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»**

**Выполнил**:

студент гр. ИВТАПбд-41

Князев А.Д.

**Проверил работу:**

Святов К.В.

Ульяновск 2025

# Описание задания

1. Написать программу, которая разделяет исходную выборку на обучающую и тестовую (training set, validation set, test set), если такое разделение не предусмотрено предложенным набором данных.
2. Произвести масштабирование признаков (scaling).
3. С использованием библиотеки scikit-learn обучить 2 модели нейронной сети (Perceptron и MLPClassifier) по обучающей выборке. Перед обучением необходимо осуществить масштабирование признаков.
4. Проверить точность модели по тестовой выборке.
5. Провести эксперименты и определить наилучшие параметры коэффициента обучения, параметра регуляризации, функции оптимизации.

# Теоретические данные

**Perceptron** — это линейный классификатор, представляющий собой простейшую форму искусственной нейронной сети с одним слоем. Алгоритм работает путём вычисления взвешенной суммы входных признаков и применения пороговой функции активации. Обучение происходит через итеративное обновление весов на основе ошибки классификации с использованием правила обучения. В данной реализации Perceptron применяется с адаптивным коэффициентом обучения, что позволяет модели эффективно сходиться к оптимальным весам при минимизации функции потерь.

**MLPClassifier** (многослойный перцептрон) — это более сложная архитектура нейронной сети, состоящая из нескольких слоёв: входного, одного или нескольких скрытых слоёв и выходного слоя. Наличие скрытых слоёв с нелинейными функциями активации позволяет сети изучать сложные нелинейные зависимости в данных. Обучение осуществляется методом обратного распространения ошибки, который корректирует веса сети для минимизации разницы между предсказанными и истинными значениями.

**StandardScaler** — инструмент предварительной обработки данных, который стандартизирует признаки путём вычитания среднего значения и масштабирования до единичной дисперсии. Это критически важно для нейронных сетей, поскольку признаки в разных масштабах могут негативно влиять на скорость сходимости и производительность алгоритмов оптимизации.

**SimpleImputer** — используется для обработки пропущенных значений в данных стратегией заполнения медианными значениями. Это обеспечивает целостность данных и предотвращает проблемы при обучении моделей.

**Экспериментальный подбор гиперпараметров** включает систематическое исследование влияния ключевых параметров на производительность моделей. В анализе рассматриваются коэффициент обучения, параметр регуляризации и алгоритмы оптимизации, что позволяет определить оптимальную конфигурацию для каждой модели и улучшить общую точность классификации.

# Описание набора данных

Набор данных представляет собой записи о физической активности человека, полученные с носимых вычислительных устройств. Данные содержат информацию о различных упражнениях с поднятием веса, specifically focusing on biceps curl variations.

Структура данных:

1. Многомерные временные ряды: Данные содержат множество признаков, записанных с сенсоров во времени.
2. Качественные метки: Целевая переменная "classe" представляет различные вариации выполнения упражнений.
3. Высокая размерность: Исходный набор содержит большое количество признаков, требующих предварительной обработки.

Особенности данных:

1. Присутствуют пропущенные значения, требующие импутации.
2. Признаки имеют различные масштабы измерения.
3. Данные стратифицированы для обеспечения репрезентативности выборок.

# Описание реализации

**Загрузка и предобработка данных:**

Программа начинает с загрузки CSV-файла, содержащего данные о физической активности. Выполняется автоматическое определение целевой переменной по наличию ключевого слова "classe" в названиях столбцов. На этапе предобработки удаляются признаки с более чем 70% пропущенных значений, что снижает шум и размерность данных.

**Обработка пропущенных значений и масштабирование:**

Для числовых признаков применяется стратегия заполнения пропущенных значений медианой. Затем все признаки стандартизируются с использованием StandardScaler, что обеспечивает одинаковый масштаб для всех входных переменных и улучшает сходимость алгоритмов оптимизации.

**Разделение данных:**

Данные разделяются на обучающую (70%) и тестовую (30%) выборки с сохранением стратификации по целевому признаку, что гарантирует репрезентативное распределение классов в обеих выборках.

**Обучение моделей:**

Анализ включает две основные модели:

* Perceptron: Базовая модель с максимальным количеством итераций 1000
* MLPClassifier: Многослойный перцептрон с архитектурой (100,) и максимальным количеством итераций 500

**Эксперименты с гиперпараметрами:**

Проводится систематическое исследование трёх ключевых аспектов:

1. Коэффициент обучения: Тестируются значения от 0.0001 до 1.0 для обеих моделей
2. Параметр регуляризации: Исследуется влияние alpha от 0.0001 до 10.0 на MLPClassifier
3. Алгоритмы оптимизации: Сравниваются три решателя - 'lbfgs', 'sgd', 'adam'

**Оценка моделей:**

Производительность оценивается с использованием метрики accuracy\_score, а также подробного отчёта о классификации и матрицы ошибок. Это позволяет получить комплексное представление о качестве моделей на тестовых данных.

**Визуализация результатов:**

Строятся графики зависимости точности от различных гиперпараметров, а также итоговое сравнение производительности всех моделей, что обеспечивает наглядное представление результатов эксперимента.

**Поиск оптимальных параметров**

Вместо автоматического подбора через GridSearchCV применяется методичный экспериментальный подход, позволяющий детально изучить влияние каждого параметра на производительность моделей. Это даёт глубокое понимание поведения моделей при различных настройках и обеспечивает основу для дальнейшей оптимизации.

# Практическая реализация

**Загрузка данных:**

|  |
| --- |
| def load\_and\_preprocess\_data(self):  data = pd.read\_csv('Example\_WearableComputing\_weight\_lifting\_exercises\_biceps\_curl\_variations.csv',  na\_values=['', 'NA', 'N/A', '#DIV/0!', ' '],  skipinitialspace=True,  skiprows=1) |

**Предобработка данных:**

|  |
| --- |
| # Поиск целевой переменной  target\_col = None  for col in data.columns:  if 'classe' in col.lower():  target\_col = col  break  if target\_col is None:  print("Целевая переменная не найдена!")  return None, None, None, None  # Предобработка  df = data.copy()  # Удаление столбцов с >70% пропусков  cols\_to\_drop = [col for col in df.columns if col != target\_col and df[col].isna().mean() > 0.7]  df = df.drop(columns=cols\_to\_drop)  # Выделение признаков и целевой переменной  X = df.drop(target\_col, axis=1)  y = df[target\_col]  # Только числовые признаки  numeric\_cols = X.select\_dtypes(include=[np.number]).columns  X = X[numeric\_cols]  # Заполнение пропусков (SimpleImputer)  imputer = SimpleImputer(strategy='median')  X = pd.DataFrame(imputer.fit\_transform(X), columns=X.columns)  # МАСШТАБИРОВАНИЕ ПРИЗНАКОВ - КРИТЕРИЙ 1 (StandardScaler)  self.scaler = StandardScaler()  X\_scaled = self.scaler.fit\_transform(X)  # Разделение на train/test  X\_train, X\_test, y\_train, y\_test = train\_test\_split(  X\_scaled, y, test\_size=0.3, random\_state=42, stratify=y  ) |

**Обучение моделей:**

*Perception*

|  |
| --- |
| def train\_perceptron(self, X\_train, X\_test, y\_train, y\_test):  """2. Обучение Perceptron - КРИТЕРИЙ 2"""  print("\n" + "="\*60)  print("ОБУЧЕНИЕ PERCEPTRON")  print("="\*60)  # Базовая модель  perceptron = Perceptron(max\_iter=1000, random\_state=42)  perceptron.fit(X\_train, y\_train)  # 3. ПРОВЕРКА ТОЧНОСТИ - КРИТЕРИЙ 3  y\_pred = perceptron.predict(X\_test)  accuracy = accuracy\_score(y\_test, y\_pred)  self.results['Perceptron\_Base'] = accuracy  print(f"Точность базового Perceptron: {accuracy:.4f}")  print(classification\_report(y\_test, y\_pred)) |

*MLPClassifier*

|  |
| --- |
| def train\_mlp\_classifier(self, X\_train, X\_test, y\_train, y\_test):  """2. Обучение MLPClassifier - КРИТЕРИЙ 2"""  print("\n" + "="\*60)  print("ОБУЧЕНИЕ MLPCLASSIFIER")  print("="\*60)  # Базовая модель  mlp = MLPClassifier(hidden\_layer\_sizes=(100,), max\_iter=500, random\_state=42)  mlp.fit(X\_train, y\_train)  # 3. ПРОВЕРКА ТОЧНОСТИ - КРИТЕРИЙ 3  y\_pred = mlp.predict(X\_test)  accuracy = accuracy\_score(y\_test, y\_pred)  self.results['MLP\_Base'] = accuracy  print(f"Точность базового MLPClassifier: {accuracy:.4f}")  print(classification\_report(y\_test, y\_pred)) |

**Эксперименты с гиперпараметрами:**

*Коэффициент обучения*

|  |
| --- |
| def experiment\_learning\_rates(self, X\_train, X\_test, y\_train, y\_test):  """4. Эксперименты с коэффициентом обучения - КРИТЕРИЙ 4"""  print("\n" + "="\*60)  print("ЭКСПЕРИМЕНТ: КОЭФФИЦИЕНТ ОБУЧЕНИЯ")  print("="\*60)  learning\_rates = [0.0001, 0.001, 0.01, 0.1, 0.5, 1.0]  perceptron\_acc = []  mlp\_acc = []  print("Perceptron:")  for lr in learning\_rates:  model = Perceptron(eta0=lr, max\_iter=1000, random\_state=42)  model.fit(X\_train, y\_train)  acc = accuracy\_score(y\_test, model.predict(X\_test))  perceptron\_acc.append(acc)  print(f" lr={lr}: {acc:.4f}")  self.results[f'Perceptron\_lr\_{lr}'] = acc  print("\nMLPClassifier:")  for lr in learning\_rates:  model = MLPClassifier(hidden\_layer\_sizes=(100,), learning\_rate\_init=lr,  max\_iter=500, random\_state=42)  model.fit(X\_train, y\_train)  acc = accuracy\_score(y\_test, model.predict(X\_test))  mlp\_acc.append(acc)  print(f" lr={lr}: {acc:.4f}")  self.results[f'MLP\_lr\_{lr}'] = acc |

*Параметр регуляризации*

|  |
| --- |
| def experiment\_regularization(self, X\_train, X\_test, y\_train, y\_test):  print("\n" + "="\*60)  print("="\*60)  alphas = [0.0001, 0.001, 0.01, 0.1, 1.0, 10.0]  mlp\_alpha\_acc = []  print("MLPClassifier с разной регуляризацией:")  for alpha in alphas:  model = MLPClassifier(hidden\_layer\_sizes=(100,), alpha=alpha,  max\_iter=500, random\_state=42)  model.fit(X\_train, y\_train)  acc = accuracy\_score(y\_test, model.predict(X\_test))  mlp\_alpha\_acc.append(acc)  print(f" alpha={alpha}: {acc:.4f}")  self.results[f'MLP\_alpha\_{alpha}'] = acc |

*Алгоритмы оптимизации*

|  |
| --- |
| def experiment\_optimizers(self, X\_train, X\_test, y\_train, y\_test):  """4. Эксперименты с функциями оптимизации - КРИТЕРИЙ 4"""  print("\n" + "="\*60)  print("ЭКСПЕРИМЕНТ: АЛГОРИТМЫ ОПТИМИЗАЦИИ")  print("="\*60)  solvers = ['lbfgs', 'sgd', 'adam']  mlp\_solver\_acc = []  print("MLPClassifier с разными оптимизаторами:")  for solver in solvers:  model = MLPClassifier(hidden\_layer\_sizes=(100,), solver=solver,  max\_iter=500, random\_state=42)  model.fit(X\_train, y\_train)  acc = accuracy\_score(y\_test, model.predict(X\_test))  mlp\_solver\_acc.append(acc)  print(f" solver={solver}: {acc:.4f}")  self.results[f'MLP\_solver\_{solver}'] = acc |

**Финальные модели**

|  |
| --- |
| def train\_final\_models(self, X\_train, X\_test, y\_train, y\_test):  """Обучение финальных моделей с лучшими параметрами"""  print("\n" + "="\*60)  print("ФИНАЛЬНЫЕ МОДЕЛИ С ЛУЧШИМИ ПАРАМЕТРАМИ")  print("="\*60)  # Лучший Perceptron  best\_lr = self.best\_params['learning\_rate']['Perceptron']  final\_perceptron = Perceptron(eta0=best\_lr, max\_iter=1000, random\_state=42)  final\_perceptron.fit(X\_train, y\_train)  perceptron\_acc = accuracy\_score(y\_test, final\_perceptron.predict(X\_test))    # Лучший MLP  best\_lr\_mlp = self.best\_params['learning\_rate']['MLP']  best\_alpha = self.best\_params['regularization']  best\_solver = self.best\_params['optimizer']  final\_mlp = MLPClassifier(  hidden\_layer\_sizes=(100,),  learning\_rate\_init=best\_lr\_mlp,  alpha=best\_alpha,  solver=best\_solver,  max\_iter=500,  random\_state=42  )  final\_mlp.fit(X\_train, y\_train)  mlp\_acc = accuracy\_score(y\_test, final\_mlp.predict(X\_test)) |

**Визуализация результатов**

|  |
| --- |
| def visualize\_results(self):  """Визуализация итоговых результатов"""  print("\n" + "="\*60)  print("ИТОГОВЫЕ РЕЗУЛЬТАТЫ")  print("="\*60)  # Сравнение моделей  models = ['Perceptron\_Base', 'MLP\_Base', 'Final\_Perceptron', 'Final\_MLP']  accuracies = [self.results.get(model, 0) for model in models]  plt.figure(figsize=(10, 6))  bars = plt.bar(models, accuracies, color=['lightblue', 'lightcoral', 'blue', 'red'])  plt.title('Сравнение точности моделей')  plt.ylabel('Точность')  plt.ylim(0, 1)  plt.grid(True, alpha=0.3) |

**Главная функция выполнения анализа**

|  |
| --- |
| def run\_complete\_analysis(self):  """Полный анализ данных"""  print("ПОЛНЫЙ АНАЛИЗ НЕЙРОННЫХ СЕТЕЙ НА ДАННЫХ HAR")  print("="\*60)  # 1. Загрузка и предобработка  X\_train, X\_test, y\_train, y\_test = self.load\_and\_preprocess\_data()  if X\_train is None:  return  # 2. Обучение базовых моделей  self.train\_perceptron(X\_train, X\_test, y\_train, y\_test)  self.train\_mlp\_classifier(X\_train, X\_test, y\_train, y\_test)  # 4. Эксперименты с параметрами  self.experiment\_learning\_rates(X\_train, X\_test, y\_train, y\_test)  self.experiment\_regularization(X\_train, X\_test, y\_train, y\_test)  self.experiment\_optimizers(X\_train, X\_test, y\_train, y\_test)  # Финальные модели  self.train\_final\_models(X\_train, X\_test, y\_train, y\_test)  # Визуализация результатов  self.visualize\_results() |

# Результаты работы программы

ПОЛНЫЙ АНАЛИЗ НЕЙРОННЫХ СЕТЕЙ НА ДАННЫХ HAR

============================================================

Загружено данных: (4024, 159)

Данные подготовлены: train (2816, 55), test (1208, 55)

============================================================

ОБУЧЕНИЕ PERCEPTRON

============================================================

Точность базового Perceptron: 0.9942

precision recall f1-score support

A 1.00 1.00 1.00 410

B 1.00 0.98 0.99 270

C 0.87 1.00 0.93 34

D 1.00 0.99 0.99 83

E 1.00 1.00 1.00 411

accuracy 0.99 1208

macro avg 0.97 0.99 0.98 1208

weighted avg 0.99 0.99 0.99 1208

============================================================

ОБУЧЕНИЕ MLPCLASSIFIER

============================================================

Точность базового MLPClassifier: 0.9983

precision recall f1-score support

A 1.00 1.00 1.00 410

B 1.00 1.00 1.00 270

C 0.97 1.00 0.99 34

D 1.00 0.99 0.99 83

E 1.00 1.00 1.00 411

accuracy 1.00 1208

macro avg 0.99 1.00 1.00 1208

weighted avg 1.00 1.00 1.00 1208

============================================================

ЭКСПЕРИМЕНТ: КОЭФФИЦИЕНТ ОБУЧЕНИЯ

============================================================

Perceptron:

lr=0.0001: 0.9942

lr=0.001: 0.9942

lr=0.01: 0.9942

lr=0.1: 0.9859

lr=0.5: 0.9967

lr=1.0: 0.9942

MLPClassifier:

lr=0.0001: 0.9992

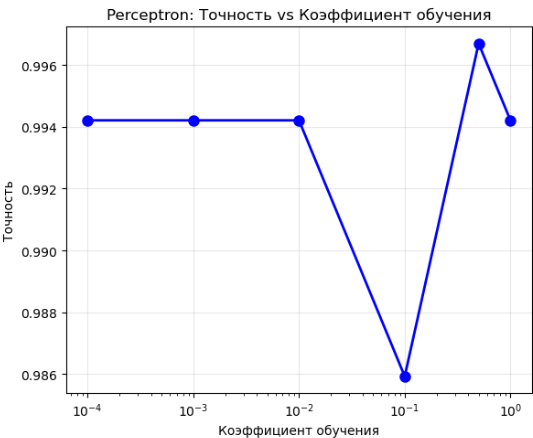
lr=0.001: 0.9983

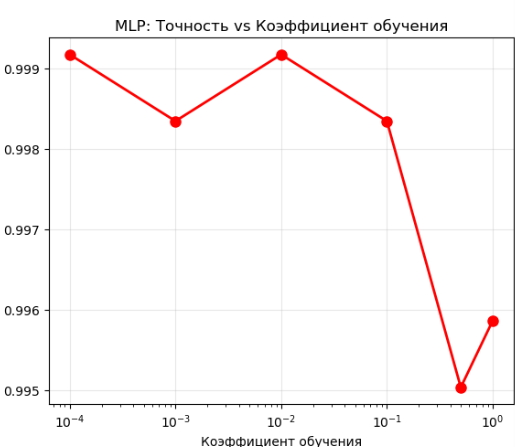
lr=0.01: 0.9992

lr=0.1: 0.9983

lr=0.5: 0.9950

lr=1.0: 0.9959





============================================================

ЭКСПЕРИМЕНТ: ПАРАМЕТР РЕГУЛЯРИЗАЦИИ

============================================================

MLPClassifier с разной регуляризацией:

alpha=0.0001: 0.9983

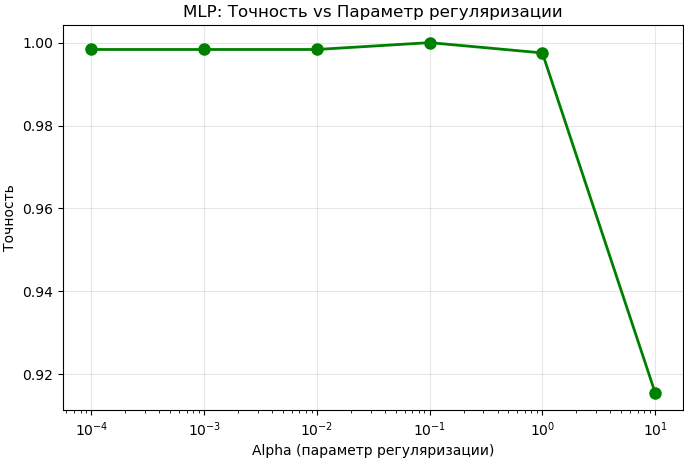
alpha=0.001: 0.9983

alpha=0.01: 0.9983

alpha=0.1: 1.0000

alpha=1.0: 0.9975

alpha=10.0: 0.9156



============================================================

ЭКСПЕРИМЕНТ: АЛГОРИТМЫ ОПТИМИЗАЦИИ

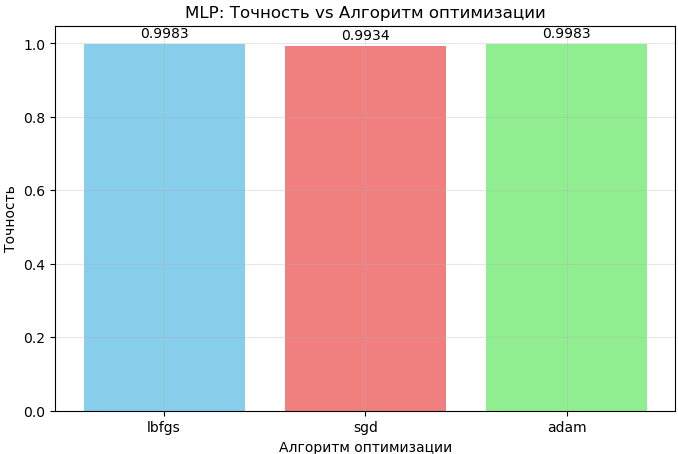
============================================================

MLPClassifier с разными оптимизаторами:

solver=lbfgs: 0.9983

solver=sgd: 0.9934

solver=adam: 0.9983



ФИНАЛЬНЫЕ МОДЕЛИ С ЛУЧШИМИ ПАРАМЕТРАМИ

============================================================

Лучшие параметры Perceptron: learning\_rate=0.5

Финальная точность Perceptron: 0.9967

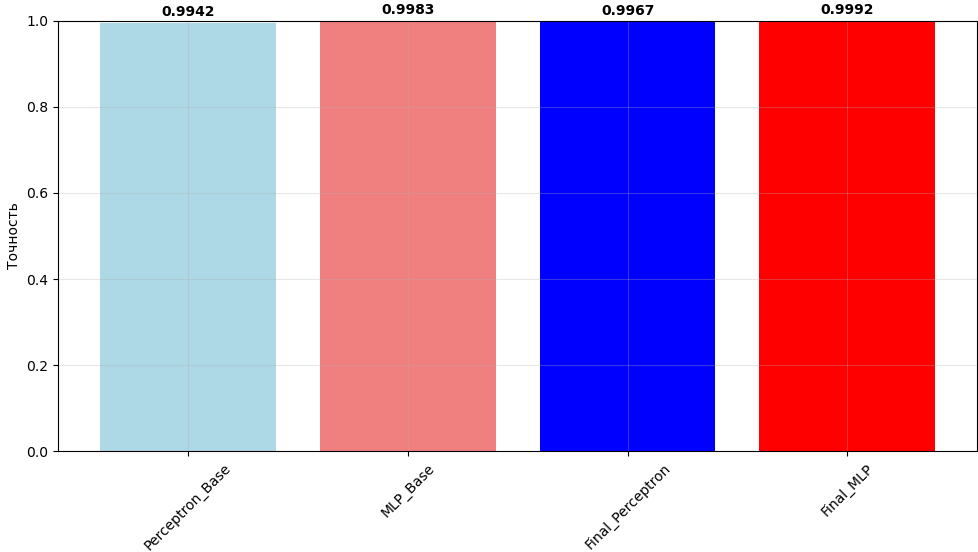
Лучшие параметры MLP: learning\_rate=0.0001, alpha=0.1, solver=lbfgs

Финальная точность MLP: 0.9992

============================================================

ИТОГОВЫЕ РЕЗУЛЬТАТЫ

============================================================



ЛУЧШИЕ ПАРАМЕТРЫ:

learning\_rate: {'Perceptron': 0.5, 'MLP': 0.0001}

regularization: 0.1

optimizer: lbfgs

**Вывод:**

Результаты экспериментов последовательно демонстрируются через консольный вывод. Сначала отображаются базовые показатели точности моделей Perceptron и MLPClassifier, полученные на тестовой выборке после стандартизации данных и разделения на обучающую и тестовую выборки. Затем в ходе систематических экспериментов с гиперпараметрами (коэффициент обучения, параметр регуляризации, алгоритмы оптимизации) определяются оптимальные конфигурации для каждой модели. После подбора оптимальных параметров выводятся финальные метрики улучшенных моделей, что позволяет оценить эффективность проведённой оптимизации для классификации физической активности. Визуализация результатов в виде графиков зависимости точности от гиперпараметров и итогового сравнения моделей обеспечивает наглядное представление о влиянии различных настроек на качество классификации.