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»

Выполнил:

Студент 4 курса

Группы ИИ-24

Бузель С.Д.

Проверил:

Андренко К. В.

Брест 2025

Цель: научиться осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода.

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.

Датасет ЛР2:

№	Выборка	Класс
3	Rice (Cammeo and Osmancik)	Class

Датасет ЛР3:

№	Выборка	Тип задачи	Целевая переменная
3	https://archive.ics.uci.edu/dataset/863/maternal+health+risk	классификация	RiskLevel

Код программы:

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler, LabelEncoder
from sklearn.metrics import accuracy_score, f1_score
from scipy.io import arff

import tensorflow as tf
from tensorflow.keras.models import Model, Sequential
from tensorflow.keras.layers import Input, Dense

# Отключаем слишком подробные логи TensorFlow
```

```
import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# --- Этап 0: Определение путей к файлам ---
MATERNAL_HEALTH_PATH = "D:/ОИИС/lab3/Maternal Health Risk Data Set.csv"
RICE_DATA_PATH = "D:/ОИИС/lab2/Rice_Cammeo_Osmancik.arff"

# --- Этап 1: Функции для загрузки и подготовки данных ---

def load_maternal_health_data(path):
    print("\nЗагрузка датасета 'Maternal Health Risk'...")
    try:
        data = pd.read_csv(path)
        X = data.drop('RiskLevel', axis=1)
        y = data['RiskLevel']

        # Преобразуем целевую переменную в числа (low risk-> 0, mid risk-> 1, high risk-> 2)
        le = LabelEncoder()
        y_encoded = le.fit_transform(y.values.ravel())

        print(f"Признаков: {X.shape[1]}, Классов: {len(np.unique(y_encoded))}, Объектов: {X.shape[0]}")
        return X.values, y_encoded
    except FileNotFoundError:
        print(f"Ошибка: Файл не найден по пути {path}")
        return None, None

def load_rice_data(path):
    print("\nЗагрузка датасета 'Rice (Cammeo and Osmancik)'...")
    try:
        data_arff, meta = arff.loadarff(path)
        data = pd.DataFrame(data_arff)
        data['Class'] = data['Class'].apply(lambda x: x.decode('utf-8'))

        X = data.drop('Class', axis=1)
        y = data['Class']

        le = LabelEncoder()
        y_encoded = le.fit_transform(y.values.ravel())

        print(f"Признаков: {X.shape[1]}, Классов: {len(np.unique(y_encoded))}, Объектов: {X.shape[0]}")
        return X.values, y_encoded
    except FileNotFoundError:
        print(f"Ошибка: Файл не найден по пути {path}")
        return None, None

# --- Этап 2: Логика обучения моделей ---

def create_dnn_classifier(input_dim, output_dim, layer_sizes):
    """Создает стандартную модель DNN для классификации."""
    model = Sequential([
        Dense(layer_sizes[0], activation='relu', input_shape=(input_dim,)),
        Dense(layer_sizes[1], activation='relu'),
        Dense(layer_sizes[2], activation='relu'),
        Dense(output_dim, activation='softmax')
    ])
    model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
    return model
```

```
def run_training_without_pretraining(X_train, y_train, X_test, y_test, input_dim, output_dim, layer_sizes):
    """
    Эксперимент 1: Обучение DNN со случайной инициализацией весов.
    """
    print("\n--- 1. Обучение без предобучения ---")
    model = create_dnn_classifier(input_dim, output_dim, layer_sizes)

    print("[Без] Обучение модели (50 эпох)...")
    model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0)

    y_pred = np.argmax(model.predict(X_test), axis=1)
    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')

    print("[Без] Обучение завершено.")
    return acc, f1

def run_training_with_pretraining(X_train, y_train, X_test, y_test, input_dim, output_dim, layer_sizes):
    """
    Эксперимент 2: Обучение DNN с предварительной настройкой весов с помощью автоэнкодеров.
    """
    print("\n--- 2. Обучение с предобучением ---")

    # --- Шаг А: Предобучение слоев ---
    print("[С предоб.] Предобучение автоэнкодерами...")
    encoders = []
    current_data = X_train
    temp_input_dim = input_dim

    for i, size in enumerate(layer_sizes):
        # 1. Создаем автоэнкодер для текущего слоя
        input_layer = Input(shape=(temp_input_dim,))
        encoded_layer = Dense(size, activation='relu')(input_layer)
        decoded_layer = Dense(temp_input_dim, activation='linear')(encoded_layer)

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

        # 2. Обучаем его
        autoencoder.fit(current_data, current_data, epochs=30, batch_size=64, verbose=0)

        # 3. Сохраняем кодирующую часть
        encoder = Model(input_layer, encoded_layer)
        encoders.append(encoder)

        # 4. Прогоняем данные через обученный кодировщик чтобы подготовить их для следующего слоя
        current_data = encoder.predict(current_data)
        temp_input_dim = size

    print("[С предоб.] Предобучение завершено.")

    # --- Шаг Б: Дообучение ---
    model = create_dnn_classifier(input_dim, output_dim, layer_sizes)

    # 1. Загружаем предобученные веса в итоговую модель
    for i, encoder in enumerate(encoders):
        model.layers[i].set_weights(encoder.layers[1].get_weights())
```

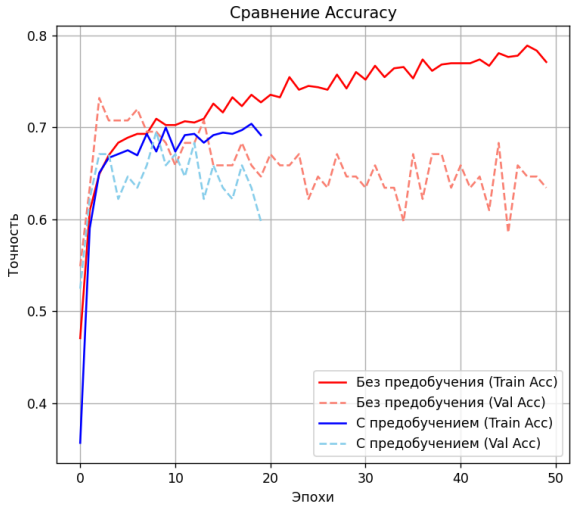
```
    print("[С предоб.] Дообучение модели (20 эпох)...")
    model.fit(X_train, y_train, epochs=20, batch_size=32, verbose=0)

    y_pred = np.argmax(model.predict(X_test), axis=1)
    acc = accuracy_score(y_test, y_pred)
    f1 = f1_score(y_test, y_pred, average='weighted')

    print("[С предоб.] Дообучение завершено.")
    return acc, f1

# --- Этап 3: Полный цикл эксперимента ---

def run_full_experiment(dataset_name, X, y):
    """Запускает оба вида обучения для одного датасета и выводит результаты."""
    print(f"\n{'*70}\nЭксперимент для датасета: {dataset_name}\n{'*70}")

    if X is None or y is None:
        return None

    # Разделение данных 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, stratify=y)

    # Стандартизация признаков
    scaler = StandardScaler()
    X_train_scaled = scaler.fit_transform(X_train)
    X_test_scaled = scaler.transform(X_test)

    # Определение архитектуры сети
    input_dim = X_train_scaled.shape[1]
    output_dim = len(np.unique(y))
    layer_sizes = [64, 32, 16] # У DNN будет иметь 3 скрытых слоя

    # Запуск экспериментов
    acc_no_pre, f1_no_pre = run_training_without_pretraining(X_train_scaled, y_train, X_test_scaled, y_test,
input_dim, output_dim, layer_sizes)
    acc_with_pre, f1_with_pre = run_training_with_pretraining(X_train_scaled, y_train, X_test_scaled, y_test,
input_dim, output_dim, layer_sizes)

    # --- Вывод результатов ---
    print("\n--- Результаты сравнения ---")
    print(f"Без предобучения → Acc: {acc_no_pre:.4f}, F1: {f1_no_pre:.4f}")
    print(f"С предобучением  → Acc: {acc_with_pre:.4f}, F1: {f1_with_pre:.4f}")

    acc_improvement = acc_with_pre - acc_no_pre
    f1_improvement = f1_with_pre - f1_no_pre

    print(f"Улучшение: Acc {acc_improvement:+.4f}, F1 {f1_improvement:+.4f}")

    return {
        "Без Acc": acc_no_pre, "С Acc": acc_with_pre, "дельтаAcc": acc_improvement,
        "Без F1": f1_no_pre, "С F1": f1_with_pre, "дельтаF1": f1_improvement
    }

# --- Этап 4: Запуск всех экспериментов и итоговая таблица ---
if __name__ == '__main__':
    all_results = {}

    # Эксперимент 1
```

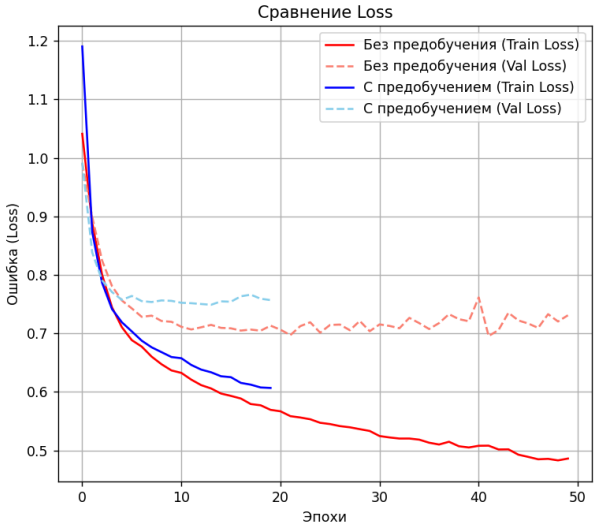
```
X_maternal, y_maternal = load_maternal_health_data(MATERNAL_HEALTH_PATH)
if X_maternal is not None:
    all_results['Maternal Health'] = run_full_experiment('Maternal Health', X_maternal, y_maternal)

# Эксперимент 2
X_rice, y_rice = load_rice_data(RICE_DATA_PATH)
if X_rice is not None:
    all_results['Rice'] = run_full_experiment('Rice', X_rice, y_rice)

# --- Итоговая таблица ---
if all_results:
    final_df = pd.DataFrame.from_dict(all_results, orient='index')
    print(f"\n\n{'='*70}\nИтоговая таблица\n{'='*70}")
    print(final_df.to_string())
```

Вывод:

Загрузка датасета 'Maternal Health Risk'...
Признаков: 6, Классов: 3, Объектов: 1014

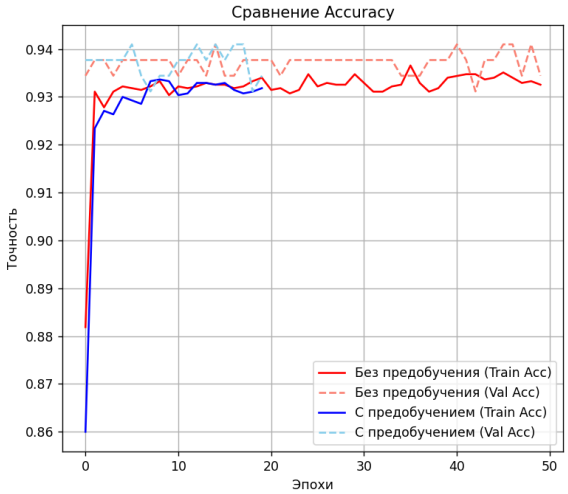
==========
Эксперимент для датасета: Maternal Health
==========
--- Результаты сравнения ---
Без предобучения → Acc: 0.6897, F1: 0.68639
С предобучением → Acc: 0.7192, F1: 0.7032
Улучшение: Acc +0.0296, F1 +0.0169

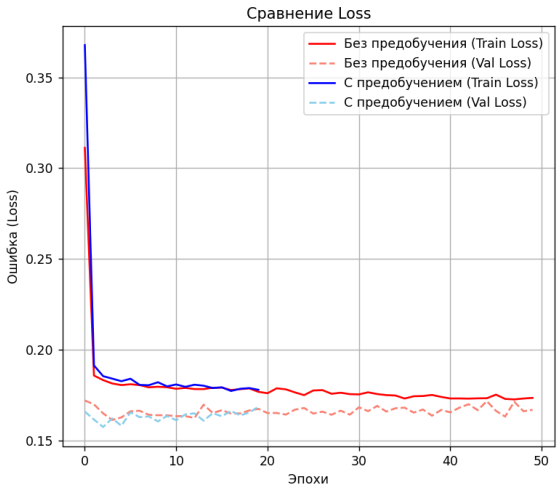




Загрузка датасета 'Rice (Cammeo and Osmancik)'...
Признаков: 7, Классов: 2, Объектов: 3810

==========
Эксперимент для датасета: Rice
==========
--- Результаты сравнения ---
Без предобучения → Acc: 0.9121, F1: 0.9117
С предобучением → Acc: 0.9173, F1: 0.9172
Улучшение: Acc +0.0052, F1 +0.0055





==========

Итоговая таблица

==========

	Без Acc	С Acc	ΔAcc	Без F1	С F1	ΔF1
Maternal Health	0.689655	0.719212	0.029557	0.686336	0.703213	0.016877
Rice	0.912073	0.917323	0.005249	0.911737	0.917203	0.005466

1. Датасет Maternal Health Risk Data Set
Результат: дельтаAcc +2.96%, дельтаF1 +1.69%.
На этом датасете предобучение дало заметный и однозначно положительный эффект.
Этот датасет относительно маленький (1014 объектов) и, судя по не самой высокой точности (~70%), классы не так-то просто разделить. В таких условиях нейросеть, стартующая со случайных весов, может легко заблудиться и попасть в не самое лучшее решение.
2. Датасет Rice_Cammeo_Osmancik
Результат: дельтаAcc +0.52%, дельтаF1 +0.55%.
На этом датасете предобучение тоже дало положительный эффект, но он очень маленький, почти незначительный.
Этот датасет, наоборот, довольно большой (3810 объектов), а классы в нем, как мы видели в ЛР2, очень хорошо разделимы. Задача настолько "простая" для нейросети, что даже подход "с нуля" почти сразу находит отличное решение (точность > 91%). Улучшить и без того хороший результат очень сложно. Предобучение дает лишь крошечное преимущество.

Вывод: Я научился осуществлять предобучение нейронных сетей с помощью автоэнкодерного подхода.