Министерство образования Республики Беларусь
Учреждение образования
"Брестский государственный технический университет"
Кафедра ИИТ

Лабораторная работа №4

По дисциплине "Интеллектуальный анализ данных"

Тема: «Предобучение нейронных сетей с использованием RBM»

Выполнил:

Студент 4 курса

Группы ИИ-24

Бузель С.Д.

Проверил:

Андренко К. В.

Брест 2025

Цель: научиться осуществлять предобучение нейронных сетей с помощью RBM.

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
import matplotlib.pyplot as plt
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
from sklearn.neural_network import BernoulliRBM   # Новая библиотека

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']
        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


def plot_training_history_v2(history_no_pre, history_ae, history_rbm, dataset_name):
    """
    Рисует сравнительные графики обучения для TPEX моделей.
    """
    fig, (ax1, ax2) = plt.subplots(1, 2, figsize=(18, 7))

    # --- График 1: Сравнение Accuracy (Точности) на валидации ---
    ax1.plot(history_no_pre.history['val_accuracy'], label='Без предобучения', color='red', linestyle='--')
    ax1.plot(history_ae.history['val_accuracy'], label='С предоб. (Автоэнкодер)', color='blue', linestyle='--')
    ax1.plot(history_rbm.history['val_accuracy'], label='С предоб. (RBM)', color='green', linestyle='--')
    ax1.set_title('Сравнение Validation Accuracy')
    ax1.set_xlabel('Эпохи')
    ax1.set_ylabel('Точность (Validation)')
    ax1.legend()
    ax1.grid(True)

    # --- График 2: Сравнение Loss (Ошибки) на валидации ---
    ax2.plot(history_no_pre.history['val_loss'], label='Без предобучения', color='red', linestyle='--')
```

```
    ax2.plot(history_ae.history['val_loss'], label='С предоб. (Автоэнкодер)', color='blue', linestyle='--')
    ax2.plot(history_rbm.history['val_loss'], label='С предоб. (RBM)', color='green', linestyle='--')
    ax2.set_title('Сравнение Validation Loss')
    ax2.set_xlabel('Эпохи')
    ax2.set_ylabel('Ошибка (Validation Loss)')
    ax2.legend()
    ax2.grid(True)

    fig.suptitle(f"Графики обучения для датасета: {dataset_name}", fontsize=16)
    plt.show()

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

def create_dnn_classifier(input_dim, output_dim, layer_sizes):
    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):
    print("\n--- 1. Обучение без предобучения ---")
    model = create_dnn_classifier(input_dim, output_dim, layer_sizes)
    print("[Без] Обучение модели (50 эпох)...")

    history = model.fit(X_train, y_train, epochs=50, batch_size=32, verbose=0, validation_split=0.1)

    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, history # Возвращаем историю обучения

def run_training_with_pretraining(X_train, y_train, X_test, y_test, input_dim, output_dim, layer_sizes):
    print("\n--- 2. Обучение с предобучением ---")
    print("[С предоб.] Предобучение автоэнкодерами...")
    encoders = []
    current_data = X_train
    temp_input_dim = input_dim

    for i, size in enumerate(layer_sizes):
        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')
        autoencoder.fit(current_data, current_data, epochs=30, batch_size=64, verbose=0)
        encoder = Model(input_layer, encoded_layer)
        encoders.append(encoder)
        current_data = encoder.predict(current_data)
        temp_input_dim = size

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

```
    model = create_dnn_classifier(input_dim, output_dim, layer_sizes)
    for i, encoder in enumerate(encoders):
        model.layers[i].set_weights(encoder.layers[1].get_weights())

    print("[С предоб.] Дообучение модели (20 эпох)...")
    history = model.fit(X_train, y_train, epochs=20, batch_size=32, verbose=0, validation_split=0.1)

    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, history # Возвращаем историю обучения

def run_training_with_rbm_pretraining(X_train, y_train, X_test, y_test, input_dim, output_dim, layer_sizes):
    """
    Эксперимент 3: Обучение DNN с предобучением с помощью Машин Больцмана (RBM).
    """
    print("\n--- 3. Обучение с предобучением RBM ---")

    # --- Шаг А: Предобучение слоев с помощью RBM ---
    print("[С RBM] Предобучение слоев...")

    # Создаём копию данных и нормализуем их для RBM.
    # При этом для основной сети будем использовать исходные стандартизированные данные.
    from sklearn.preprocessing import MinMaxScaler
    scaler_rbm = MinMaxScaler()
    X_train_rbm_scaled = scaler_rbm.fit_transform(X_train)

    pretrained_weights = []
    current_data = X_train_rbm_scaled

    for i, size in enumerate(layer_sizes):
        print(f"[С RBM] Обучение RBM для слоя {i+1} ({size} нейронов)...")
        # Создаем и обучаем RBM
        rbm = BernoulliRBM(n_components=size, n_iter=20, learning_rate=0.01, verbose=0, random_state=42)
        rbm.fit(current_data)

        # Извлекаем веса и смещения (bias) из обученной RBM.
        # У RBM они будут называться "components_" (веса) и "intercept_hidden_" (смещения).
        weights = rbm.components_.T
        biases = rbm.intercept_hidden_
        pretrained_weights.append((weights, biases))

        # Преобразуем данные для подачи на вход следующей RBM
        current_data = rbm.transform(current_data)

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

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

    # Загружаем извлеченные веса в нашу итоговую модель
    for i, (weights, biases) in enumerate(pretrained_weights):
        model.layers[i].set_weights([weights, biases])

    print("[С RBM] Дообучение модели (20 эпох)...")
    # Дообучаем на исходных стандартизированных данных, а не на [0,1]
```

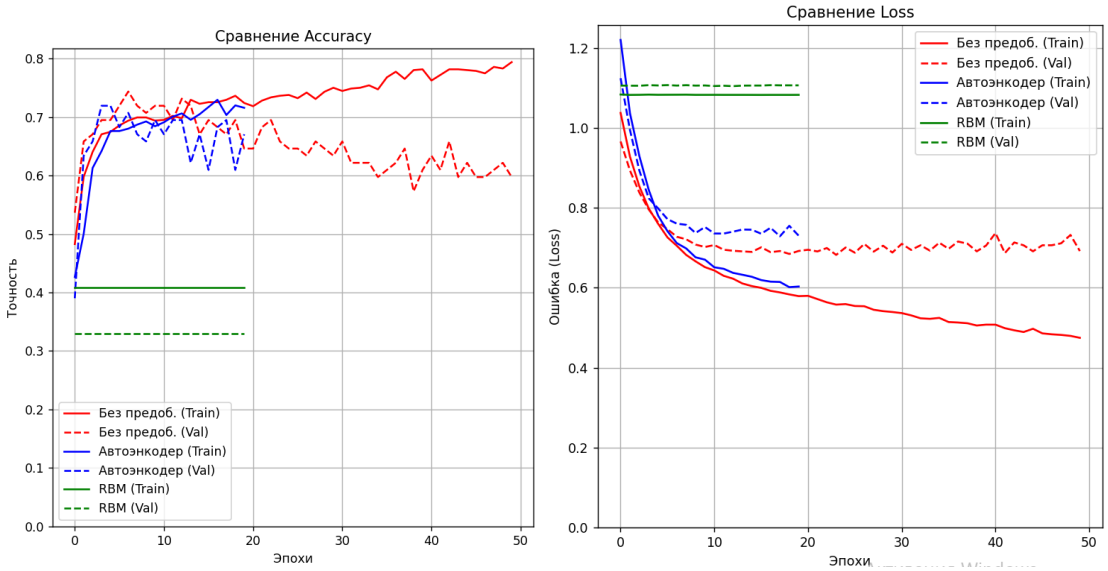
```
    history = model.fit(X_train, y_train, epochs=20, batch_size=32, verbose=0, validation_split=0.1)

    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 RBM] Дообучение завершено.")
    return acc, f1, history

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

def run_full_experiment(dataset_name, X, y):
    """Запускает ВСЕ ТРИ 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 и стандартизация
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
    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]

    # --- Запуск всех трех экспериментов ---
    # Эксперимент 1 (из ЛР3)
    acc_no_pre, f1_no_pre, history_no_pre = run_training_without_pretraining(
        X_train_scaled, y_train, X_test_scaled, y_test, input_dim, output_dim, layer_sizes)

    # Эксперимент 2 (из ЛР3)
    acc_with_ae, f1_with_ae, history_with_ae = run_training_with_pretraining(
        X_train_scaled, y_train, X_test_scaled, y_test, input_dim, output_dim, layer_sizes)

    # Эксперимент 3 (для ЛР4)
    acc_with_rbm, f1_with_rbm, history_with_rbm = run_training_with_rbm_pretraining(
        X_train_scaled, y_train, X_test_scaled, y_test, input_dim, output_dim, layer_sizes)

    # --- Вывод результатов сравнения ---
    print("\n--- Результаты сравнения ---")
    print(f"Без предобучения    -> Acc: {acc_no_pre:.4f}, F1: {f1_no_pre:.4f}")
    print(f"С предоб. (Автоэнкодер) -> Acc: {acc_with_ae:.4f}, F1: {f1_with_ae:.4f}")
    print(f"С предоб. (RBM)       -> Acc: {acc_with_rbm:.4f}, F1: {f1_with_rbm:.4f}")

    # --- Вызов функции для отрисовки графиков ---
    # Передаем в нее все три истории обучения
    plot_training_history_v2(history_no_pre, history_with_ae, history_with_rbm, dataset_name)

    # --- Возвращаем словарь со всеми результатами ---
    return {
        "Без предоб. (Acc)": acc_no_pre, "Автоэнкодер (Acc)": acc_with_ae, "RBM (Acc)": acc_with_rbm,
        "Без предоб. (F1)": f1_no_pre, "Автоэнкодер (F1)": f1_with_ae, "RBM (F1)": f1_with_rbm
    }
```

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

    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)

    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)

    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:

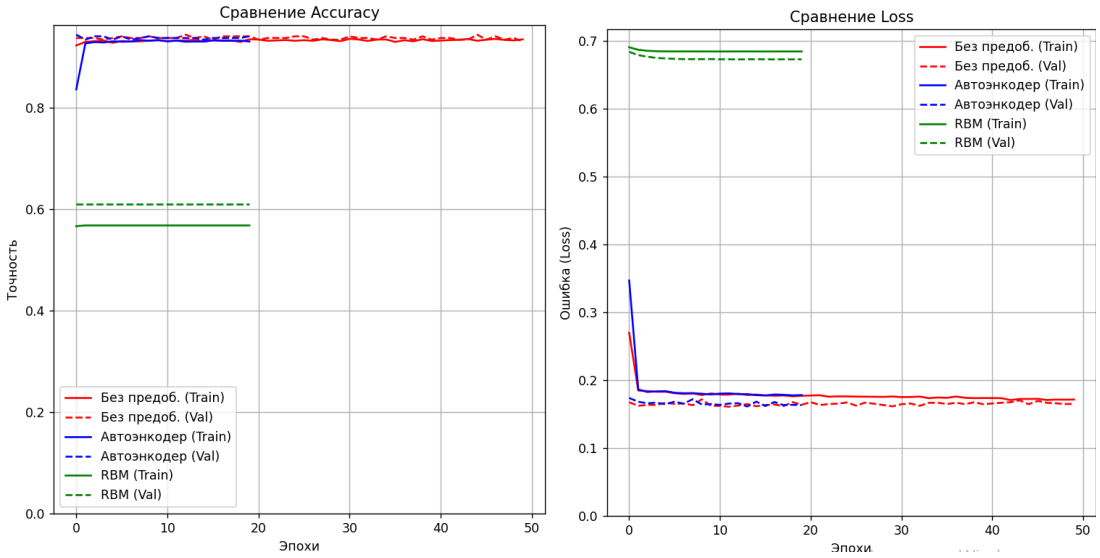
Загрузка датасета 'Maternal Health Risk'...
Признаков: 6, Классов: 3, Объектов: 1014

==========
Эксперимент для датасета: Maternal Health
==========
--- Результаты сравнения ---
Без предобучения -> Acc: 0.6946, F1: 0.6819
С предоб. (Автоэнкодер) -> Acc: 0.7143, F1: 0.7001
С предоб. (RBM) -> Acc: 0.3990, F1: 0.2276



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

==========
Эксперимент для датасета: Rice
==========
--- Результаты сравнения ---
Без предобучения -> Acc: 0.9134, F1: 0.9133
С предоб. (Автоэнкодер) -> Acc: 0.9186, F1: 0.9185
С предоб. (RBM) -> Acc: 0.5722, F1: 0.4165



==========

Итоговая таблица

==========

	Без предоб. (Acc)	Автоэнкодер (Acc)	RBM (Acc)	Без предоб. (F1)	Автоэнкодер (F1)	RBM (F1)
Maternal Health	0.694581	0.714286	0.399015	0.681898	0.700134	0.227607
Rice	0.913386	0.918635	0.572178	0.913280	0.918463	0.416477

Обучение без предобучения и с использованием автоэнкодеров продемонстрировали высокую и сопоставимую точность (около 91% на датасете Rice). В то же время, предобучение с помощью Ограниченных Машин Больцмана (RBM) показало значительно худший результат (57% на том же датасете).

Такой низкий результат RBM можно объяснить несколькими факторами. Во-первых, использовалась модель BernoulliRBM, которая предназначена для бинарных данных, в то время как признаки в обоих датасетах являются непрерывными. Это создает фундаментальное несоответствие между моделью и данными. Во-вторых, RBM являются устаревшим и крайне чувствительным к гиперпараметрам подходом, требующим тщательного подбора для достижения приемлемого результата.

Для модели RBM можно увидеть огромный разрыв между сплошной и штриховой линией. Это визуальное доказательство того, что модель RBM идеально "вызубрила" обучающие данные, однако реального успеха на валидационных данных не показала.

Таким образом, для данных задач классификации современный подход предобучения с помощью автоэнкодеров оказался более эффективным и стабильным, чем исторический метод на основе RBM.

Вывод: Я научился осуществлять предобучение нейронных сетей с помощью RBM.