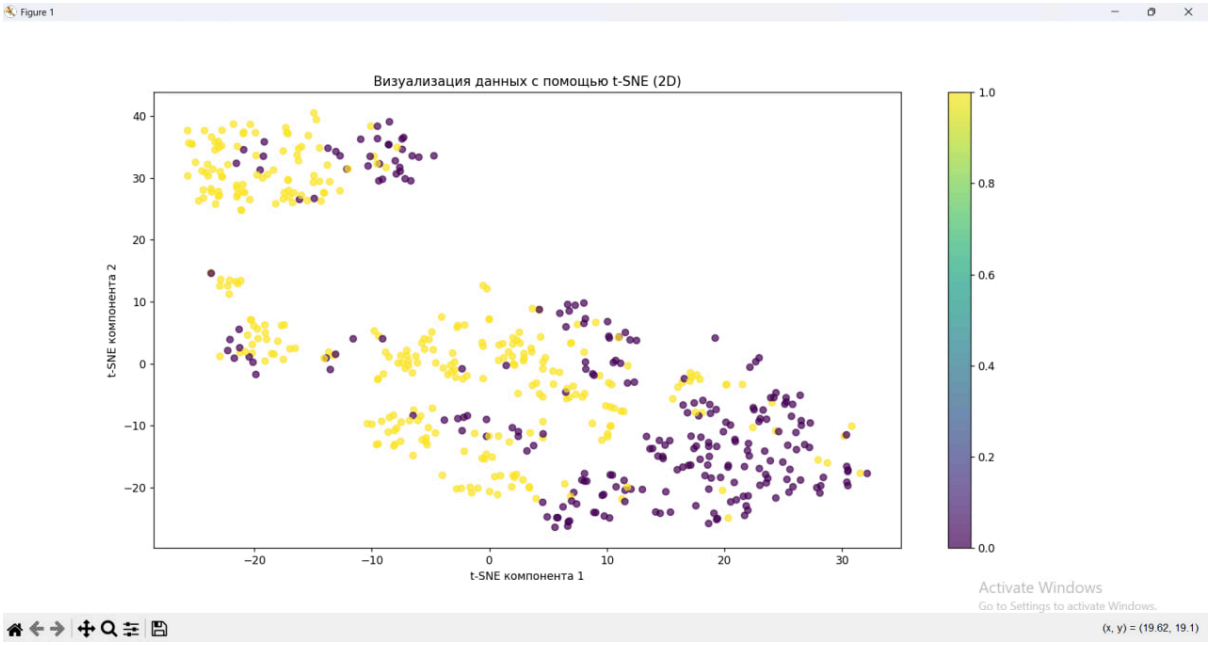
```
# Кодирование целевой переменной
y = LabelEncoder().fit_transform(y)

# Разделение на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
random_state=42)

# Нормализация данных
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)

from tensorflow import keras
from tensorflow.keras import layers

# Размерность входных данных
input_dim = X_train_scaled.shape[1]
encoding_dim = 5  # Можно экспериментировать с размерностью кода

# Энкодер
input_layer = layers.Input(shape=(input_dim,))
encoded = layers.Dense(32, activation='relu')(input_layer)
encoded = layers.Dense(16, activation='relu')(encoded)
encoded = layers.Dense(encoding_dim, activation='relu')(encoded)  # Код

# Декодер
decoded = layers.Dense(16, activation='relu')(encoded)
decoded = layers.Dense(32, activation='relu')(decoded)
decoded = layers.Dense(input_dim, activation='sigmoid')(decoded)

autoencoder = keras.Model(input_layer, decoded)
autoencoder.compile(optimizer='adam', loss='mse')

# Обучение автоэнкодера
history = autoencoder.fit(X_train_scaled, X_train_scaled,
                          epochs=100,
                          batch_size=256,
                          shuffle=True,
                          validation_data=(X_test_scaled, X_test_scaled))

# Создание модели энкодера
encoder = keras.Model(input_layer, encoded)

# "Замораживание" весов энкодера для предобучения
encoder.trainable = False

# Создание модели классификатора с предобученным энкодером
classifier_input = layers.Input(shape=(input_dim,))
classifier_features = encoder(classifier_input)
classifier_hidden = layers.Dense(32, activation='relu')(classifier_features)
classifier_output = layers.Dense(1, activation='sigmoid')(classifier_hidden)
# Для бинарной классификации

classifier = keras.Model(classifier_input, classifier_output)
classifier.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

# Обучение классификатора с замороженным энкодером
history_classifier = classifier.fit(X_train_scaled, y_train,
                                    epochs=50,
                                    batch_size=256,
```

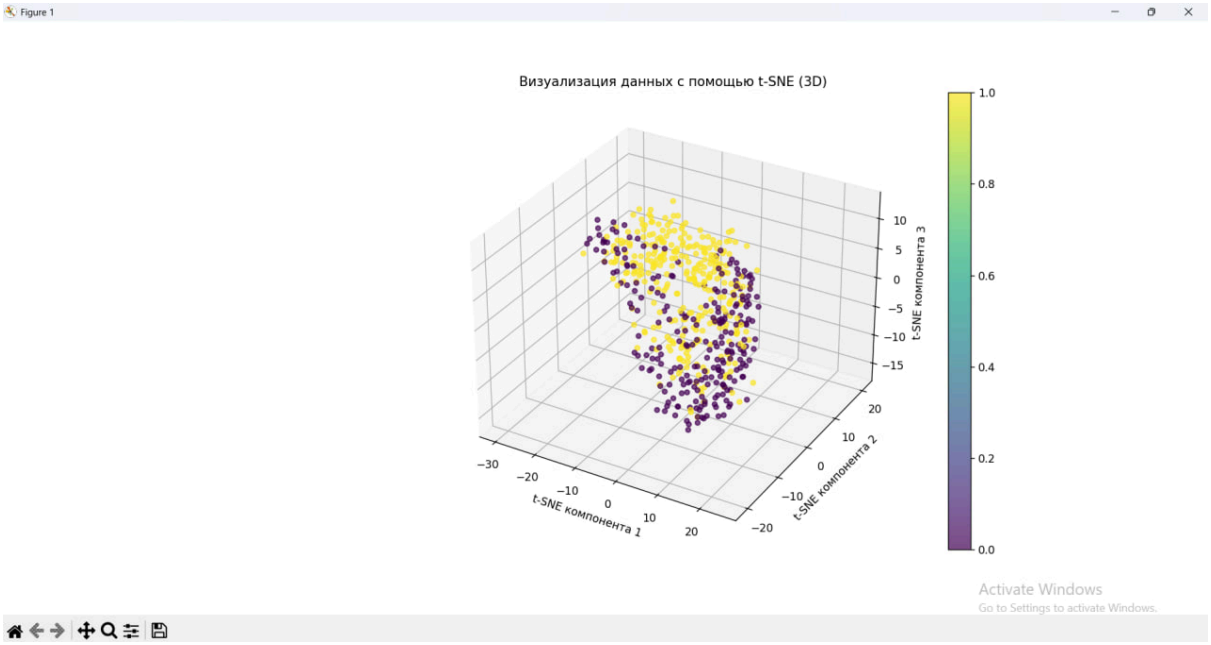
```
                                    validation_data=(X_test_scaled, y_test))

# Разморозка энкодера
encoder.trainable = True

# Перекомпиляция модели для тонкой настройки
classifier.compile(optimizer=keras.optimizers.Adam(1e-5),  # Используем очень
низкую скорость обучения
                   loss='binary_crossentropy',
                   metrics=['accuracy'])

# Дообучение модели
history_finetune = classifier.fit(X_train_scaled, y_train,
                                  epochs=30,
                                  batch_size=256,
                                  validation_data=(X_test_scaled, y_test))

# Модель без предобучения
classifier_scratch = keras.Sequential([
    layers.Dense(32, activation='relu', input_shape=(input_dim,)),
    layers.Dense(16, activation='relu'),
    layers.Dense(1, activation='sigmoid')
])
classifier_scratch.compile(optimizer='adam', loss='binary_crossentropy',
metrics=['accuracy'])

history_scratch = classifier_scratch.fit(X_train_scaled, y_train,
                                         epochs=50,
                                         batch_size=256,
                                         validation_data=(X_test_scaled,
y_test))

# Создание автоэнкодера с 2 нейронами в слое кодирования для визуализации на
плоскости
encoder_2d = keras.Sequential([
    layers.Dense(32, activation='relu', input_shape=(input_dim,)),
    layers.Dense(16, activation='relu'),
    layers.Dense(2, activation='relu')  # Код размерности 2
])
decoder_2d = keras.Sequential([
    layers.Dense(16, activation='relu', input_shape=(2,)),
    layers.Dense(32, activation='relu'),
    layers.Dense(input_dim, activation='sigmoid')
])
autoencoder_2d = keras.Sequential([encoder_2d, decoder_2d])
autoencoder_2d.compile(optimizer='adam', loss='mse')

# Обучение
autoencoder_2d.fit(X_train_scaled, X_train_scaled, epochs=100,
batch_size=256, shuffle=True, verbose=0)

# Получение кодированного представления для всего датасета
X_encoded_2d = encoder_2d.predict(X_train_scaled)

import matplotlib.pyplot as plt

# Визуализация
plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_encoded_2d[:, 0], X_encoded_2d[:, 1], c=y_train,
cmap='viridis', alpha=0.7)
plt.colorbar(scatter)
```

```
plt.title('Визуализация данных с помощью автоэнкодера (2D)')
plt.xlabel('Размерность кода 1')
plt.ylabel('Размерность кода 2')
plt.show()

from sklearn.manifold import TSNE

# Визуализация в 2D с помощью t-SNE
tsne_2d = TSNE(n_components=2, random_state=42, perplexity=30)
X_tsne_2d = tsne_2d.fit_transform(X_train_scaled)

plt.figure(figsize=(10, 8))
scatter = plt.scatter(X_tsne_2d[:, 0], X_tsne_2d[:, 1], c=y_train,
cmap='viridis', alpha=0.7)
plt.colorbar(scatter)
plt.title('Визуализация данных с помощью t-SNE (2D)')
plt.xlabel('t-SNE компонента 1')
plt.ylabel('t-SNE компонента 2')
plt.show()

# Визуализация в 3D с помощью t-SNE
tsne_3d = TSNE(n_components=3, random_state=42, perplexity=30)
X_tsne_3d = tsne_3d.fit_transform(X_train_scaled)

fig = plt.figure(figsize=(10, 8))
ax = fig.add_subplot(111, projection='3d')
scatter = ax.scatter(X_tsne_3d[:, 0], X_tsne_3d[:, 1], X_tsne_3d[:, 2],
c=y_train, cmap='viridis', alpha=0.7)
plt.colorbar(scatter)
ax.set_title('Визуализация данных с помощью t-SNE (3D)')
ax.set_xlabel('t-SNE компонента 1')
ax.set_ylabel('t-SNE компонента 2')
ax.set_zlabel('t-SNE компонента 3')
plt.show()
```

Вывод:







Вывод: научилась осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода