МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ АВТОНОМНОЕ ОБРАЗОВАТЕЛЬНОЕ
УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«СЕВЕРО-КАВКАЗСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ»

Методические указания

по выполнению лабораторных работ
по дисциплине «Математические основы искусственного интеллекта»
для студентов
Специальности 10.05.03 Информационная безопасность
автоматизированных систем
Специализация «Анализ безопасности информационных систем»

Ставрополь
2026

Оглавление

Введение

Математические основы искусственного интеллекта - дисциплина, изучающая алгоритмические и математические методы, лежащие в основе современных систем ИИ. Современные методы машинного обучения опираются на аппарат линейной алгебры, математического анализа, теории вероятностей и дискретной математики.

Исторически развитие дисциплины связано с работами Алана Тьюринга (1950), Джона Маккарти (1956), Фрэнка Розенблатта (1957) и других ученых. Современные алгоритмы ИИ используют методы оптимизации, статистического вывода и теорию информации.

Курс лабораторных работ состоит из двенадцати лабораторных работ, списка литературы и оглавления. Предлагаемое учебное пособие предназначено для студентов, обучающихся по специальности: 10.05.03 «Информационная безопасность автоматизированных систем».

Лабораторная работа №1. Основы линейной алгебры для ИИ: операции с матрицами и векторами. Решение систем линейных уравнений.

Цель работы: освоение базовых операций линейной алгебры, применяемых в алгоритмах искусственного интеллекта, включая работу с матрицами и векторами, а также практическое решение систем линейных уравнений различными методами.

Теоретическая часть
Линейная алгебра является математической основой для большинства алгоритмов машинного обучения, компьютерной графики и обработки больших данных. Ключевыми объектами изучения выступают матрицы и векторы, а также операции над ними.

1. Матрицы и векторы
Матрица – это прямоугольная таблица элементов (чисел, символов или функций), организованных в *m* строк и *n* столбцов. Обозначается как $A = [a_{ij}]$, где a_{ij} – элемент на пересечении *i*-й строки и *j*-го столбца. Вектор – частный случай матрицы с одним столбцом (вектор-столбец) или одной строкой (вектор-строка).

Основные операции:

- **Сложение и вычитание матриц** определено только для матриц одинаковой размерности. Результат – матрица $C = A \pm B$, где $c_{ij} = a_{ij} \pm b_{ij}$.
- **Умножение матрицы на скаляр** – каждый элемент умножается на число: $B = k \cdot A$, $b_{ij} = k \cdot a_{ij}$.
- **Умножение матриц** ($A \cdot B$) возможно, если число столбцов A равно числу строк B. Элемент c_{ij} результирующей матрицы вычисляется как скалярное произведение *i*-й строки A и *j*-го столбца B.
- **Транспонирование** – замена строк на столбцы: $A^T = [a_{ji}]$.
- **Определитель** ($det(A)$) – скалярная характеристика квадратной матрицы, используемая для проверки обратимости. Для матрицы 2×2: $det(A) = a_{11} \cdot a_{22} - a_{12} \cdot a_{21}$.
- **Обратная матрица** (A^{-1}) существует, если $det(A) \neq 0$. Вычисляется через алгебраические дополнения или методом Гаусса-Жордана.

2. Системы линейных уравнений (СЛУ)
СЛУ вида $A \cdot X = B$, где A – матрица коэффициентов, X – вектор неизвестных, B – вектор свободных членов, решается следующими методами:

- **Метод Гаусса.** Последовательное исключение переменных путем приведения матрицы A к ступенчатому виду с последующей обратной подстановкой.
- **Метод Крамера.** Применим для квадратных невырожденных систем. Решение: $x_j = det(A_j)/det(A)$, где A_j – матрица A с заменой *j*-го столбца на B.
- **Матричный метод.** Если A обратима, то $X = A^{-1} \cdot B$.

Практическая часть
Задание 1. Операции с матрицами

1. Задайте матрицы A и B размерности 2×2 и 3×3.
2. Выполните операции:
   - $C = A + B$;
   - *D = 3·A*;
   - $E = A \cdot B$;
   - $F = A^T$.
3. Для квадратных матриц вычислите определители и обратные матрицы.

Задание 2. Решение СЛУ

1. Решите систему вручную методом Гаусса:

   {2x+3y=5,4x−y=3.{2*x*+3*y*=5,4*x*−*y*=3.

2. Решите ту же систему методом Крамера и матричным методом.
3. Сравните результаты, оцените погрешности.

Аппаратура и материалы

1. Персональный компьютер с ОС Windows/Linux.
2. Программное обеспечение: Python (с библиотеками NumPy, SciPy) или MATLAB.

Методика выполнения

1. Для ручных вычислений используйте разлинованные листы, проверяя результаты на каждом шаге.
2. В программной части:
   1) Для операций с матрицами примените функции numpy.array, numpy.dot, numpy.linalg.inv.
   2) Для решения СЛУ используйте numpy.linalg.solve.

Пример кода (Python):

```
import numpy as np
```

```
A = np.array([[2, 3], [4, -1]])
B = np.array([5, 3])
X = np.linalg.solve(A, B)
print("Решение СЛУ:", X)
```

Оформление отчета

1. **Титульный лист**
2. **Теоретическая часть**: краткое описание методов (1–2 страницы).
3. **Практическая часть**:
    - Результаты ручных вычислений с промежуточными шагами.
    - Листинг программы и выводы.
4. **Выводы**: анализ точности методов, их преимущества и недостатки.

Вопросы для защиты

1. Какие условия необходимы для умножения матриц?
2. Почему метод Гаусса считается универсальным?
3. Как проверить совместность СЛУ?
4. В чем преимущество матричного метода перед методом Крамера?
5. Приведите примеры использования матриц в нейронных сетях.

Лабораторная работа №2. Методы оптимизации в машинном обучении: градиентный спуск и его модификации. Поиск минимума функции.

Цель работы: изучение и практическое применение методов оптимизации, используемых в машинном обучении, включая классический градиентный спуск и его модификации для поиска минимума функции потерь. Освоение навыков реализации алгоритмов оптимизации на практике.

Теоретическая часть

1. Основы оптимизации в машинном обучении
В задачах машинного обучения часто требуется минимизировать функцию потерь, которая характеризует ошибку модели на обучающих данных. Процесс оптимизации заключается в подборе таких параметров модели, при которых значение функции потерь становится минимальным. Для решения этой задачи применяются различные методы оптимизации, среди которых градиентный спуск занимает центральное место.

2. Градиентный спуск
Градиентный спуск — это итерационный метод оптимизации, позволяющий находить локальный минимум дифференцируемой функции. Основная идея

метода заключается в движении в направлении, противоположном градиенту функции в текущей точке, так как градиент указывает направление наискорейшего роста функции.

Алгоритм градиентного спуска:

1. Инициализировать начальные значения параметров (обычно случайным образом).
2. Вычислить градиент функции потерь по параметрам в текущей точке.
3. Обновить параметры, сделав шаг в направлении, противоположном градиенту:
   $\theta = \theta - \eta \cdot \nabla J(\theta)$,
   где θ — вектор параметров, η — скорость обучения (learning rate), $\nabla J(\theta)$ — градиент функции потерь $J(\theta)$.
4. Повторять шаги 2-3 до достижения критерия остановки (например, заданного числа итераций или малого изменения функции потерь).

3. Модификации градиентного спуска
Классический градиентный спуск имеет несколько недостатков, таких как медленная сходимость и возможность застревания в локальных минимумах. Для устранения этих проблем разработаны различные модификации:

- **Стохастический градиентный спуск (SGD):** на каждой итерации градиент вычисляется не по всем данным, а по одному случайному примеру из обучающей выборки. Это ускоряет обучение и позволяет избежать локальных минимумов.
- **Mini-batch градиентный спуск:** компромисс между классическим и стохастическим методами. Градиент вычисляется по небольшой подвыборке (mini-batch) данных.
- **Метод моментов (Momentum):** добавляет инерцию процессу обновления параметров, что ускоряет сходимость и помогает преодолевать локальные минимумы.
- **AdaGrad, RMSprop, Adam:** адаптивные методы, которые автоматически настраивают скорость обучения для каждого параметра, что повышает эффективность оптимизации.

4. Критерии выбора метода оптимизации
Выбор конкретного метода зависит от характера задачи, размера данных, требуемой точности и вычислительных ресурсов. Важно учитывать:

- Скорость сходимости метода.
- Устойчивость к шуму в данных.
- Способность избегать локальных минимумов.
- Вычислительную сложность.

Практическая часть

Задание 1. Реализация градиентного спуска

1. Задайте функцию потерь $J(\theta) = \theta^2 + 2\theta + 1$.
2. Реализуйте алгоритм градиентного спуска для поиска минимума этой функции.
3. Проведите эксперименты с разными значениями скорости обучения η (0.1, 0.01, 0.001).
4. Постройте график изменения функции потерь от номера итерации для каждого η.
5. Сравните количество итераций, необходимое для достижения заданной точности при разных η.

Задание 2. Сравнение модификаций градиентного спуска

1. Реализуйте SGD, Momentum и Adam для той же функции потерь.
2. Используйте одинаковые начальные условия и скорость обучения для всех методов.
3. Сравните скорость сходимости методов по количеству итераций до достижения минимума с заданной точностью.
4. Проанализируйте поведение методов при наличии шума в данных (добавьте небольшой случайный шум к градиенту).

Аппаратура и материалы

1. Персональный компьютер с ОС Windows/Linux.
2. Программное обеспечение: Python (с библиотеками NumPy, Matplotlib) или MATLAB.

Методика выполнения

1. Для реализации алгоритмов используйте языки программирования Python или MATLAB.
2. Для визуализации результатов применяйте библиотеки построения графиков.
3. Все эксперименты проводите с фиксированным начальным приближением для обеспечения корректности сравнения методов.

Пример кода (Python):

```
import numpy as np
import matplotlib.pyplot as plt
def loss_function(theta):
    return theta**2 + 2*theta + 1
def gradient(theta):
    return 2*theta + 2
def gradient_descent(eta, iterations):
```

```
    theta = 10  # начальное значение
    history = []
    for i in range(iterations):
        grad = gradient(theta)
        theta = theta - eta * grad
        history.append(loss_function(theta))
    return history
# Параметры эксперимента
eta_values = [0.1, 0.01, 0.001]
iterations = 100
# Проведение экспериментов
results = {}
for eta in eta_values:
    results[eta] = gradient_descent(eta, iterations)
# Визуализация результатов
plt.figure(figsize=(10, 6))
for eta, history in results.items():
    plt.plot(history, label=f'η = {eta}')
plt.xlabel('Итерации')
plt.ylabel('Функция потерь')
plt.legend()
plt.grid()
plt.show()
```

Оформление отчета

1. **Титульный лист**
2. **Теоретическая часть**: описание методов оптимизации (1-2 страницы).
3. **Практическая часть**:
    - Результаты экспериментов с графиками.
    - Листинги программ с комментариями.
    - Сравнительный анализ методов.
4. **Выводы**: рекомендации по выбору метода оптимизации в зависимости от условий задачи.

Вопросы для защиты

1. В чем преимущества и недостатки градиентного спуска?
2. Как скорость обучения влияет на сходимость алгоритма?
3. Почему SGD может быть эффективнее классического градиентного спуска?
4. Каким образом метод моментов помогает ускорить сходимость?
5. В каких задачах машинного обучения применяются изученные методы оптимизации?

Лабораторная работа №3. Линейная регрессия и метод наименьших квадратов: построение и анализ линейной модели. Оценка точности.

Цель работы: изучение принципов построения линейных регрессионных моделей методом наименьших квадратов, освоение методик оценки точности моделей и их практического применения для решения задач прогнозирования. Получение навыков реализации алгоритмов линейной регрессии на практике.

Теоретическая часть

1. Основы линейной регрессии
Линейная регрессия представляет собой статистический метод моделирования зависимости между независимой (X) и зависимой (Y) переменными. Модель предполагает, что эта зависимость может быть описана линейным уравнением:
$Y = \beta_0 + \beta_1 X + \varepsilon$,
где β_0 - свободный член (intercept), β_1 - коэффициент регрессии, ε - случайная ошибка.

2. Метод наименьших квадратов (МНК)
МНК - стандартный подход к приближению данных линейной моделью. Основная идея заключается в минимизации суммы квадратов отклонений наблюдаемых значений от значений, предсказанных моделью:
$\Sigma(y_i - \hat{y}_i)^2 \rightarrow \min$,
где y_i - наблюдаемые значения, $\hat{y}_i$ - предсказанные значения.

3. Вычисление коэффициентов регрессии
Для простой линейной регрессии (одна независимая переменная) коэффициенты вычисляются по формулам:
$\beta_1 = \Sigma(x_i - \bar{x})(y_i - \bar{y}) / \Sigma(x_i - \bar{x})^2$
$\beta_0 = \bar{y} - \beta_1 \bar{x}$,
где $\bar{x}$ и $\bar{y}$ - средние значения переменных.

4. Оценка качества модели
Для оценки точности модели используются следующие показатели:

- Коэффициент детерминации R^2: показывает долю дисперсии зависимой переменной, объясняемую моделью ($0 \leq R^2 \leq 1$)
- Среднеквадратическая ошибка (MSE): среднее значение квадратов ошибок
- Средняя абсолютная ошибка (MAE): среднее абсолютных значений ошибок

5. Предположения линейной регрессии
Для корректного применения метода необходимо выполнение следующих условий:

1. Линейность зависимости между переменными
2. Гомоскедастичность ошибок (постоянство дисперсии)
3. Отсутствие мультиколлинеарности
4. Нормальное распределение ошибок
5. Отсутствие автокорреляции ошибок

Практическая часть

Задание 1. Построение модели линейной регрессии

1. Сгенерируйте набор данных с линейной зависимостью:
   - Создайте массив X из 100 значений в диапазоне от 0 до 10
   - Задайте $Y = 2 + 3*X + \varepsilon$, где ε - случайный шум (нормальное распределение, $\mu=0$, $\sigma=1$)

2. Реализуйте расчет коэффициентов регрессии по формулам МНК
3. Постройте график исходных данных и линии регрессии
4. Вычислите предсказанные значения $\hat{y}$ для всех x_i

Задание 2. Оценка точности модели

1. Вычислите следующие показатели качества:
   - Коэффициент детерминации R^2
   - Среднеквадратическую ошибку (MSE)
   - Среднюю абсолютную ошибку (MAE)

2. Проведите анализ остатков:
   - Постройте график остатков ($y_i - \hat{y}_i$) от предсказанных значений
   - Проверьте визуально выполнение предположений регрессии

Задание 3. Сравнение с готовыми реализациями

1. Используйте библиотеку scikit-learn для построения той же модели
2. Сравните полученные коэффициенты с рассчитанными вручную
3. Проверьте совпадение метрик качества

Аппаратура и материалы

1. Персональный компьютер с ОС Windows/Linux
2. Программное обеспечение: Python 3.x с библиотеками NumPy, Matplotlib, scikit-learn

Методика выполнения

1. Для генерации данных используйте numpy.random
2. Реализацию МНК выполните с помощью базовых операций NumPy
3. Для визуализации применяйте matplotlib.pyplot
4. Готовую реализацию возьмите из sklearn.linear_model.LinearRegression

Пример кода (Python):

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error, mean_absolute_error
# Генерация данных
np.random.seed(42)
X = np.linspace(0, 10, 100)
y = 2 + 3 * X + np.random.normal(0, 1, 100)
# Ручная реализация МНК
X_mean, y_mean = np.mean(X), np.mean(y)
beta1 = np.sum((X - X_mean) * (y - y_mean)) / np.sum((X - X_mean)**2)
beta0 = y_mean - beta1 * X_mean
# Предсказания
y_pred = beta0 + beta1 * X
# Оценка качества
r2 = r2_score(y, y_pred)
mse = mean_squared_error(y, y_pred)
mae = mean_absolute_error(y, y_pred)
# Визуализация
plt.figure(figsize=(10, 6))
plt.scatter(X, y, label='Исходные данные')
plt.plot(X, y_pred, color='red', label=f'Линия регрессии: y = {beta0:.2f} + {beta1:
.2f}x')
plt.xlabel('X')
plt.ylabel('Y')
plt.legend()
plt.grid()
plt.show()
# Сравнение с sklearn
model = LinearRegression().fit(X.reshape(-1, 1), y)
print(f"Ручной расчет: beta0 = {beta0:.4f}, beta1 = {beta1:.4f}")
print(f"Sklearn: beta0 = {model.intercept_:.4f}, beta1 = {model.coef_[0]:.4f}")
```

Оформление отчета

1. **Титульный лист**

2. **Теоретическая часть** (1-2 страницы):
    - Описание метода наименьших квадратов
    - Формулы для расчета коэффициентов
    - Методы оценки качества модели

3. **Практическая часть**:
    - Графики исходных данных и линии регрессии
    - Таблица с расчетными коэффициентами
    - Значения метрик качества (R^2, MSE, MAE)
    - График остатков с анализом
    - Листинг программы с комментариями

4. **Выводы**:
    - Анализ точности полученной модели
    - Сравнение ручной реализации и библиотечной
    - Оценка выполнения предположений линейной регрессии

Вопросы для защиты

1. В чем суть метода наименьших квадратов?
2. Как интерпретируется коэффициент детерминации R^2?
3. Какие предположения лежат в основе линейной регрессии?
4. Почему важно анализировать остатки регрессии?
5. В чем преимущества и недостатки линейной регрессии?

Лабораторная работа №4. Логистическая регрессия для классификации: сигмоидная функция, кросс-энтропия, применение в бинарной классификации

Цель работы: изучение принципов работы логистической регрессии как метода бинарной классификации, освоение математического аппарата сигмоидной функции и функции потерь кросс-энтропии, практическая реализация алгоритма классификации на реальных данных.

Теоретическая часть

1. Основы логистической регрессии
Логистическая регрессия - это статистический метод классификации, используемый для предсказания вероятности принадлежности объекта к определенному классу. В отличие от линейной регрессии, которая предсказывает непрерывные значения, логистическая регрессия выдаёт вероятность в диапазоне от 0 до 1.

2. Сигмоидная функция
Ключевым элементом логистической регрессии является сигмоидная функция (логистическая функция), которая преобразует линейную комбинацию признаков в вероятность:
$\sigma(z) = 1 / (1 + e^{(-z)})$
где $z = w_0 + w_1x_1 + ... + w_nx_n$ - линейная комбинация признаков с весами.

Свойства сигмоидной функции:

- Принимает значения в интервале (0, 1)
- Имеет S-образную форму
- $\sigma(0) = 0.5$
- Асимптотически приближается к 0 и 1 при $z \to -\infty$ и $z \to +\infty$ соответственно

3. Функция потерь (кросс-энтропия)
Для обучения модели используется функция потерь - бинарная кросс-энтропия:
$L(y, \hat{y}) = -[y \cdot \log(\hat{y}) + (1-y) \cdot \log(1-\hat{y})]$
где y - истинное значение (0 или 1), $\hat{y}$ - предсказанная вероятность.

Для всего набора данных из m примеров:
$J(w) = -(1/m) \Sigma [y^i \cdot \log(\hat{y}^i) + (1-y^i) \cdot \log(1-\hat{y}^i)]$

4. Оптимизация параметров
Минимизация функции потерь выполняется методами градиентного спуска.
Градиент функции потерь по параметрам w:
$\partial J/\partial w_j = (1/m) \Sigma (\hat{y}^i - y^i)x_j^i$

Обновление весов на каждом шаге:
$w_j := w_j - \alpha \cdot \partial J/\partial w_j$
где α - скорость обучения.

5. Принятие решения
После вычисления вероятности $\hat{y}$ применяется пороговое правило:
$\hat{y} \geq 0.5 \to$ класс 1
$\hat{y} < 0.5 \to$ класс 0

Практическая часть

Задание 1. Реализация логистической регрессии

1. Сгенерируйте синтетический набор данных для бинарной классификации с двумя признаками (используйте make_classification из sklearn.datasets)
2. Визуализируйте данные на scatter plot, раскрасив точки по классам

3. Реализуйте вручную:
    - Сигмоидную функцию
    - Функцию потерь (кросс-энтропию)
    - Градиентный спуск для оптимизации параметров

4. Обучите модель на тренировочных данных

Задание 2. Оценка качества модели

1. Вычислите предсказания на тестовом наборе
2. Постройте матрицу ошибок (confusion matrix)
3. Рассчитайте метрики качества:
    - Accuracy (доля правильных ответов)
    - Precision (точность)
    - Recall (полнота)
    - F1-score (гармоническое среднее precision и recall)

4. Постройте ROC-кривую и вычислите AUC-ROC

Задание 3. Сравнение с готовой реализацией

1. Используйте LogisticRegression из sklearn.linear_model
2. Сравните полученные коэффициенты с вашей реализацией
3. Проверьте совпадение метрик качества
4. Визуализируйте границу принятия решений для обеих моделей

Аппаратура и материалы

1. Персональный компьютер с ОС Windows/Linux
2. Программное обеспечение: Python 3.x с библиотеками NumPy, Matplotlib, scikit-learn

Методика выполнения

1. Для генерации данных используйте make_classification с параметрами n_samples=1000, n_features=2, n_classes=2
2. Разделите данные на тренировочную и тестовую выборки в соотношении 70/30
3. Нормализуйте данные с помощью StandardScaler
4. Для визуализации используйте matplotlib.pyplot
5. Готовую реализацию возьмите из sklearn.linear_model.LogisticRegression

Пример кода (Python):

```
import numpy as np
```

```
import matplotlib.pyplot as plt
from sklearn.datasets import make_classification
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import confusion_matrix, classification_report, roc_curve, au
c
# Генерация данных
X, y = make_classification(n_samples=1000, n_features=2, n_classes=2,
                           n_redundant=0, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state
=42)
# Нормализация
scaler = StandardScaler()
X_train = scaler.fit_transform(X_train)
X_test = scaler.transform(X_test)
# Сигмоидная функция
def sigmoid(z):
    return 1 / (1 + np.exp(-z))
# Функция потерь
def cross_entropy_loss(y, y_pred):
    return -np.mean(y * np.log(y_pred) + (1-y) * np.log(1-y_pred))
# Градиентный спуск
def logistic_regression(X, y, learning_rate=0.01, n_iters=1000):
    m, n = X.shape
    weights = np.zeros(n)
    bias = 0
    losses = []
    for i in range(n_iters):
        linear_model = np.dot(X, weights) + bias
        y_pred = sigmoid(linear_model)
        # Градиенты
        dw = (1/m) * np.dot(X.T, (y_pred - y))
        db = (1/m) * np.sum(y_pred - y)
        # Обновление параметров
        weights -= learning_rate * dw
        bias -= learning_rate * db
        # Сохранение значения функции потерь
        loss = cross_entropy_loss(y, y_pred)
        losses.append(loss)
    return weights, bias, losses
# Обучение модели
w, b, losses = logistic_regression(X_train, y_train)
# Предсказания
def predict(X, weights, bias):
    linear_model = np.dot(X, weights) + bias
```

```
    y_pred = sigmoid(linear_model)
    return [1 if i >= 0.5 else 0 for i in y_pred]
y_pred = predict(X_test, w, b)
# Оценка качества
print("Confusion Matrix:")
print(confusion_matrix(y_test, y_pred))
print("\nClassification Report:")
print(classification_report(y_test, y_pred))
# ROC-кривая
fpr, tpr, thresholds = roc_curve(y_test, y_pred)
roc_auc = auc(fpr, tpr)
plt.figure(figsize=(10, 6))
plt.plot(fpr, tpr, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], 'k--')
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('ROC Curve')
plt.legend()
plt.show()
```

Оформление отчета

1. **Титульный лист** с указанием учебного заведения, названия работы, ФИО исполнителя и преподавателя, даты выполнения
2. **Теоретическая часть** (1-2 страницы):
    - Описание логистической регрессии
    - Сигмоидная функция и её свойства
    - Функция потерь кросс-энтропия
    - Метод оптимизации параметров
3. **Практическая часть**:
    - График сгенерированных данных
    - График изменения функции потерь в процессе обучения
    - Матрица ошибок и метрики качества
    - ROC-кривая с AUC
    - График границы принятия решений
    - Листинг программы с комментариями
4. **Выводы**:
    - Анализ качества классификации
    - Сравнение ручной реализации и библиотечной
    - Оценка влияния параметров на работу модели

Вопросы для защиты

1. Почему в логистической регрессии используется сигмоидная функция?
2. Как интерпретируется значение кросс-энтропии?

3. В чем отличие логистической регрессии от линейной?
4. Какие метрики качества используются для оценки бинарного классификатора?
5. Как выбирается порог классификации в логистической регрессии?

Лабораторная работа №5. Метод опорных векторов (SVM): линейные и нелинейные SVM. Ядерные функции

Цель работы: изучение теоретических основ и практическое освоение метода опорных векторов для задач классификации, включая линейный и нелинейный случаи с применением ядерных функций. Получение навыков реализации и настройки SVM-моделей на реальных данных.

Теоретическая часть

1. Основы метода опорных векторов

Метод опорных векторов (Support Vector Machine, SVM) - это мощный алгоритм машинного обучения для задач классификации и регрессии. Основная идея SVM заключается в нахождении оптимальной разделяющей гиперплоскости в пространстве признаков, которая максимизирует зазор между классами.

2. Линейный SVM

Для линейно разделимых данных SVM ищет гиперплоскость $w \cdot x + b = 0$, которая удовлетворяет условиям:

- $w \cdot x^i + b \geq +1$ для $y^i = +1$
- $w \cdot x^i + b \leq -1$ для $y^i = -1$

Ширина зазора между классами равна $2/||w||$. Задача сводится к минимизации $||w||^2/2$ при указанных ограничениях (задача квадратичного программирования).

3. Нелинейный SVM и ядерный трюк

Для нелинейно разделимых данных применяется преобразование признаков в пространство более высокой размерности, где разделение становится линейным. Ядерные функции позволяют вычислять скалярные произведения в этом пространстве без явного преобразования:

$K(x^i, x^j) = \varphi(x^i) \cdot \varphi(x^j)$

Основные ядерные функции:

- Линейное: $K(x^i, x^j) = x^i \cdot x^j$
- Полиномиальное: $K(x^i, x^j) = (\gamma x^i \cdot x^j + r)^d$

- Радиальное базисное (RBF): $K(x^i, x^j) = \exp(-\gamma||x^i - x^j||^2)$
- Сигмоидное: $K(x^i, x^j) = \tanh(\gamma x^i \cdot x^j + r)$

4. Мягкий зазор (soft margin)
Для данных с перекрывающимися классами вводится параметр регуляризации C, который контролирует компромисс между максимизацией зазора и минимизацией ошибок классификации.

Практическая часть

Задание 1. Линейный SVM

1. Сгенерируйте линейно разделимый набор данных с двумя признаками и двумя классами (make_blobs)
2. Разделите данные на обучающую и тестовую выборки (70/30)
3. Обучите линейную SVM-модель с разными значениями параметра C (0.1, 1, 10)
4. Визуализируйте разделяющие гиперплоскости и опорные векторы для каждого случая
5. Оцените качество классификации на тестовой выборке

Задание 2. Нелинейный SVM с ядерными функциями

1. Сгенерируйте нелинейно разделимый набор данных (make_circles или make_moons)
2. Исследуйте влияние разных ядерных функций:
    - Линейное
    - Полиномиальное (степени 2, 3, 5)
    - RBF (разные значения γ)
3. Для каждого случая:
    - Обучите модель
    - Визуализируйте границу решения
    - Вычислите точность классификации
4. Сравните результаты и сделайте выводы

Задание 3. Подбор параметров

1. Для RBF-ядра исследуйте влияние параметров C и γ на качество модели
2. Постройте тепловую карту точности в координатах (C, γ)
3. Выберите оптимальные параметры с помощью кросс-валидации

Аппаратура и материалы

1. Персональный компьютер с ОС Windows/Linux

2. Программное обеспечение: Python 3.x с библиотеками NumPy, Matplotlib, scikit-learn

Методика выполнения

1. Для генерации данных используйте make_blobs и make_circles из sklearn.datasets
2. Для визуализации применяйте matplotlib.pyplot
3. Для реализации SVM используйте sklearn.svm.SVC
4. Для оценки качества применяйте accuracy_score и confusion_matrix

Пример кода (Python):

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs, make_circles
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
# Задание 1. Линейный SVM
X, y = make_blobs(n_samples=100, centers=2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state
=42)
# Обучение моделей с разными C
for C in [0.1, 1, 10]:
    model = SVC(kernel='linear', C=C)
    model.fit(X_train, y_train)
    # Визуализация
    plt.figure(figsize=(8, 6))
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='autumn')
    # Построение разделяющей гиперплоскости
    ax = plt.gca()
    xlim = ax.get_xlim()
    ylim = ax.get_ylim()
    # Сетка для предсказаний
    xx = np.linspace(xlim[0], xlim[1], 30)
    yy = np.linspace(ylim[0], ylim[1], 30)
    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T
    Z = model.decision_function(xy).reshape(XX.shape)
    # Границы и опорные векторы
    ax.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1],
               alpha=0.5, linestyles=['--', '-', '--'])
    ax.scatter(model.support_vectors_[:, 0],
               model.support_vectors_[:, 1],
```

```
            s=100, linewidth=1, facecolors='none', edgecolors='k')
    plt.title(f'Линейный SVM (C={C})')
    plt.show()
    # Оценка качества
    y_pred = model.predict(X_test)
    print(f'Точность (C={C}): {accuracy_score(y_test, y_pred):.2f}')
# Задание 2. Нелинейный SVM
X, y = make_circles(n_samples=100, factor=0.5, noise=0.1, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state
=42)
# Различные ядерные функции
kernels = [
    ('linear', {}),
    ('poly', {'degree': 2}),
    ('poly', {'degree': 3}),
    ('rbf', {'gamma': 0.1}),
    ('rbf', {'gamma': 1}),
    ('rbf', {'gamma': 10})
]
for kernel, params in kernels:
    model = SVC(kernel=kernel, **params)
    model.fit(X_train, y_train)
    # Визуализация
    plt.figure(figsize=(8, 6))
    plt.scatter(X[:, 0], X[:, 1], c=y, cmap='autumn')
    xx = np.linspace(-1.5, 1.5, 100)
    yy = np.linspace(-1.5, 1.5, 100)
    YY, XX = np.meshgrid(yy, xx)
    xy = np.vstack([XX.ravel(), YY.ravel()]).T
    Z = model.decision_function(xy).reshape(XX.shape)
    plt.contourf(XX, YY, Z > 0, alpha=0.2)
    plt.contour(XX, YY, Z, colors='k', levels=[-1, 0, 1],
                alpha=0.5, linestyles=['--', '-', '--'])
    plt.title(f'Ядро: {kernel}, параметры: {params}')
    plt.show()
    y_pred = model.predict(X_test)
    print(f'Точность ({kernel} {params}): {accuracy_score(y_test, y_pred):.2f}')
```

Оформление отчета

1. **Титульный лист** с указанием учебного заведения, названия работы, ФИО исполнителя и преподавателя, даты выполнения
2. **Теоретическая часть** (1-2 страницы):
    - Принцип работы линейного SVM
    - Ядерный трюк и нелинейные SVM

    - Основные ядерные функции и их параметры
3. **Практическая часть**:
    - Графики с разделяющими границами для разных ядер
    - Таблица с метриками качества для разных моделей
    - Тепловая карта зависимости точности от параметров
    - Листинг программы с комментариями
4. **Выводы**:
    - Сравнение эффективности разных ядер
    - Анализ влияния параметров на качество классификации
    - Рекомендации по выбору типа SVM для разных задач

Вопросы для защиты

1. В чем заключается идея максимизации зазора в SVM?
2. Какие преимущества дает использование ядерного трюка?
3. Как параметр C влияет на работу SVM?
4. В чем разница между линейным и RBF-ядром?
5. Как выбираются опорные векторы?

Лабораторная работа №6. Деревья решений и случайные леса: критерии разделения (энтропия, Джини), ансамбли моделей

Цель работы: изучение принципов построения деревьев решений и ансамблевых методов на их основе, освоение критериев разделения узлов (энтропия, индекс Джини), практическое применение случайных лесов для задач классификации и регрессии.

Теоретическая часть

1. Деревья решений
Дерево решений - это алгоритм машинного обучения, который строит модель в виде древовидной структуры, где:

- Внутренние узлы представляют признаки/атрибуты
- Ветви - правила принятия решений
- Листья - конечные результаты (классы или значения)

2. Критерии разделения узлов
Основные критерии для выбора наилучшего разделения:

1. **Энтропия (Information Gain):**
   Мера неопределенности системы. Для множества S:
   $Entropy(S) = -\Sigma(p_i \cdot \log_2 p_i)$
   где p_i - доля элементов класса i в S.

Прирост информации:
$IG(S,A) = Entropy(S) - \Sigma(|S_v|/|S|)\cdot Entropy(S_v)$
где A - атрибут, S_v - подмножество для значения v атрибута A.

2. **Индекс Джини:**
   Мера нечистоты узла:
   $Gini(S) = 1 - \Sigma p_i^2$
   где p_i - вероятность принадлежности к классу i.

3. Алгоритм построения дерева (ID3/CART):

1. Начать со всего набора данных как корневого узла
2. Для каждого атрибута вычислить критерий разделения
3. Выбрать атрибут с наилучшим разделением
4. Создать дочерние узлы для каждого значения атрибута
5. Рекурсивно повторить для каждого дочернего узла
6. Остановка при выполнении условий:
   - Все элементы одного класса
   - Нет атрибутов для разделения
   - Достигнута максимальная глубина

4. Случайные леса (Random Forest)
Ансамблевый метод, строящий множество деревьев и агрегирующий их результаты:

1. Бэггинг (bootstrap aggregating):
   - Многократная случайная выборка с повторением
   - Построение дерева для каждой выборки

2. Случайный выбор признаков:
   - Для каждого разделения рассматривается случайное подмножество признаков

3. Голосование (классификация) или усреднение (регрессия)

Преимущества:

- Устойчивость к переобучению
- Работа с пропущенными значениями
- Оценка важности признаков

Практическая часть

Задание 1. Построение дерева решений

1. Загрузите набор данных Iris (sklearn.datasets.load_iris)

2. Разделите данные на обучающую и тестовую выборки (70/30)
3. Обучите DecisionTreeClassifier с разными критериями:
    - Энтропия
    - Индекс Джини

4. Визуализируйте построенные деревья
5. Сравните точность на тестовой выборке

Задание 2. Анализ параметров дерева

1. Исследуйте влияние параметров:
    - max_depth (максимальная глубина)
    - min_samples_split (минимальное число образцов для разделения)
    - min_samples_leaf (минимальное число образцов в листе)

2. Постройте графики зависимости точности от параметров
3. Выберите оптимальные параметры

Задание 3. Случайный лес

1. Обучите RandomForestClassifier на тех же данных
2. Исследуйте влияние параметров:
    - n_estimators (количество деревьев)
    - max_features (число признаков для разделения)

3. Сравните точность с одиночным деревом
4. Проанализируйте важность признаков

Аппаратура и материалы

1. Персональный компьютер с ОС Windows/Linux
2. Программное обеспечение: Python 3.x с библиотеками NumPy, Matplotlib, scikit-learn, graphviz

Методика выполнения

1. Для визуализации деревьев используйте export_graphviz или plot_tree
2. Для оценки качества применяйте accuracy_score
3. Для анализа важности признаков - feature_importances_
4. Эксперименты проводите с фиксированным random_state для воспроизводимости

Пример кода (Python):

```
import numpy as np
import matplotlib.pyplot as plt
```

```
from sklearn.datasets import load_iris
from sklearn.tree import DecisionTreeClassifier, export_graphviz
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import graphviz
# Загрузка данных
iris = load_iris()
X, y = iris.data, iris.target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state
=42)
# Задание 1. Деревья с разными критериями
for criterion in ['gini', 'entropy']:
    clf = DecisionTreeClassifier(criterion=criterion, random_state=42)
    clf.fit(X_train, y_train)
    # Визуализация
    dot_data = export_graphviz(clf, out_file=None,
                               feature_names=iris.feature_names,
                               class_names=iris.target_names,
                               filled=True, rounded=True)
    graph = graphviz.Source(dot_data)
    graph.render(f'iris_tree_{criterion}')
    # Оценка точности
    y_pred = clf.predict(X_test)
    print(f'Точность (критерий {criterion}): {accuracy_score(y_test, y_pred):.2f}')
# Задание 2. Влияние параметров дерева
depths = range(1, 10)
train_acc = []
test_acc = []
for depth in depths:
    clf = DecisionTreeClassifier(max_depth=depth, random_state=42)
    clf.fit(X_train, y_train)
    train_acc.append(accuracy_score(y_train, clf.predict(X_train)))
    test_acc.append(accuracy_score(y_test, clf.predict(X_test)))
plt.figure(figsize=(10, 6))
plt.plot(depths, train_acc, label='Обучающая выборка')
plt.plot(depths, test_acc, label='Тестовая выборка')
plt.xlabel('Глубина дерева')
plt.ylabel('Точность')
plt.legend()
plt.grid()
plt.show()
# Задание 3. Случайный лес
n_estimators = range(1, 101, 10)
forest_acc = []
```

```
for n in n_estimators:
    rf = RandomForestClassifier(n_estimators=n, random_state=42)
    rf.fit(X_train, y_train)
    forest_acc.append(accuracy_score(y_test, rf.predict(X_test)))
plt.figure(figsize=(10, 6))
plt.plot(n_estimators, forest_acc)
plt.xlabel('Количество деревьев')
plt.ylabel('Точность')
plt.grid()
plt.show()
# Важность признаков
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X_train, y_train)
plt.figure(figsize=(10, 6))
plt.barh(iris.feature_names, rf.feature_importances_)
plt.xlabel('Важность признака')
plt.show()
```

Оформление отчета

1. **Титульный лист**
2. **Теоретическая часть** (1-2 страницы):
   - Принцип работы деревьев решений
   - Критерии разделения узлов
   - Алгоритм построения случайного леса

3. **Практическая часть**:
   - Графики построенных деревьев
   - Зависимость точности от параметров
   - Сравнение разных методов
   - Анализ важности признаков
   - Листинг программы с комментариями

4. **Выводы**:
   - Сравнение критериев разделения
   - Оптимальные параметры моделей
   - Преимущества ансамблевых методов

Вопросы для защиты

1. Как работает критерий информационного прироста?
2. В чем разница между энтропией и индексом Джини?
3. Какие параметры дерева влияют на переобучение?
4. Почему случайный лес работает лучше одиночного дерева?
5. Как интерпретировать важность признаков?

Лабораторная работа №7. Кластеризация данных: метод k-средних. Алгоритм кластеризации. Метрики оценки качества

Цель работы: изучение принципов работы алгоритма k-средних, освоение методов оценки качества кластеризации, практическое применение алгоритма для группировки данных, исследование влияния параметров на результаты кластеризации.

Теоретическая часть

1. Понятие кластеризации
Кластеризация - это задача разбиения множества объектов на группы (кластеры) таким образом, чтобы:

- Объекты внутри одного кластера были "похожи" друг на друга
- Объекты разных кластеров существенно "отличались"

2. Алгоритм k-средних (k-means)
Итерационный алгоритм, минимизирующий сумму квадратов расстояний от точек до центроидов их кластеров:

1. **Инициализация**: Выбрать k начальных центроидов (случайно или специальным способом)
2. **Назначение кластеров**: Каждой точке назначить ближайший центроид
3. **Пересчет центроидов**: Вычислить новые центроиды как среднее точек кластера
4. **Проверка сходимости**: Если центроиды не изменились или достигнуто максимальное число итераций - остановка, иначе вернуться к шагу 2

Математическая формулировка:
Минимизировать $J = \Sigma\Sigma \, ||x - \mu_i||^2$, где μ_i - центроид кластера C_i

3. Методы инициализации центроидов

- Случайный выбор k точек из набора данных
- Алгоритм k-means++ (предпочтительный выбор далеко расположенных центроидов)
- Случайное распределение в пространстве признаков

4. Метрики оценки качества кластеризации

1. **Внутрикластерная дисперсия** (Inertia):
   $\Sigma\Sigma \, ||x - \mu_i||^2 \rightarrow \min$

2. **Силуэтный коэффициент** (Silhouette Score):
   s(i) = (b(i) - a(i)) / max{a(i), b(i)}
   где a(i) - среднее расстояние до объектов своего кластера,
   b(i) - среднее расстояние до объектов ближайшего другого кластера
3. **Индекс Дэвиса-Болдина** (DBI):
   Среднее максимального отношения внутрикластерного рассеяния к межкластерному расстоянию

5. Определение оптимального числа кластеров
Методы:

- Метод локтя (анализ графика зависимости инерции от k)
- Анализ силуэтного коэффициента
- Индекс Калински-Харабаза

Практическая часть

Задание 1. Реализация k-means

1. Сгенерируйте синтетические данные (make_blobs) с явными кластерами
2. Реализуйте алгоритм k-means вручную:
   - Инициализация центроидов
   - Назначение кластеров
   - Пересчет центроидов
   - Проверка сходимости
3. Визуализируйте процесс кластеризации на каждой итерации
4. Сравните с готовой реализацией (sklearn.cluster.KMeans)

Задание 2. Оценка качества кластеризации

1. Для разных значений k (2-10) вычислите:
   - Внутрикластерную дисперсию
   - Силуэтный коэффициент
   - Индекс Дэвиса-Болдина
2. Постройте графики зависимостей метрик от k
3. Определите оптимальное число кластеров

Задание 3. Применение к реальным данным

1. Загрузите набор данных (например, Iris)
2. Проведите кластеризацию для k=3
3. Сравните с истинными классами (если доступны)
4. Визуализируйте результаты (PCA для уменьшения размерности)

Аппаратура и материалы

1. Персональный компьютер с ОС Windows/Linux
2. Программное обеспечение: Python 3.x с библиотеками NumPy, Matplotlib, scikit-learn

Методика выполнения

1. Для генерации данных используйте make_blobs
2. Для визуализации применяйте matplotlib.pyplot
3. Для оценки качества используйте sklearn.metrics
4. Для уменьшения размерности - sklearn.decomposition.PCA

Пример кода (Python):

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import make_blobs
from sklearn.cluster import KMeans
from sklearn.metrics import silhouette_score, davies_bouldin_score
from sklearn.decomposition import PCA
# Задание 1. Реализация k-means
X, y = make_blobs(n_samples=300, centers=4, cluster_std=0.6, random_state=42)
# Визуализация исходных данных
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], s=50)
plt.title("Исходные данные")
plt.show()
# Ручная реализация k-means
class KMeansManual:
    def __init__(self, n_clusters=3, max_iter=100):
        self.n_clusters = n_clusters
        self.max_iter = max_iter
    def fit(self, X):
        # Инициализация центроидов
        self.centroids = X[np.random.choice(X.shape[0], self.n_clusters, replace=Fals
e)]
        for _ in range(self.max_iter):
            # Назначение кластеров
            distances = np.sqrt(((X - self.centroids[:, np.newaxis])**2).sum(axis=2))
            self.labels_ = np.argmin(distances, axis=0)
            # Пересчет центроидов
            new_centroids = np.array([X[self.labels_ == i].mean(axis=0)
                                      for i in range(self.n_clusters)])
            # Проверка сходимости
```

```
            if np.allclose(self.centroids, new_centroids):
                break

            self.centroids = new_centroids
        return self
model = KMeansManual(n_clusters=4)
model.fit(X)
# Визуализация результатов
plt.figure(figsize=(10, 6))
plt.scatter(X[:, 0], X[:, 1], c=model.labels_, s=50, cmap='viridis')
plt.scatter(model.centroids[:, 0], model.centroids[:, 1],
            c='red', s=200, alpha=0.8, marker='X')
plt.title("Ручная реализация k-means")
plt.show()
# Задание 2. Оценка качества
k_values = range(2, 11)
inertias = []
silhouettes = []
davies_bouldin = []
for k in k_values:
    kmeans = KMeans(n_clusters=k, random_state=42)
    labels = kmeans.fit_predict(X)
    inertias.append(kmeans.inertia_)
    silhouettes.append(silhouette_score(X, labels))
    davies_bouldin.append(davies_bouldin_score(X, labels))
# График метода локтя
plt.figure(figsize=(10, 6))
plt.plot(k_values, inertias, 'bo-')
plt.xlabel('Число кластеров')
plt.ylabel('Инерция')
plt.title('Метод локтя')
plt.grid()
plt.show()
# График силуэтного коэффициента
plt.figure(figsize=(10, 6))
plt.plot(k_values, silhouettes, 'bo-')
plt.xlabel('Число кластеров')
plt.ylabel('Силуэтный коэффициент')
plt.grid()
plt.show()
# График индекса Дэвиса-Болдина
plt.figure(figsize=(10, 6))
plt.plot(k_values, davies_bouldin, 'bo-')
plt.xlabel('Число кластеров')
plt.ylabel('Индекс Дэвиса-Болдина')
```

```
plt.grid()
plt.show()
# Задание 3. Применение к реальным данным
from sklearn.datasets import load_iris
iris = load_iris()
X_iris = iris.data
y_iris = iris.target
# Уменьшение размерности для визуализации
pca = PCA(n_components=2)
X_iris_2d = pca.fit_transform(X_iris)
kmeans_iris = KMeans(n_clusters=3, random_state=42)
labels_iris = kmeans_iris.fit_predict(X_iris)
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(X_iris_2d[:, 0], X_iris_2d[:, 1], c=y_iris, cmap='viridis')
plt.title("Истинные классы")
plt.subplot(1, 2, 2)
plt.scatter(X_iris_2d[:, 0], X_iris_2d[:, 1], c=labels_iris, cmap='viridis')
plt.title("Кластеризация k-means")
plt.show()
```

Оформление отчета

1. **Титульный лист**
2. **Теоретическая часть** (1-2 страницы):
   - Принцип работы алгоритма k-means
   - Методы инициализации центроидов
   - Метрики оценки качества кластеризации

3. **Практическая часть**:
   - Графики процесса кластеризации
   - Графики зависимостей метрик от числа кластеров
   - Визуализация результатов на реальных данных
   - Листинг программы с комментариями

4. **Выводы**:
   - Анализ работы алгоритма
   - Определение оптимального числа кластеров
   - Сравнение с истинными классами (для Iris)

Вопросы для защиты

1. Как работает алгоритм k-means?
2. Какие проблемы могут возникнуть при случайной инициализации?
3. Как определить оптимальное число кластеров?

4. В чем преимущества силуэтного коэффициента перед внутрикластерной дисперсией?
5. Какие ограничения имеет метод k-means?

Лабораторная работа №8. Понижение размерности: PCA и t-SNE. Метод главных компонент. Визуализация многомерных данных

Цель работы: изучение методов понижения размерности данных, освоение алгоритмов PCA и t-SNE, практическое применение этих методов для визуализации многомерных данных, анализ их сравнительных характеристик и областей применения.

Теоретическая часть

1. Проблема многомерности данных

При работе с реальными наборами данных часто приходится сталкиваться с большим количеством признаков (десятки, сотни и более). Это создает проблемы:

- Вычислительная сложность обработки
- Проклятие размерности (sparsity данных)
- Трудности визуализации и интерпретации

2. Метод главных компонент (PCA)

PCA - линейный метод понижения размерности, который находит направления максимальной дисперсии данных и проецирует данные на эти направления (главные компоненты).

Математические основы:

1. Центрирование данных: $X' = X - \mu$
2. Вычисление ковариационной матрицы: $C = (X')^T X'/(n-1)$
3. Сингулярное разложение (SVD): $C = U\Sigma V^T$
4. Главные компоненты - собственные векторы U
5. Проецирование: $Z = X'U$

Свойства PCA:

- Сохраняет глобальную структуру данных
- Линейное преобразование
- Компоненты ортогональны
- Упорядочены по объясненной дисперсии

3. t-SNE (t-distributed Stochastic Neighbor Embedding)
Нелинейный метод визуализации, сохраняющий локальные расстояния между точками.

Алгоритм:

1. Вычисление матрицы сходств в исходном пространстве (условные вероятности pj|i)
2. Построение аналогичной матрицы в пространстве меньшей размерности (qj|i)
3. Минимизация расхождения Кульбака-Лейблера между распределениями:
   $KL(P||Q) = \Sigma p_{ij} \log(p_{ij}/q_{ij})$

Особенности t-SNE:

- Сохраняет локальную структуру данных
- Нелинейное преобразование
- Чувствителен к параметру perplexity
- Результаты разных запусков могут отличаться

4. Сравнение PCA и t-SNE

- PCA лучше для выявления глобальной структуры
- t-SNE лучше для визуализации локальных кластеров
- PCA воспроизводим и детерминирован
- t-SNE требует подбора параметров
- PCA быстрее для больших данных

Практическая часть

Задание 1. Применение PCA

1. Загрузите набор данных digits (sklearn.datasets.load_digits)
2. Примените PCA с разным числом компонент (2, 10, 30)
3. Визуализируйте данные в 2D после преобразования
4. Постройте график объясненной дисперсии
5. Восстановите изображения цифр из уменьшенного представления

Задание 2. Применение t-SNE

1. Используйте тот же набор данных digits
2. Примените t-SNE с разными значениями perplexity (5, 30, 50)
3. Визуализируйте результаты в 2D
4. Сравните кластерную структуру с PCA

Задание 3. Сравнительный анализ

1. Для набора данных wine (sklearn.datasets.load_wine):
   - Примените PCA и t-SNE
   - Визуализируйте результаты
   - Сравните сохранение структуры классов

2. Измерьте время выполнения для обоих методов
3. Проанализируйте различия в результатах

Аппаратура и материалы

1. Персональный компьютер с ОС Windows/Linux
2. Программное обеспечение: Python 3.x с библиотеками NumPy, Matplotlib, scikit-learn

Методика выполнения

1. Для загрузки данных используйте sklearn.datasets
2. Для PCA - sklearn.decomposition.PCA
3. Для t-SNE - sklearn.manifold.TSNE
4. Для визуализации - matplotlib.pyplot
5. Для оценки времени - time.time()

Пример кода (Python):

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import load_digits, load_wine
from sklearn.decomposition import PCA
from sklearn.manifold import TSNE
import time
# Задание 1. PCA
digits = load_digits()
X_digits, y_digits = digits.data, digits.target
# Визуализация исходных данных
plt.figure(figsize=(10, 6))
for i in range(10):
    plt.subplot(2, 5, i+1)
    plt.imshow(X_digits[y_digits == i][0].reshape(8, 8), cmap='gray')
    plt.title(f"Цифра {i}")
    plt.axis('off')
plt.show()
# Применение PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X_digits)
```

```
plt.figure(figsize=(10, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=y_digits,
                      cmap='tab10', alpha=0.6)
plt.colorbar(scatter)
plt.title("PCA проекция данных digits (2 компоненты)")
plt.xlabel("Первая главная компонента")
plt.ylabel("Вторая главная компонента")
plt.grid()
plt.show()
# График объясненной дисперсии
pca_full = PCA().fit(X_digits)
plt.figure(figsize=(10, 6))
plt.plot(np.cumsum(pca_full.explained_variance_ratio_))
plt.xlabel('Число компонент')
plt.ylabel('Накопленная объясненная дисперсия')
plt.grid()
plt.show()
# Восстановление изображений
pca30 = PCA(n_components=30)
X_pca30 = pca30.fit_transform(X_digits)
X_reconstructed = pca30.inverse_transform(X_pca30)
plt.figure(figsize=(10, 6))
for i in range(5):
    plt.subplot(2, 5, i+1)
    plt.imshow(X_digits[i].reshape(8, 8), cmap='gray')
    plt.title("Исходная")
    plt.axis('off')
    plt.subplot(2, 5, i+6)
    plt.imshow(X_reconstructed[i].reshape(8, 8), cmap='gray')
    plt.title("Восстановл.")
    plt.axis('off')
plt.show()
# Задание 2. t-SNE
perplexities = [5, 30, 50]
plt.figure(figsize=(15, 5))
for i, perplexity in enumerate(perplexities, 1):
    tsne = TSNE(n_components=2, perplexity=perplexity, random_state=42)
    X_tsne = tsne.fit_transform(X_digits)
    plt.subplot(1, 3, i)
    plt.scatter(X_tsne[:, 0], X_tsne[:, 1], c=y_digits, cmap='tab10', alpha=0.6)
    plt.title(f"t-SNE, perplexity={perplexity}")
    plt.grid()
plt.tight_layout()
plt.show()
# Задание 3. Сравнение на данных wine
```

```
wine = load_wine()
X_wine, y_wine = wine.data, wine.target
# PCA
start = time.time()
X_wine_pca = PCA(n_components=2).fit_transform(X_wine)
pca_time = time.time() - start
# t-SNE
start = time.time()
X_wine_tsne = TSNE(n_components=2, perplexity=30, random_state=42).fit_tran
sform(X_wine)
tsne_time = time.time() - start
plt.figure(figsize=(12, 6))
plt.subplot(1, 2, 1)
plt.scatter(X_wine_pca[:, 0], X_wine_pca[:, 1], c=y_wine, cmap='viridis')
plt.title(f"PCA (время: {pca_time:.2f}с)")
plt.subplot(1, 2, 2)
plt.scatter(X_wine_tsne[:, 0], X_wine_tsne[:, 1], c=y_wine, cmap='viridis')
plt.title(f"t-SNE (время: {tsne_time:.2f}с)")
plt.tight_layout()
plt.show()
```

Оформление отчета

1. **Титульный лист**
2. **Теоретическая часть** (1-2 страницы):
   - Принцип работы PCA
   - Алгоритм t-SNE
   - Сравнительная характеристика методов

3. **Практическая часть**:
   - Графики проекций данных
   - График объясненной дисперсии
   - Примеры восстановленных изображений
   - Сравнение времени выполнения
   - Листинг программы с комментариями

4. **Выводы**:
   - Анализ результатов PCA
   - Анализ результатов t-SNE
   - Рекомендации по выбору метода

Вопросы для защиты

1. Как PCA находит главные компоненты?
2. В чем смысл объясненной дисперсии?

3. Как параметр perplexity влияет на t-SNE?
4. Когда следует использовать PCA, а когда t-SNE?
5. Почему t-SNE требует больше вычислительных ресурсов?

Лабораторная работа №9. Байесовские методы классификации: наивный байесовский классификатор. Применение в NLP

Цель работы: изучение теоретических основ и практическое освоение наивного байесовского классификатора, исследование его применения в задачах обработки естественного языка (NLP), сравнение эффективности различных вариантов алгоритма на реальных текстовых данных.

Теоретическая часть

1. Теорема Байеса

Основой байесовских методов является теорема Байеса:
$P(A|B) = P(B|A)\cdot P(A)/P(B)$
где:

- $P(A|B)$ - апостериорная вероятность
- $P(B|A)$ - правдоподобие
- $P(A)$ - априорная вероятность
- $P(B)$ - нормирующая константа

2. Наивный байесовский классификатор

Наивный байесовский классификатор основан на предположении о независимости признаков при данном классе. Для класса C и признаков $x_1,...,x_n$:
$P(C|x_1,...,x_n) \propto P(C)\cdot ПР(x_i|C)$

Основные типы:

1. **Гауссовский (GaussianNB)** - для непрерывных признаков
2. **Мультиномиальный (MultinomialNB)** - для дискретных счетов (частот слов)
3. **Бернуллиевский (BernoulliNB)** - для бинарных признаков

3. Применение в NLP

Основные этапы обработки текста:

1. Токенизация (разбиение на слова/фразы)
2. Удаление стоп-слов (предлоги, союзы)
3. Лемматизация/стемминг (приведение к базовой форме)
4. Векторизация (преобразование в числовые признаки):
    - Мешок слов (Bag of Words)

- TF-IDF (Term Frequency-Inverse Document Frequency)

4. Оценка качества классификации
Основные метрики:

- Accuracy (доля верных ответов)
- Precision (точность)
- Recall (полнота)
- F1-score (гармоническое среднее)
- Confusion matrix (матрица ошибок)

Практическая часть

Задание 1. Реализация наивного Байеса

1. Сгенерируйте синтетические данные с двумя классами и двумя признаками
2. Реализуйте вручную:
    - Расчет априорных вероятностей
    - Расчет условных вероятностей
    - Классификацию новых объектов

3. Визуализируйте разделяющую границу
4. Сравните с готовой реализацией (sklearn.naive_bayes.GaussianNB)

Задание 2. Классификация текстов

1. Загрузите набор данных 20 Newsgroups
2. Проведите предобработку текстов:
    - Токенизация
    - Удаление стоп-слов
    - Стемминг

3. Преобразуйте тексты в векторы:
    - Мешок слов
    - TF-IDF

4. Обучите MultinomialNB и BernoulliNB
5. Оцените качество классификации

Задание 3. Сравнение методов

1. Сравните результаты для:
    - Разных методов векторизации
    - Разных вариантов наивного Байеса
    - Разных размеров словаря

2. Проанализируйте наиболее значимые слова для каждого класса
3. Постройте кривые обучения

Аппаратура и материалы

1. Персональный компьютер с ОС Windows/Linux
2. Программное обеспечение: Python 3.x с библиотеками NumPy, Matplotlib, scikit-learn, NLTK

Методика выполнения

1. Для работы с текстами используйте sklearn.feature_extraction.text
2. Для предобработки - nltk.stem, nltk.corpus.stopwords
3. Для визуализации - matplotlib.pyplot
4. Для оценки качества - sklearn.metrics

Пример кода (Python):

```
import numpy as np
import matplotlib.pyplot as plt
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB, BernoulliNB
from sklearn.metrics import classification_report, confusion_matrix
from nltk.stem import PorterStemmer
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
import string
import nltk
nltk.download('punkt')
nltk.download('stopwords')
# Задание 1. Реализация наивного Байеса
class NaiveBayesManual:
    def __init__(self):
        self.classes = None
        self.priors = None
        self.means = None
        self.stds = None
    def fit(self, X, y):
        self.classes = np.unique(y)
        n_classes = len(self.classes)
        n_features = X.shape[1]
        self.priors = np.zeros(n_classes)
        self.means = np.zeros((n_classes, n_features))
        self.stds = np.zeros((n_classes, n_features))
        for i, c in enumerate(self.classes):
```

```
            X_c = X[y == c]
            self.priors[i] = X_c.shape[0] / X.shape[0]
            self.means[i, :] = X_c.mean(axis=0)
            self.stds[i, :] = X_c.std(axis=0)
        return self
    def predict(self, X):
        posteriors = []
        for i, c in enumerate(self.classes):
            prior = np.log(self.priors[i])
            likelihood = np.sum(np.log(self._pdf(i, X)), axis=1)
            posterior = prior + likelihood
            posteriors.append(posterior)
        return self.classes[np.argmax(np.array(posteriors).T, axis=1)]
    def _pdf(self, class_idx, X):
        mean = self.means[class_idx]
        std = self.stds[class_idx]
        numerator = np.exp(-(X - mean)**2 / (2 * std**2))
        denominator = std * np.sqrt(2 * np.pi)
        return numerator / denominator
# Генерация данных и сравнение
np.random.seed(42)
X = np.random.randn(200, 2)
y = np.concatenate([np.zeros(100), np.ones(100)])
X[y == 1] += 2
model = NaiveBayesManual().fit(X, y)
# Визуализация
x_min, x_max = X[:, 0].min() - 1, X[:, 0].max() + 1
y_min, y_max = X[:, 1].min() - 1, X[:, 1].max() + 1
xx, yy = np.meshgrid(np.linspace(x_min, x_max, 100),
                     np.linspace(y_min, y_max, 100))
Z = model.predict(np.c_[xx.ravel(), yy.ravel()]).reshape(xx.shape)
plt.figure(figsize=(10, 6))
plt.contourf(xx, yy, Z, alpha=0.3)
plt.scatter(X[:, 0], X[:, 1], c=y, edgecolor='k')
plt.title("Разделяющая граница наивного Байеса")
plt.show()
# Задание 2. Классификация текстов
categories = ['sci.med', 'sci.space']
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories)
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories)
# Функция предобработки
stemmer = PorterStemmer()
stop_words = set(stopwords.words('english'))
def preprocess(text):
    # Удаление пунктуации
```

```
    text = text.translate(str.maketrans('', '', string.punctuation))
    # Токенизация
    tokens = word_tokenize(text.lower())
    # Удаление стоп-слов и стемминг
    tokens = [stemmer.stem(token) for token in tokens
              if token not in stop_words and len(token) > 2]
    return ' '.join(tokens)
# Предобработка данных
X_train = [preprocess(text) for text in newsgroups_train.data]
X_test = [preprocess(text) for text in newsgroups_test.data]
y_train, y_test = newsgroups_train.target, newsgroups_test.target
# Векторизация
vectorizers = {
    'Count': CountVectorizer(max_features=1000),
    'TF-IDF': TfidfVectorizer(max_features=1000)
}
models = {
    'MultinomialNB': MultinomialNB(),
    'BernoulliNB': BernoulliNB()
}
results = {}
for vec_name, vectorizer in vectorizers.items():
    X_train_vec = vectorizer.fit_transform(X_train)
    X_test_vec = vectorizer.transform(X_test)
    for model_name, model in models.items():
        model.fit(X_train_vec, y_train)
        y_pred = model.predict(X_test_vec)
        report = classification_report(y_test, y_pred,
                                       target_names=newsgroups_test.target_names,
                                       output_dict=True)
        results[f"{vec_name}_{model_name}"] = report['accuracy']

        print(f"\n{vec_name} {model_name} Classification Report:")
        print(classification_report(y_test, y_pred,
                                    target_names=newsgroups_test.target_names))
# Визуализация результатов
plt.figure(figsize=(10, 6))
plt.bar(results.keys(), results.values())
plt.title("Сравнение точности разных методов")
plt.ylabel("Accuracy")
plt.xticks(rotation=45)
plt.show()
# Анализ значимых слов
vectorizer = TfidfVectorizer(max_features=1000)
X_train_vec = vectorizer.fit_transform(X_train)
```

```
model = MultinomialNB().fit(X_train_vec, y_train)
# Получение наиболее информативных слов
feature_names = vectorizer.get_feature_names_out()
top_n = 20
coefs = model.feature_log_prob_[1] - model.feature_log_prob_[0]
top_words = np.argsort(coefs)[-top_n:]
print("\nНаиболее значимые слова для классификации:")
for i in top_words:
    print(f"{feature_names[i]}: {coefs[i]:.2f}")
```

Оформление отчета

1. **Титульный лист**
2. **Теоретическая часть** (1-2 страницы):
   - Теорема Байеса и ее применение
   - Принцип работы наивного байесовского классификатора
   - Особенности применения в NLP

3. **Практическая часть**:
   - График разделяющей границы
   - Таблицы с метриками качества
   - Списки значимых слов
   - Листинг программы с комментариями

4. **Выводы**:
   - Анализ работы алгоритма
   - Сравнение вариантов реализации
   - Оценка применимости в NLP задачах

Вопросы для защиты

1. В чем заключается "наивность" наивного Байеса?
2. Как теорема Байеса применяется в классификации?
3. Какие методы векторизации текстов вы знаете?
4. Почему MultinomialNB хорошо работает с текстами?
5. Как оценить важность отдельных слов в классификации?

Лабораторная работа №10. Нейронные сети: перцептрон и обратное распространение. Архитектура нейросетей. Обучение на примере XOR

Цель работы: изучение принципов работы искусственных нейронных сетей, освоение алгоритмов обучения перцептрона и метода обратного распространения ошибки, практическая реализация нейросети для решения задачи XOR, анализ влияния параметров сети на процесс обучения.

Теоретическая часть

1. Искусственный нейрон

Основной элемент нейросети, моделирующий биологический нейрон.
Математическая модель:
$y = f(\Sigma w_i x_i + b)$
где:

- x_i - входные сигналы
- w_i - весовые коэффициенты
- b - смещение (bias)
- f - функция активации

2. Функции активации

1. Сигмоида: $f(x) = 1/(1 + e^{-x})$
2. Гиперболический тангенс: $f(x) = \tanh(x)$
3. ReLU: $f(x) = \max(0, x)$
4. Линейная: $f(x) = x$

3. Архитектура нейронных сетей

- **Однослойный перцептрон**: входной и выходной слои
- **Многослойный перцептрон (MLP)**: входной, скрытые и выходной слои
- **Полносвязная сеть**: каждый нейрон связан со всеми нейронами следующего слоя

4. Алгоритм обратного распространения ошибки

1. Прямой проход (вычисление выхода сети)
2. Вычисление ошибки $E = \frac{1}{2}\Sigma(y - \hat{y})^2$
3. Обратный проход (распространение ошибки от выходов к входам)
4. Обновление весов: $\Delta w_{ij} = -\eta \cdot \partial E/\partial w_{ij}$ (η - скорость обучения)

5. Задача XOR

Классическая проблема, демонстрирующая необходимость скрытых слоев:

- Линейно неразделимая функция
- Требуется минимум один скрытый слой с двумя нейронами
- Пример входов и выходов:
  $(0,0) \rightarrow 0$
  $(0,1) \rightarrow 1$
  $(1,0) \rightarrow 1$
  $(1,1) \rightarrow 0$

Практическая часть

Задание 1. Реализация перцептрона

1. Реализуйте однослойный перцептрон с сигмоидной функцией активации
2. Обучите на задачах AND и OR
3. Продемонстрируйте невозможность решения XOR
4. Визуализируйте разделяющие границы

Задание 2. Многослойная сеть для XOR

1. Реализуйте сеть с архитектурой 2-2-1:
    - Входной слой: 2 нейрона
    - Скрытый слой: 2 нейрона с tanh
    - Выходной слой: 1 нейрон с сигмоидой

2. Реализуйте алгоритм обратного распространения
3. Обучите сеть на данных XOR
4. Протестируйте на всех комбинациях входов

Задание 3. Анализ параметров

1. Исследуйте влияние:
    - Скорости обучения (0.1, 0.5, 1.0)
    - Количества эпох (100, 1000, 10000)
    - Функции активации (sigmoid, tanh, ReLU)

2. Постройте графики изменения ошибки
3. Определите оптимальные параметры

Аппаратура и материалы

1. Персональный компьютер с ОС Windows/Linux
2. Программное обеспечение: Python 3.x с библиотеками NumPy, Matplotlib

Методика выполнения

1. Для матричных операций используйте NumPy
2. Для визуализации применяйте Matplotlib
3. Реализуйте все вычисления "вручную" без использования нейросетевых фреймворков
4. Для воспроизводимости зафиксируйте random seed

Пример кода (Python):

```
import numpy as np
import matplotlib.pyplot as plt
# Функции активации
def sigmoid(x):
    return 1 / (1 + np.exp(-x))
def tanh(x):
    return np.tanh(x)
def relu(x):
    return np.maximum(0, x)
# Производные функций активации
def sigmoid_derivative(x):
    return sigmoid(x) * (1 - sigmoid(x))
def tanh_derivative(x):
    return 1 - np.tanh(x)**2
def relu_derivative(x):
    return (x > 0).astype(float)
# Задание 1. Перцептрон
class Perceptron:
    def __init__(self, input_size, activation=sigmoid, activation_der=sigmoid_deriv
ative):
        self.weights = np.random.randn(input_size)
        self.bias = np.random.randn()
        self.activation = activation
        self.activation_der = activation_der
    def forward(self, x):
        return self.activation(np.dot(x, self.weights) + self.bias)
    def train(self, X, y, epochs=1000, lr=0.1):
        errors = []
        for epoch in range(epochs):
            error = 0
            for x, target in zip(X, y):
                # Прямой проход
                output = self.forward(x)
                # Ошибка
                err = target - output
                error += err**2
                # Обновление весов
                self.weights += lr * err * self.activation_der(np.dot(x, self.weights) + sel
f.bias) * x
                self.bias += lr * err * self.activation_der(np.dot(x, self.weights) + self.bia
s)
            errors.append(error/len(X))
        return errors
# Данные AND и OR
X_and = np.array([[0,0], [0,1], [1,0], [1,1]])
```

```
y_and = np.array([0, 0, 0, 1])
X_or = np.array([[0,0], [0,1], [1,0], [1,1]])
y_or = np.array([0, 1, 1, 1])
# Обучение перцептрона
p_and = Perceptron(input_size=2)
errors_and = p_and.train(X_and, y_and)
p_or = Perceptron(input_size=2)
errors_or = p_or.train(X_or, y_or)
# Визуализация ошибок
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(errors_and)
plt.title('Ошибка обучения (AND)')
plt.xlabel('Эпоха')
plt.ylabel('MSE')
plt.subplot(1, 2, 2)
plt.plot(errors_or)
plt.title('Ошибка обучения (OR)')
plt.xlabel('Эпоха')
plt.ylabel('MSE')
plt.show()
# Проверка на XOR (не должен обучиться)
X_xor = np.array([[0,0], [0,1], [1,0], [1,1]])
y_xor = np.array([0, 1, 1, 0])
p_xor = Perceptron(input_size=2)
errors_xor = p_xor.train(X_xor, y_xor, epochs=10000)
print("Предсказания XOR однослойным перцептроном:")
for x in X_xor:
    print(f"{x} -> {p_xor.forward(x):.2f}")
# Задание 2. Многослойная сеть для XOR
class NeuralNetwork:
    def __init__(self, layers, activation=tanh, activation_der=tanh_derivative):
        self.layers = []
        for i in range(len(layers) - 1):
            self.layers.append({
                'weights': np.random.randn(layers[i], layers[i+1]),
                'bias': np.random.randn(layers[i+1])
            })
        self.activation = activation
        self.activation_der = activation_der
        self.output_activation = sigmoid
        self.output_activation_der = sigmoid_derivative
    def forward(self, x):
        activations = [x]
        for i, layer in enumerate(self.layers):
```

```
            if i == len(self.layers) - 1:  # Выходной слой
                z = np.dot(activations[-1], layer['weights']) + layer['bias']
                a = self.output_activation(z)
            else:  # Скрытые слои
                z = np.dot(activations[-1], layer['weights']) + layer['bias']
                a = self.activation(z)
            activations.append(a)
        return activations
    def train(self, X, y, epochs=10000, lr=0.1):
        errors = []
        for epoch in range(epochs):
            error = 0
            for x, target in zip(X, y):
                # Прямой проход
                activations = self.forward(x)
                output = activations[-1]
                # Ошибка
                err = target - output
                error += np.sum(err**2)
                # Обратное распространение
                deltas = [err * self.output_activation_der(activations[-1])]
                for i in reversed(range(len(self.layers) - 1)):
                    delta = np.dot(deltas[-1], self.layers[i+1]['weights'].T) * \
                            self.activation_der(activations[i+1])
                    deltas.append(delta)
                deltas.reverse()
                # Обновление весов
                for i in range(len(self.layers)):
                    self.layers[i]['weights'] += lr * np.outer(activations[i], deltas[i])
                    self.layers[i]['bias'] += lr * deltas[i]
            errors.append(error/len(X))
        return errors
# Создание и обучение сети 2-2-1
nn = NeuralNetwork([2, 2, 1])
errors_nn = nn.train(X_xor, y_xor, epochs=10000, lr=0.1)
# Визуализация ошибки
plt.figure(figsize=(8, 6))
plt.plot(errors_nn)
plt.title('Ошибка обучения многослойной сети (XOR)')
plt.xlabel('Эпоха')
plt.ylabel('MSE')
plt.show()
# Проверка работы сети
print("\nПредсказания XOR многослойной сетью:")
for x in X_xor:
```

```
    print(f"{x} -> {nn.forward(x)[-1][0]:.4f}")
# Задание 3. Анализ параметров
learning_rates = [0.01, 0.1, 0.5]
epochs_list = [1000, 5000, 10000]
activations = [
    ('sigmoid', sigmoid, sigmoid_derivative),
    ('tanh', tanh, tanh_derivative),
    ('ReLU', relu, relu_derivative)
]
plt.figure(figsize=(15, 10))
# Влияние скорости обучения
for i, lr in enumerate(learning_rates, 1):
    nn = NeuralNetwork([2, 2, 1])
    errors = nn.train(X_xor, y_xor, epochs=10000, lr=lr)
    plt.subplot(3, 3, i)
    plt.plot(errors)
    plt.title(f'Скорость обучения = {lr}')
    plt.xlabel('Эпоха')
    plt.ylabel('MSE')
# Влияние количества эпох
for i, epochs in enumerate(epochs_list, 4):
    nn = NeuralNetwork([2, 2, 1])
    errors = nn.train(X_xor, y_xor, epochs=epochs, lr=0.1)
    plt.subplot(3, 3, i)
    plt.plot(errors)
    plt.title(f'Эпох = {epochs}')
    plt.xlabel('Эпоха')
    plt.ylabel('MSE')
# Влияние функции активации
for i, (name, act, act_der) in enumerate(activations, 7):
    nn = NeuralNetwork([2, 2, 1], activation=act, activation_der=act_der)
    errors = nn.train(X_xor, y_xor, epochs=10000, lr=0.1)
    plt.subplot(3, 3, i)
    plt.plot(errors)
    plt.title(f'Активация: {name}')
    plt.xlabel('Эпоха')
    plt.ylabel('MSE')
plt.tight_layout()
plt.show()
```

Оформление отчета

1. **Титульный лист**
2. **Теоретическая часть** (1-2 страницы):
    - Архитектура нейронных сетей

    - Алгоритм обратного распространения ошибки
    - Особенности задачи XOR
3. **Практическая часть**:
    - Графики обучения для разных задач
    - Результаты тестирования сетей
    - Анализ влияния параметров
    - Листинг программы с комментариями
4. **Выводы**:
    - Сравнение однослойного и многослойного перцептронов
    - Оптимальные параметры обучения
    - Практические рекомендации

Вопросы для защиты

1. Почему однослойный перцептрон не может решить XOR?
2. Как работает алгоритм обратного распространения ошибки?
3. Какие функции активации лучше подходят для скрытых слоев?
4. Как скорость обучения влияет на процесс обучения?
5. Какие методы ускорения обучения нейросетей вы знаете?

Лабораторная работа №11. Сверточные нейронные сети (CNN): фильтры, пулинг, классификация изображений

Цель работы: изучение архитектуры сверточных нейронных сетей, освоение принципов работы фильтров и операций пулинга, практическое применение CNN для классификации изображений, анализ влияния параметров сети на качество распознавания.

Теоретическая часть

1. Основы CNN
Сверточные нейронные сети - специальный класс нейросетей для обработки данных с сеточной структурой (изображения, видео). Ключевые особенности:

- Локальные связи (рецептивные поля)
- Разделение весов (weight sharing)
- Инвариантность к небольшим смещениям

2. Слои CNN

1. **Сверточный слой (Convolutional)**:
    - Применяет фильтры (ядра свертки) к входным данным

- Вычисляет скалярное произведение фильтра и участка изображения
- Параметры: размер фильтра (3x3, 5x5), шаг (stride), дополнение (padding)

2. **Слой подвыборки (Pooling)**:
   - Уменьшает пространственные размеры
   - Виды: max pooling, average pooling
   - Сохраняет наиболее значимые признаки

3. **Полносвязный слой (Fully Connected)**:
   - Обычные нейроны для финальной классификации
   - Соединяет все нейроны предыдущего слоя с текущим

3. Операция свертки
Для входного изображения I и фильтра K размером F×F:
$S(i,j) = (I*K)(i,j) = \Sigma\sum I(m,n)\cdot K(i-m,j-n)$

4. Пулинг (Max Pooling)
Выбирает максимальное значение в окне:
$P(i,j) = \max\{I(m,n)\}$, где $(m,n) \in$ окно

5. Преимущества CNN

- Автоматическое выделение признаков
- Устойчивость к трансформациям
- Эффективная обработка многомерных данных

Практическая часть

Задание 1. Реализация свертки и пулинга

1. Создайте тестовое изображение (градиент, геометрические фигуры)
2. Реализуйте вручную:
   - Операцию свертки с разными фильтрами (размытие, выделение границ)
   - Операцию max pooling с разными размерами окна

3. Визуализируйте результаты каждого преобразования

Задание 2. Построение CNN

1. Загрузите набор данных CIFAR-10 (32x32 цветные изображения 10 классов)
2. Постройте CNN архитектуры:
   - Conv2D (32 фильтра 3x3) → ReLU → MaxPooling (2x2)

- Conv2D (64 фильтра 3x3) → ReLU → MaxPooling (2x2)
- Flatten → Dense (128) → ReLU → Dense (10) → Softmax

3. Обучите модель, оцените точность

Задание 3. Анализ работы CNN

1. Визуализируйте активации сверточных слоев
2. Исследуйте влияние:
   - Количества и размера фильтров
   - Размера окна пулинга
   - Функций активации (ReLU, LeakyReLU)

3. Сравните с полносвязной сетью аналогичной сложности

Аппаратура и материалы

1. Персональный компьютер с ОС Windows/Linux
2. Программное обеспечение: Python 3.x с библиотеками TensorFlow/Keras, NumPy, Matplotlib

Методика выполнения

1. Для работы с изображениями используйте OpenCV или PIL
2. Для построения CNN - TensorFlow/Keras
3. Для визуализации - Matplotlib
4. Для оценки качества - sklearn.metrics

Пример кода (Python):

```
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.datasets import cifar10
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense
from tensorflow.keras.utils import to_categorical
# Задание 1. Ручная реализация свертки и пулинга
def apply_convolution(image, kernel):
    """Применение свертки к изображению"""
    ih, iw = image.shape
    kh, kw = kernel.shape
    output = np.zeros((ih - kh + 1, iw - kw + 1))
    for i in range(ih - kh + 1):
        for j in range(iw - kw + 1):
            output[i,j] = np.sum(image[i:i+kh, j:j+kw] * kernel)
    return output
```

```
def apply_max_pooling(image, pool_size=2):
    """Применение max pooling к изображению"""
    ih, iw = image.shape
    output = np.zeros((ih // pool_size, iw // pool_size))
    for i in range(0, ih, pool_size):
        for j in range(0, iw, pool_size):
            output[i//pool_size, j//pool_size] = np.max(image[i:i+pool_size, j:j+pool_si
ze])
    return output
# Создание тестового изображения
image = np.zeros((100, 100))
image[20:40, 30:70] = 1  # Горизонтальная полоса
image[60:80, 30:70] = 1  # Горизонтальная полоса
image[30:70, 20:40] = 1  # Вертикальная полоса
image[30:70, 60:80] = 1  # Вертикальная полоса
# Фильтры
edge_filter = np.array([[-1, -1, -1],
                        [-1,  8, -1],
                        [-1, -1, -1]])
blur_filter = np.ones((3,3)) / 9
# Применение фильтров
edge_image = apply_convolution(image, edge_filter)
blur_image = apply_convolution(image, blur_filter)
# Применение pooling
pooled_image = apply_max_pooling(image, pool_size=2)
# Визуализация
plt.figure(figsize=(12, 8))
plt.subplot(2, 2, 1)
plt.imshow(image, cmap='gray')
plt.title("Исходное изображение")
plt.subplot(2, 2, 2)
plt.imshow(edge_image, cmap='gray')
plt.title("Фильтр границ")
plt.subplot(2, 2, 3)
plt.imshow(blur_image, cmap='gray')
plt.title("Фильтр размытия")
plt.subplot(2, 2, 4)
plt.imshow(pooled_image, cmap='gray')
plt.title("Max Pooling 2x2")
plt.show()
# Задание 2. Построение CNN для CIFAR-10
# Загрузка данных
(X_train, y_train), (X_test, y_test) = cifar10.load_data()
X_train = X_train.astype('float32') / 255
X_test = X_test.astype('float32') / 255
```

```
y_train = to_categorical(y_train, 10)
y_test = to_categorical(y_test, 10)
# Создание модели
model = Sequential([
    Conv2D(32, (3, 3), activation='relu', padding='same', input_shape=(32, 32, 3)),
    MaxPooling2D((2, 2)),
    Conv2D(64, (3, 3), activation='relu', padding='same'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(128, activation='relu'),
    Dense(10, activation='softmax')
])
# Компиляция модели
model.compile(optimizer='adam',
              loss='categorical_crossentropy',
              metrics=['accuracy'])
# Обучение
history = model.fit(X_train, y_train,
                    epochs=10,
                    batch_size=64,
                    validation_data=(X_test, y_test))
# Оценка точности
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f"\nТочность на тестовых данных: {test_acc:.4f}")
# Визуализация обучения
plt.figure(figsize=(12, 5))
plt.subplot(1, 2, 1)
plt.plot(history.history['accuracy'], label='Точность на обучении')
plt.plot(history.history['val_accuracy'], label='Точность на валидации')
plt.title('Точность модели')
plt.xlabel('Эпохи')
plt.ylabel('Точность')
plt.legend()
plt.subplot(1, 2, 2)
plt.plot(history.history['loss'], label='Потери на обучении')
plt.plot(history.history['val_loss'], label='Потери на валидации')
plt.title('Потери модели')
plt.xlabel('Эпохи')
plt.ylabel('Потери')
plt.legend()
plt.show()
# Задание 3. Визуализация активаций
from tensorflow.keras.models import Model
# Создание модели для визуализации активаций
layer_outputs = [layer.output for layer in model.layers[:3]]
```

```
activation_model = Model(inputs=model.input, outputs=layer_outputs)
# Выбор тестового изображения
test_image = X_test[0:1]
activations = activation_model.predict(test_image
# Визуализация активаций первого слоя
first_layer_activation = activations[0]
plt.figure(figsize=(15, 5))
for i in range(32):
    plt.subplot(4, 8, i+1)
    plt.imshow(first_layer_activation[0, :, :, i], cmap='viridis')
    plt.axis('off')
plt.suptitle('Активации первого сверточного слоя')
plt.show()
```

Оформление отчета

1. **Титульный лист**
2. **Теоретическая часть** (1-2 страницы):
    - Принципы работы CNN
    - Операции свертки и пулинга
    - Архитектура современных CNN
3. **Практическая часть**:
    - Результаты ручной свертки и пулинга
    - Графики обучения CNN
    - Визуализация активаций слоев
    - Сравнение с полносвязной сетью
    - Листинг программы с комментариями
4. **Выводы**:
    - Анализ эффективности CNN
    - Влияние параметров сети
    - Практические рекомендации

Вопросы для защиты

1. В чем преимущество сверточных слоев перед полносвязными?
2. Как работает операция max pooling?
3. Почему CNN инвариантны к небольшим смещениям?
4. Как выбрать размер и количество фильтров?
5. Какие методы улучшения CNN вы знаете?

Лабораторная работа №12. Рекуррентные нейронные сети (RNN, LSTM). Обработка временных рядов и текстов

Цель и содержание: провести исследование алгоритмов функционирования рекуррентных нейронных сетей (RNN) и их модификаций, таких как LSTM (Long Short-Term Memory), для обработки временных рядов и текстовых данных.

Теоретическое обоснование
Рекуррентные нейронные сети (RNN) — это класс искусственных нейронных сетей, предназначенных для обработки последовательностей данных, где важна зависимость между элементами, расположенными в разное время. В отличие от традиционных нейронных сетей, RNN обладают внутренней памятью, что позволяет им учитывать предыдущие состояния при обработке текущего элемента последовательности.

Основная идея RNN заключается в наличии циклов в структуре сети, которые позволяют информации сохраняться между шагами обработки. Математически это выражается через скрытое состояние ht*ht*, которое обновляется на каждом шаге:

$$h_t = \sigma(W_{hh}h_{t-1} + W_{xh}x_t + b_h)$$

где:

- *xt* — входной вектор на шаге t*t*;
- *ht* — скрытое состояние на шаге t*t*;
- *Whh*,*Wxh* — матрицы весов;
- *bh* — вектор смещения;
- *σ* — функция активации (обычно гиперболический тангенс или ReLU).

Однако классические RNN страдают от проблемы затухающего или взрывающегося градиента, что затрудняет обучение на длинных последовательностях. Для решения этой проблемы были предложены LSTM-сети, которые включают в себя три типа ворот (input, forget, output), управляющих потоком информации:

1. **Забывающий gate** (ft*ft*): решает, какую информацию удалить из состояния памяти.
2. **Входной gate** (it*it*): определяет, какие новые данные сохранить в состоянии памяти.
3. **Выходной gate** (ot*ot*): регулирует, какая информация будет передана на следующий шаг.

Формулы для LSTM:

$$f_t = \sigma(W_f \cdot [h_{t-1}, x_t] + b_f)$$

$$i_t = \sigma(W_i \cdot [h_{t-1}, x_t] + b_i)$$

$$\tilde{C}_t = \tanh(W_C \cdot [h_{t-1}, x_t] + b_C)$$

$$C_t = f_t \cdot C_{t-1} + i_t \cdot \tilde{C}_t$$

$$o_t = \sigma(W_o \cdot [h_{t-1}, x_t] + b_o)$$

$$h_t = o_t \cdot \tanh(C_t)$$

Области применения:

- **Обработка временных рядов:** прогнозирование цен на акции, погоды, спроса на товары.
- **Обработка текстов:** машинный перевод, генерация текста, классификация последовательностей.

Аппаратура и материалы:

1. Компьютер с ОС Windows/Linux.
2. Среда разработки Python (Jupyter Notebook, PyCharm).
3. Библиотеки: TensorFlow/Keras, PyTorch, NumPy, Pandas.

Методика и порядок выполнения работы:

1. **Подготовка данных:**
   - Для временных рядов: загрузить данные (например, котировки акций), нормализовать их, разбить на обучающую и тестовую выборки.
   - Для текстов: токенизация, создание словаря, преобразование в последовательности чисел.

2. **Построение модели:**
   - Реализовать RNN и LSTM с использованием Keras/PyTorch. Пример архитектуры:

```
model = Sequential([
    LSTM(64, input_shape=(seq_length, num_features)),
    Dense(1)  # для регрессии
])
```

3. **Обучение модели:**

- Выбрать оптимизатор (Adam, SGD), функцию потерь (MSE для регрессии, Cross-Entropy для классификации).
- Обучить модель на тренировочных данных, контролируя переобучение.

4. **Оценка результатов:**
   - Проверить точность на тестовой выборке.
   - Визуализировать предсказания (для временных рядов).

Содержание отчета:

1. Описание архитектуры RNN/LSTM.
2. Код реализации с комментариями.
3. Графики обучения и предсказаний.
4. Анализ точности и ошибок.

Вопросы для защиты:

1. Чем RNN отличается от полносвязных сетей?
2. Как LSTM решает проблему затухающего градиента?
3. Примеры применения RNN в NLP.
4. Как подготовить текстовые данные для обработки RNN?
5. Какие метрики используются для оценки моделей?

Лабораторная работа №13. Генетические алгоритмы и эволюционные методы. Оптимизация функций. Примеры задач

Цель и содержание: исследовать принципы работы генетических алгоритмов и эволюционных методов, изучить их применение для оптимизации функций, рассмотреть практические примеры задач.

Теоретическое обоснование
Генетические алгоритмы (ГА) — это методы оптимизации, основанные на принципах естественного отбора и генетики. Они моделируют процессы эволюции, такие как наследование, мутация, отбор и скрещивание, для поиска оптимальных решений в сложных пространствах поиска. ГА работают с популяцией возможных решений, которые называются особями или хромосомами. Каждая особь представляет собой закодированное решение задачи, например, в виде битовой строки, вектора чисел или другого представления.

Основные этапы генетического алгоритма:

1. **Инициализация популяции:** создается начальный набор случайных решений.

2. **Оценка приспособленности (fitness function):** для каждой особи вычисляется значение целевой функции, определяющее её качество.
3. **Селекция (отбор):** особи с лучшими значениями fitness имеют больше шансов быть выбранными для размножения. Методы селекции:
    - Рулеточный отбор (вероятность выбора пропорциональна fitness).
    - Турнирный отбор (случайный выбор нескольких особей, лучшая проходит дальше).

4. **Скрещивание (кроссовер):** выбранные особи обмениваются частями своих хромосом, создавая потомков. Примеры операторов:
    - Одноточечный кроссовер: хромосомы разбиваются в случайной точке и обмениваются частями.
    - Двухточечный кроссовер: две точки разрыва, обмен средним сегментом.
    - Равномерный кроссовер: каждый ген выбирается случайно от одного из родителей.

5. **Мутация:** случайное изменение генов потомков для внесения разнообразия. Вероятность мутации обычно мала (0,1–5%).
6. **Формирование новой популяции:** потомки заменяют худших особей предыдущего поколения.
7. **Критерии остановки:** алгоритм завершается при достижении максимального числа поколений, удовлетворительного значения fitness или отсутствия улучшений.

Примеры задач оптимизации:

1. **Минимизация функции:** например, поиск минимума функции f(x)=x2$f(x)=x2$ в диапазоне [−10,10][−10,10].
2. **Задача коммивояжёра:** поиск кратчайшего маршрута через заданные города.
3. **Настройка гиперпараметров моделей машинного обучения.**
4. **Оптимизация формы детали в инженерии.**

Аппаратура и материалы:

1. Компьютер с ОС Windows/Linux.
2. Среда разработки Python (Jupyter Notebook, PyCharm).
3. Библиотеки: DEAP, PyGAD, NumPy, Matplotlib.

Методика и порядок выполнения работы:

1. **Реализация ГА для оптимизации функции:**
    - Выбрать тестовую функцию (например, Растригина или Розенброка).

- Закодировать решение: вещественные числа или битовые строки.
- Определить функцию приспособленности (например, fitness=1/(1+f(x))$fitness=1/(1+f(x))$).
- Настроить параметры ГА: размер популяции, вероятность кроссовера и мутации, число поколений.
- Запустить алгоритм, сохраняя лучшие значения fitness на каждом поколении.

2. **Визуализация результатов:**
   - Построить график сходимости (fitness от номера поколения).
   - Отобразить лучшую особь и её значение.

3. **Анализ влияния параметров:**
   - Исследовать, как изменение вероятности мутации или размера популяции влияет на скорость сходимости.

Содержание отчета:

1. Описание генетического алгоритма и его операторов.
2. Код реализации с комментариями.
3. Графики зависимости fitness от поколения.
4. Анализ результатов и выводы.

Вопросы для защиты:

1. Какие биологические принципы лежат в основе ГА?
2. Как работает оператор кроссовера?
3. Почему важна мутация в ГА?
4. Какие критерии остановки алгоритма вы знаете?
5. Приведите примеры задач, где ГА эффективны.

Лабораторная работа №14. Оценка моделей и кросс-валидация. Метрики качества (Accuracy, Precision, Recall, F1). Разбиение данных.

Цель и содержание: изучить методы оценки качества моделей машинного обучения, освоить технику кросс-валидации, научиться рассчитывать основные метрики классификации и правильно разбивать данные на обучающую и тестовую выборки.

Теоретическое обоснование
Оценка качества моделей машинного обучения является критически важным этапом в процессе их разработки. Для получения достоверных результатов необходимо правильно разделять исходные данные и использовать соответствующие метрики оценки. Основные подходы к разбиению данных включают: простое разделение на обучающую и тестовую выборки (обычно в

пропорции 70/30 или 80/20), k-кратную кросс-валидацию и стратифицированную выборку. Кросс-валидация, особенно k-кратная, позволяет более эффективно использовать имеющиеся данные, уменьшая зависимость результатов от конкретного разбиения.

Основные метрики качества для задач классификации:

1. **Accuracy (точность)** - доля правильно классифицированных объектов среди всех объектов:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

   где TP - истинно положительные, TN - истинно отрицательные, FP - ложно положительные, FN - ложно отрицательные случаи.

2. **Precision (точность положительного прогноза)** - доля истинно положительных случаев среди всех случаев, которые модель отнесла к положительному классу:

$$Precision = \frac{TP}{TP + FP}$$

3. **Recall (полнота)** - доля истинно положительных случаев, которые модель правильно идентифицировала среди всех фактически положительных случаев:

$$Recall = \frac{TP}{TP + FN}$$

4. **F1-мера** - гармоническое среднее между precision и recall:

$$F1 = 2 \times \frac{Precision \times Recall}{Precision + Recall}$$

Для задач регрессии используются другие метрики: средняя абсолютная ошибка (MAE), среднеквадратичная ошибка (MSE), коэффициент детерминации (R^2). Выбор метрик зависит от конкретной задачи и требований к модели.

Аппаратура и материалы:

1. Компьютер с установленной средой разработки (Python, Jupyter Notebook)
2. Библиотеки: scikit-learn, pandas, numpy, matplotlib
3. Наборы данных для классификации (например, Iris, Titanic)

Методика и порядок выполнения работы:

1. Подготовка данных:
    - Загрузить выбранный набор данных
    - Провести предварительную обработку (заполнение пропусков, кодирование категориальных признаков)
    - Нормализовать числовые признаки при необходимости

2. Реализация различных методов разбиения данных:
    - Простое разбиение с помощью train_test_split
    - KFold кросс-валидация (k=5, k=10)
    - Стратифицированная кросс-валидация для несбалансированных данных

3. Обучение моделей:
    - Выбрать несколько алгоритмов классификации (например, логистическая регрессия, случайный лес, SVM)
    - Обучить модели на различных вариантах разбиения данных

4. Оценка качества:
    - Рассчитать метрики для каждого варианта
    - Сравнить результаты между собой
    - Построить матрицы ошибок (confusion matrix)
    - Визуализировать результаты

5. Анализ влияния разбиения:
    - Сравнить стабильность метрик при разных способах разбиения
    - Оценить дисперсию результатов при кросс-валидации

Содержание отчета:

1. Описание использованных методов разбиения данных
2. Формулы и описание рассчитанных метрик
3. Код реализации с комментариями
4. Таблицы и графики с результатами
5. Сравнительный анализ разных методов оценки
6. Выводы о влиянии способа разбиения на оценку качества моделей

Вопросы для защиты работы:

1. В чем преимущества кросс-валидации перед простым разбиением?
2. Когда следует использовать стратифицированную выборку?
3. В каких случаях accuracy может быть плохой метрикой?
4. Как интерпретировать F1-меру?
5. Почему важно оценивать модели на тестовой выборке?
6. Какие метрики следует использовать для несбалансированных данных?
7. Как выбрать оптимальное k для k-кратной кросс-валидации?

Список литературы

Основная литература:

1. Новиков, Ф. А. Символический искусственный интеллект: математические основы представления знаний : учебник для вузов / Ф. А. Новиков. — Москва : Издательство Юрайт, 2025. — 278 с. — (Высшее образование). — ISBN 978-5-534-00734-3. Платонов, А. В. Машинное обучение : учебное пособие для вузов / А. В. Платонов. — 2-е изд. — Москва : Издательство Юрайт, 2025. — 89 с. — (Высшее образование). — ISBN 978-5-534-20732-3.
2. Иванов, В. М. Интеллектуальные системы : учебное пособие для вузов / В. М. Иванов ; под научной редакцией А. Н. Сесекина. — Москва : Издательство Юрайт, 2025. — 88 с. — (Высшее образование). — ISBN 978-5-534-20851-1.
3. Тропин, М. П. Математическая обработка информации : учебное пособие для вузов / М. П. Тропин. — Москва : Издательство Юрайт, 2025. — 151 с. — (Высшее образование). — ISBN 978-5-534-20557-2.

Дополнительной литература:

1. Основы технологий искусственного интеллекта : учебное пособие / А. В. Кревецкий, Ю. А. Ипатов, Н. И. Роженцова ; под общ. ред. А. В. Кревецкий, Поволжский государственный технологический университет. - Йошкар-Ола : Поволжский государственный технологический университет, 2023. - 272 с. - ISBN 978-5-8158-2358-7.
2. Символический искусственный интеллект: математические основы представления знаний : учеб. пособие для академического бакалавриата / Новиков Ф. А. - М. : Юрайт, 2017. - 277 с. : ил., портр. - (Бакалавр. Академический курс. Модуль). - Библиогр.: с. 273. - ISBN 978-5-534-00734-3.