Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ульяновский государственный технический университет» Кафедра «Вычислительная техника»

Дисциплина «Системы искусственного интеллекта»

Лабораторная работа №4

«Основы нейронных сетей» Вариант №15

Выполнил студент группы УПАСбд-41:

Пяткин.И.А Проверил: Хайруллин И.Д.

Ульяновск, 2025

# Задание:

1. Написать программу, которая разделяет исходную выборку на обучаю- щую и тестовую (training set, validation set, test set), если такое разделе- ние не предусмотрено предложенным набором данных.
2. Произвести масштабирование признаков (scaling).
3. С использованием библиотеки sckit-learn обучить 2 модели нейронной сети (Perceptron и MLPClassifier) по обучающей выборке. Перед обуче- нием необходимо осуществить масштабирование признаков.
4. Проверить точность модели по тестовой выборке.
5. Провести эксперименты и определить наилучшие параметры коэффи- циента обучения, параметра регуляризации, функции оптимизации. Данные экспериментов необходимо представить в отчете (графики, ход проведения эксперимента, выводы).

Датасет по варианту: Phishing Websites

# Информация по датасету:

Данный датасет содержит информацию о существующих фишинговых сайтах. Состоит из двух типов данных 0 - нет и 1 — да, содержит в себе 30 признаков и 1 колонку — Result (является фишинговым сайтом или нет).

# Описание реализации:

Первым делом, необходимо читать данные из arff файла.

Необходимо разделить выборку на тренировочную и тестовую(80% и 20%).

**Масштабирование данных:**

Масштабирование функций – важный шаг в моделировании алгоритмов с помощью наборов данных.

Для данной лабораторной работы была использована функция масшта- бирования StandartScaler() - функция, которая масштабирует весь набор дан-

ных, чтобы среднее значение было равно 0, а стандартное отклонение было равно 1.

**Обучение модели по MLPClassifier:**

**Многослойный персептрон (MLP)** — реализует алгоритм многослойного перцептрона (MLP), который обучается с использованием обратного распространения. Содержит скрытые слои.

Преимущества многослойного перцептрона:

* Возможность изучать нелинейные модели.
* Работает с сложными данными датасета, в нашем случае это фишинговые сайты, которые требуют подробного разбора и проверки наших признаков.

В лабораторной работе используется функция MLPClassifier(), в нем необходимо указать такие параметры, как количество скрытых слоев, макси- мальное количество итераций, параметр регуляризации, обучение. Далее обу- чить модель и предсказать данные.

**Обучение модели по Perceptron:**

**Персептрон** - это разновидность линейного классификатора (бинарно- го классификатора), что означает, что его можно использовать для классифи- кации данных, которые можно разделить линейным способом.

В лабораторной работе используется функция Perceptron(), в нем необ- ходимо указать такие параметры, метод регуляризации, максимальное коли- чество итераций, параметр регуляризации. Далее обучить модель и предска- зать данные.

По полученным предсказанием по каждой модели необходимо вывести точность данных моделей с помощью функции accuracy\_score() - вычисляет

точность предсказания между обучающей и тестовой выборкой.

Были проведены эксперименты и выбраны эффективные параметры для многослойного и однослойного персептрона.

# Вывод:

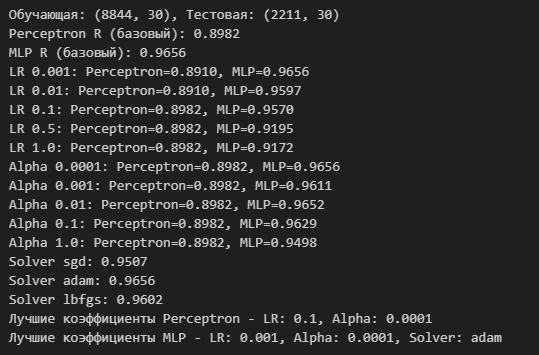


Рисунок 1 - Результаты экспериментов с параметрами

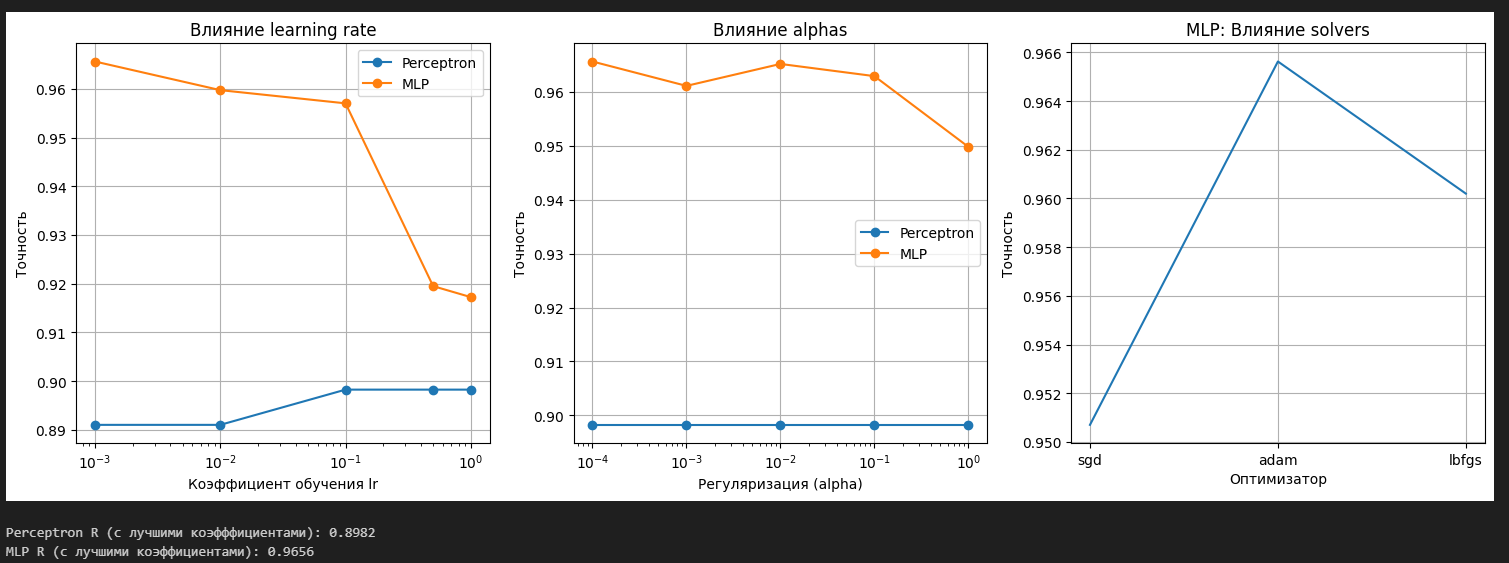


Рисунок 2 - Графики результатов экспериментов

Вывод: лучшие коэффициенты Perceptron - LR: 0.1, Alpha: 0.0001. Лучшие коэффициенты MLP - LR: 0.001, Alpha: 0.0001, Solver: adam. Обучив 2 модели нейронной сети взяв лучшие коэффициенты мы получили результаты: Perceptron R = 0,8982 и MLP R = 0,9656. Получились очень хорошие результаты. Модель MLP показывает более эффективный результат по сравнению c Perceptron, т.к. для нашего датасета используются сложные признаки по проверке фишинговых сайтов.

# Приложение:

import numpy as np

import pandas as pd

import matplotlib.pyplot as plt

from scipy.io import arff

from sklearn.model\_selection import train\_test\_split

from sklearn.preprocessing import StandardScaler

from sklearn.linear\_model import Perceptron

from sklearn.neural\_network import MLPClassifier

from sklearn.metrics import accuracy\_score

data, meta = arff.loadarff('Training Dataset.arff')

df = pd.DataFrame(data).astype(float)

X = df.drop('Result', axis=1)

y = df['Result']

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=1, stratify=y)

scaler = StandardScaler()

X\_train\_scaled = scaler.fit\_transform(X\_train)

X\_test\_scaled = scaler.transform(X\_test)

print(f"Обучающая: {X\_train.shape}, Тестовая: {X\_test.shape}")

perceptron\_base = Perceptron(random\_state=1)

perceptron\_base.fit(X\_train\_scaled, y\_train)

perceptron\_base\_acc = accuracy\_score(y\_test, perceptron\_base.predict(X\_test\_scaled))

mlp\_base = MLPClassifier(random\_state=1, max\_iter=1000)

mlp\_base.fit(X\_train\_scaled, y\_train)

mlp\_base\_acc = accuracy\_score(y\_test, mlp\_base.predict(X\_test\_scaled))

print(f"Perceptron R (базовый): {perceptron\_base\_acc:.4f}")

print(f"MLP R (базовый): {mlp\_base\_acc:.4f}")

learning\_rates = [0.001, 0.01, 0.1, 0.5, 1.0]

alphas = [0.0001, 0.001, 0.01, 0.1, 1.0]

solvers = ['sgd', 'adam', 'lbfgs']

perceptron\_lr\_accuracies = []

mlp\_lr\_accuracies = []

for lr in learning\_rates:

    perceptron = Perceptron(eta0=lr, max\_iter=1000, random\_state=1)

    perceptron.fit(X\_train\_scaled, y\_train)

    perceptron\_acc = accuracy\_score(y\_test, perceptron.predict(X\_test\_scaled))

    perceptron\_lr\_accuracies.append(perceptron\_acc)

    mlp = MLPClassifier(learning\_rate\_init=lr, max\_iter=1000, random\_state=1)

    mlp.fit(X\_train\_scaled, y\_train)

    mlp\_acc = accuracy\_score(y\_test, mlp.predict(X\_test\_scaled))

    mlp\_lr\_accuracies.append(mlp\_acc)

    print(f"LR {lr}: Perceptron={perceptron\_acc:.4f}, MLP={mlp\_acc:.4f}")

perceptron\_alpha\_accuracies = []

mlp\_alpha\_accuracies = []

for alpha in alphas:

    perceptron = Perceptron(alpha=alpha, max\_iter=1000, random\_state=1)

    perceptron.fit(X\_train\_scaled, y\_train)

    perceptron\_acc = accuracy\_score(y\_test, perceptron.predict(X\_test\_scaled))

    perceptron\_alpha\_accuracies.append(perceptron\_acc)

    mlp = MLPClassifier(alpha=alpha, max\_iter=1000, random\_state=1)

    mlp.fit(X\_train\_scaled, y\_train)

    mlp\_acc = accuracy\_score(y\_test, mlp.predict(X\_test\_scaled))

    mlp\_alpha\_accuracies.append(mlp\_acc)

    print(f"Alpha {alpha}: Perceptron={perceptron\_acc:.4f}, MLP={mlp\_acc:.4f}")

mlp\_solver\_accuracies = []

for solver in solvers:

    mlp = MLPClassifier(solver=solver, max\_iter=1000, random\_state=1)

    mlp.fit(X\_train\_scaled, y\_train)

    accuracy = accuracy\_score(y\_test, mlp.predict(X\_test\_scaled))

    mlp\_solver\_accuracies.append(accuracy)

    print(f"Solver {solver}: {accuracy:.4f}")

best\_lr\_perceptron = learning\_rates[np.argmax(perceptron\_lr\_accuracies)]

best\_alpha\_perceptron = alphas[np.argmax(perceptron\_alpha\_accuracies)]

best\_lr\_mlp = learning\_rates[np.argmax(mlp\_lr\_accuracies)]

best\_alpha\_mlp = alphas[np.argmax(mlp\_alpha\_accuracies)]

best\_solver\_mlp = solvers[np.argmax(mlp\_solver\_accuracies)]

print(f"Лучшие коэффициенты Perceptron - LR: {best\_lr\_perceptron}, Alpha: {best\_alpha\_perceptron}")

print(f"Лучшие коэффициенты MLP - LR: {best\_lr\_mlp}, Alpha: {best\_alpha\_mlp}, Solver: {best\_solver\_mlp}")

best\_perceptron = Perceptron(

    eta0=best\_lr\_perceptron,

    alpha=best\_alpha\_perceptron,

    random\_state=1

)

best\_perceptron.fit(X\_train\_scaled, y\_train)

best\_mlp = MLPClassifier(

    learning\_rate\_init=best\_lr\_mlp,

    alpha=best\_alpha\_mlp,

    solver=best\_solver\_mlp,

    random\_state=1,

    max\_iter=1000

)

best\_mlp.fit(X\_train\_scaled, y\_train)

plt.figure(figsize=(15, 5))

plt.subplot(1, 3, 1)

plt.semilogx(learning\_rates, perceptron\_lr\_accuracies, marker='o', label='Perceptron')

plt.semilogx(learning\_rates, mlp\_lr\_accuracies, marker='o', label='MLP')

plt.xlabel('Коэффициент обучения lr')

plt.ylabel('Точность')

plt.title('Влияние learning rate')

plt.legend()

plt.grid(True)

plt.subplot(1, 3, 2)

plt.semilogx(alphas, perceptron\_alpha\_accuracies, marker='o', label='Perceptron')

plt.semilogx(alphas, mlp\_alpha\_accuracies, marker='o', label='MLP')

plt.xlabel('Регуляризация (alpha)')

plt.ylabel('Точность')

plt.title('Влияние alphas')

plt.legend()

plt.grid(True)

plt.subplot(1, 3, 3)

plt.plot(solvers, mlp\_solver\_accuracies)

plt.xlabel('Оптимизатор')

plt.ylabel('Точность')

plt.title('MLP: Влияние solvers')

plt.grid(True)

plt.tight\_layout()

plt.show()

final\_perceptron\_acc = accuracy\_score(y\_test, best\_perceptron.predict(X\_test\_scaled))

final\_mlp\_acc = accuracy\_score(y\_test, best\_mlp.predict(X\_test\_scaled))

print(f"Perceptron R (с лучшими коэфффициентами): {final\_perceptron\_acc:.4f}")

print(f"MLP R (с лучшими коэффициентами): {final\_mlp\_acc:.4f}")