Министерство образования и науки РФ

Федеральное государственное автономное

образовательное учреждение высшего образования

«Санкт-Петербургский национальный исследовательский университет

информационных технологий, механики и оптики»

**факультет программной инженерии и компьютерной техники**

**ОТЧЁТ ПО МОДУЛЮ №2**

по дисциплине

‘Системы искусственного интеллекта’

*Выполнил:*

Студент группы P33111

Павлов Александр Сергеевич

*Преподаватель:*

Авдюшина А.Е.

Изображение выглядит как Шрифт, логотип, Графика, белый

Автоматически созданное описание

Санкт-Петербург, 2023

# Лабораторная работа 1. Метод линейной регрессии

## Введение

**Задание:**

* Выбор датасетов:
  + Студенты с **четным** порядковым номером в группе должны использовать набор данных о [жилье в Калифорнии](https://developers.google.com/machine-learning/crash-course/california-housing-data-description?hl=ru) Скачать [тут](https://download.mlcc.google.com/mledu-datasets/california_housing_train.csv)
  + Студенты с **нечетным** порядковым номером в группе должны использовать [про обучение студентов](https://www.kaggle.com/datasets/nikhil7280/student-performance-multiple-linear-regression)
* Получите и визуализируйте (графически) статистику по датасету (включая количество, среднее значение, стандартное отклонение, минимум, максимум и различные квантили).
* Проведите предварительную обработку данных, включая обработку отсутствующих значений, кодирование категориальных признаков и нормировка.
* Разделите данные на обучающий и тестовый наборы данных.
* Реализуйте линейную регрессию с использованием метода наименьших квадратов без использования сторонних библиотек, кроме NumPy и Pandas (для использования коэффициентов использовать библиотеки тоже нельзя). Использовать минимизацию суммы квадратов разностей между фактическими и предсказанными значениями для нахождения оптимальных коэффициентов.
* Постройте **три модели** с различными наборами признаков.
* Для каждой модели проведите оценку производительности, используя метрику коэффициент детерминации, чтобы измерить, насколько хорошо модель соответствует данным.
* Сравните результаты трех моделей и сделайте выводы о том, какие признаки работают лучше всего для каждой модели.
* Бонусное задание
  + Ввести синтетический признак при построении модели

## Описание метода

Метод линейной регрессии — это статистический алгоритм, используемый для оценки связи между зависимой переменной и одной или несколькими независимыми переменными.

Принцип работы метода линейной регрессии основан на поиске наилучшей прямой, которая наиболее точно соответствует наблюдаемым данным. Эта линия называется линией регрессии.

Для построения линейной регрессии используется метод наименьших квадратов, который минимизирует сумму квадратов разностей между наблюдаемыми значениями зависимой переменной и предсказанными значениями, полученными по линии регрессии.

## Псевдокод метода

class LinearRegression():

  def \_\_init\_\_(self):

    self.coefficients = None

  def determination\_coefficient(self, y, y\_pred) -> float:

    sse = np.sum(np.square(y - y\_pred))

    sst = np.sum(np.square(y - np.mean(y)))

    return 1 - (sse / sst)

  def fit(self, X, y) -> None:

    ones\_column = np.ones((X.shape[0], 1))

    X = np.concatenate((ones\_column, X), axis=1)

    X\_transpose\_X = X.T @ X

    X\_transpose\_y = X.T @ y

    coefficients = np.linalg.inv(X\_transpose\_X) @ X\_transpose\_y

    self.coefficients = coefficients

  def predict(self, X) -> list[float]:

    ones\_column = np.ones((X.shape[0], 1))

    X = np.concatenate((ones\_column, X), axis=1)

    predictions = X @ self.coefficients

    return predictions

## Результаты выполнения

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт, документ

Автоматически созданное описание

## Примеры использования метода

1️. Финансовый анализ: Метод линейной регрессии может использоваться для прогнозирования цен на акции, основываясь на различных финансовых показателях, таких как объем продаж, прибыль компании и т. д. Этот метод выбирается, потому что он может обнаружить линейные связи между финансовыми показателями и ценами на акции.

2. Медицинские исследования: Линейная регрессия может помочь в определении связи между риском заболевания и различными факторами, такими как возраст, пол, образ жизни и показатели здоровья. Этот метод выбирается, потому что он позволяет выявить тенденции и предсказать риск заболевания на основе указанных факторов.

# Лабораторная работа 2. Метод k-ближайших соседей (k-NN)

## Введение

**Задание:**

Выбор датасета:

Четный номер в группе - Датасет [о вине](https://www.kaggle.com/datasets/davorbudimir/winedataset)

Нечетный номер в группе - Датасет [про диабет](https://www.kaggle.com/datasets/abdallamahgoub/diabetes/data)

* Проведите предварительную обработку данных, включая обработку отсутствующих значений, кодирование категориальных признаков и масштабирование.
* Получите и визуализируйте (графически) статистику по датасету (включая количество, среднее значение, стандартное отклонение, минимум, максимум и различные квантили), постройте 3d-визуализацию признаков.
* Реализуйте метод k-ближайших соседей \*\*\*\*без использования сторонних библиотек, кроме NumPy и Pandas.
* Постройте две модели k-NN с различными наборами признаков:
  + Модель 1: Признаки случайно отбираются .
  + Модель 2: Фиксированный набор признаков, который выбирается заранее.
* Для каждой модели проведите оценку на тестовом наборе данных при разных значениях k. Выберите несколько различных значений k, например, k=3, k=5, k=10, и т. д. Постройте матрицу ошибок.

## Описание метода

Метод k-ближайших соседей (k-nearest neighbors, k-NN) - это алгоритм классификации и регрессии. Его назначение заключается в определении класса (в случае классификации) или значения целевой переменной (в случае регрессии) нового объекта на основе сходства с его ближайшими соседями в обучающей выборке. Принцип работы метода k-ближайших соседей:

1️. Задается значение k, которое определяет количество ближайших соседей, которые будут использоваться для классификации или регрессии.

2️. Измеряется расстояние между новым объектом и всеми объектами в обучающей выборке, используя некоторую меру.

3️. Выбираются k объектов с наименьшим расстоянием до нового объекта.

4️. Для классификации подсчитывается, какие классы наиболее часто встречаются среди выбранных соседей, и новый объект присваивается наиболее часто встречающемуся классу.

5️. Для регрессии вычисляется среднее (или медианное) значение целевой переменной выбранных соседей, которое становится предсказанным значением для нового объекта.

Преимущества метода k-ближайших соседей включают простоту реализации, адаптацию к различным типам данных.

## Псевдокод метода

from collections import Counter

def euclidean\_distance(point1, point2):

    return np.sqrt(np.sum((point1 - point2) \*\* 2))

class KNN:

  def \_\_init\_\_(self, k):

    self.k = k

  def fit(self, X, y):

    self.X\_train = np.array(X)

    self.y\_train = np.array(y)

  def predict(self, X):

    return [self.\_predict(x) for x in np.array(X)]

  def \_predict(self, X):

    distances = [euclidean\_distance(X, x\_train) for x\_train in self.X\_train]

    k\_indices = np.argsort(distances)[:self.k]

    k\_nearest\_labels = [self.y\_train[i] for i in k\_indices]

    return Counter(k\_nearest\_labels).most\_common()[0][0]

  def accuracy(self, y, y\_pred):

    return np.sum(y == y\_pred) / len(y)

## Результаты выполнения

Изображение выглядит как текст, Шрифт, снимок экрана

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, диаграмма, Красочность

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, диаграмма, Прямоугольник

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, диаграмма, Прямоугольник

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, диаграмма, Красочность

Автоматически созданное описание

## Примеры использования метода

1️. Классификация текстов: Метод k-ближайших соседей может использоваться для классификации текстовых документов по их содержимому. Он выбирается, потому что основывается на схожести между текстами, используя меру расстояния, такую как косинусное расстояние. Это позволяет определить, какие документы могут быть похожими и отнести новый текст к определенной категории или классу.

2️. Рекомендательные системы: Метод k-ближайших соседей может быть применен для рекомендации товаров или контента пользователю на основе его сходства с другими пользователями. Он выбирается, потому что использует информацию о предпочтениях и поведении других пользователей, чтобы найти наиболее похожих соседей и рекомендовать товары или контент, которые вероятно заинтересуют данного пользователя.

# Лабораторная работа 3. Деревья решений

## Введение

**Задание**

1. Для студентов с четным порядковым номером в группе – датасет с [классификацией грибов](https://archive.ics.uci.edu/ml/datasets/Mushroom), а нечетным – [датасет с данными про оценки студентов инженерного и педагогического факультетов](https://archive.ics.uci.edu/dataset/856/higher+education+students+performance+evaluation) (для данного датасета нужно ввести метрику: студент успешный/неуспешный на основании грейда)
2. Отобрать **случайным** образом sqrt(n) признаков
3. Реализовать без использования сторонних библиотек построение дерева решений (дерево не бинарное, numpy и pandas использовать можно, использовать списки для реализации дерева - нельзя)
4. Провести оценку реализованного алгоритма с использованием Accuracy, precision и recall
5. Построить AUC-ROC и AUC-PR (в пунктах 4 и 5 использовать библиотеки нельзя)

## Описание метода

Метод деревьев решений (decision trees) — это алгоритм машинного обучения, используемый для решения задач классификации и регрессии. Его назначение заключается в построении модели, которая принимает решение путем последовательного задания вопросов о значениях признаков и, в зависимости от ответов, переходит к следующему вопросу или делает прогноз.

Принцип работы метода деревьев решений:

1️. Вначале выбирается признак, который лучше всего разделяет обучающую выборку на классы (в случае классификации) или уменьшает среднеквадратичную ошибку (в случае регрессии).

2️. На основе выбранного признака создается корневой узел дерева.

3️. Для каждого значения выбранного признака создаются дочерние узлы, и обучающая выборка разбивается на подвыборки соответствующие значениям признака.

4️. Шаги 1–3 рекурсивно повторяются для каждого дочернего узла до выполнения одного из критериев останова, например достижения максимальной глубины дерева или минимального количества объектов в узле.

5️. В листовых узлах дерева, которые не разветвляются далее, принимается окончательное решение или делается прогноз.

## Псевдокод метода

class Node():

  def \_\_init\_\_(self, feature = None, threshold = None, left = None, right = None, value = None):

    self.feature = feature

    self.threshold = threshold

    self.left = left

    self.right = right

    self.value = value

  def is\_leaf\_node(self):

    return self.value is not None

class DecisionTree():

  def \_\_init\_\_(self):

      self.tree = None

  def fit(self, X, y):

      self.tree = self.grow\_tree(X, y)

  def predict(self, X):

      return np.array([self.travers\_tree(x, self.tree) for \_, x in X.iterrows()])

  def entropy(self, y):

      hist = np.bincount(y)

      ps = hist / len(y)

      return -np.sum([p \* np.log2(p) for p in ps if p > 0])

  def most\_common(self, y):

      labels = np.unique(y)

      count = [list(y).count(i) for i in labels]

      return labels[np.argmax(count)]

  def best\_split(self, X, y):

      best\_feature, best\_threshold = None, None

      best\_gain = -1

      for i in range(X.shape[1]):

          thresholds = np.unique(X.iloc[:, i])

          for threshold in thresholds:

              gain = self.information\_gain(X.iloc[:, i], y, threshold)

              if gain > best\_gain:

                  best\_gain = gain

                  best\_feature = i

                  best\_threshold = threshold

      return best\_feature, best\_threshold

  def information\_gain(self, X\_column, y, threshold):

      if len(np.unique(y)) == 1:

          return 0

      n = len(y)

      left\_indexes = np.argwhere(X\_column.values <= threshold).flatten()

      right\_indexes = np.argwhere(X\_column.values > threshold).flatten()

      e\_l, n\_l = self.entropy(y.iloc[left\_indexes]), len(left\_indexes)

      e\_r, n\_r = self.entropy(y.iloc[right\_indexes]), len(right\_indexes)

      child = (n\_l / n) \* e\_l + (n\_r / n) \* e\_r

      return self.entropy(y) - child

  def grow\_tree(self, X, y):

      n\_samples, n\_features = X.shape

      n\_labels = len(np.unique(y))

      if n\_labels == 1:

          return Node(value=self.most\_common(y))

      best\_feature, best\_threshold = self.best\_split(X, y)

      left\_indexes = np.argwhere(X.iloc[:, best\_feature].values <= best\_threshold).flatten()

      right\_indexes = np.argwhere(X.iloc[:, best\_feature].values > best\_threshold).flatten()

      if len(left\_indexes) == 0 or len(right\_indexes) == 0:

          return Node(value=self.most\_common(y))

      left = self.grow\_tree(X.iloc[left\_indexes, :],

                            y.iloc[left\_indexes])

      right = self.grow\_tree(X.iloc[right\_indexes, :], y.iloc[right\_indexes])

      return Node(best\_feature, best\_threshold, left, right)

  def travers\_tree(self, x, tree):

      if tree.is\_leaf\_node():

          return tree.value

      if x[tree.feature] <= tree.threshold:

          return self.travers\_tree(x, tree.left)

      return self.travers\_tree(x, tree.right)

  def confusion\_matrix(self, y\_true, y\_pred):

    res = np.zeros((2, 2))

    for pred, true in zip(y\_pred, y\_true):

      if pred == 1 and true == 1:

          res[0, 0] += 1

      elif pred == 1 and true == 0:

          res[0, 1] += 1

      elif pred == 0 and true == 1:

          res[1, 0] += 1

      elif pred == 0 and true == 0:

          res[1, 1] += 1

    return res

  def confusion\_matrix\_auc(self, y\_true, y\_pred, threshold):

    res = np.zeros((2, 2))

    for pred, true in zip(y\_pred, y\_true):

      pred = 1 if pred >= threshold else 0

      if pred == 1 and true == 1:

          res[0, 0] += 1

      elif pred == 1 and true == 0:

          res[0, 1] += 1

      elif pred == 0 and true == 1:

          res[1, 0] += 1

      elif pred == 0 and true == 0:

          res[1, 1] += 1

    return res

  def accuracy(self, confusion\_matrix):

    return (confusion\_matrix[0, 0] + confusion\_matrix[1, 1]) / (confusion\_matrix[0, 0] + confusion\_matrix[1, 1] + confusion\_matrix[0, 1] + confusion\_matrix[1, 0])

  def precision(self, confusion\_matrix):

    return confusion\_matrix[0, 0] / (confusion\_matrix[0, 0] + confusion\_matrix[0, 1])

  def recall(self, confusion\_matrix):

    return confusion\_matrix[0, 0] / (confusion\_matrix[0, 0] + confusion\_matrix[1, 0])

  def fallout(self, confusion\_matrix):

    return confusion\_matrix[0, 1] / (confusion\_matrix[1, 1] + confusion\_matrix[0, 1])

## Результаты выполнения

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, линия, диаграмма

Автоматически созданное описание

## Примеры использования метода

1️. Кредитный скоринг: Банкам часто требуется оценить кредитную надежность заемщиков. Деревья решений могут использоваться для создания модели, которая анализирует различные признаки заемщиков (например, доход, историю платежей) и предсказывает, вероятно ли, что они вернут кредит. Деревья решений позволяют интерпретировать факторы, которые влияют на принятие решения, их предсказательную силу и легко понять логику принятия решения.

2. Медицинская диагностика: В медицинской сфере деревья решений могут использоваться для поддержки принятия решений при постановке диагноза или определении лечебной стратегии. Он может анализировать медицинские показатели пациента, симптомы и историю заболевания для определения правильного диагноза или рекомендации дальнейших действий. Важно отметить, что деревья решений могут предоставлять прозрачное объяснение, почему определенное решение было принято, что особенно ценно в медицинской сфере.

# Лабораторная работа 4. Логистическая регрессия

## Введение

**Задание:**

1. Выбор датасета:

* Датасет о пассажирах Титаника: [Titanic Dataset](https://www.kaggle.com/c/titanic)
* Датасет о диабете: [Diabetes Dataset](https://www.kaggle.com/uciml/pima-indians-diabetes-database)
* Загрузите выбранный датасет и выполните предварительную обработку данных.
* Получите и визуализируйте (графически) статистику по датасету (включая количество, среднее значение, стандартное отклонение, минимум, максимум и различные квантили).
* Разделите данные на обучающий и тестовый наборы в соотношении, которое вы считаете подходящим.
* Реализуйте логистическую регрессию "с нуля" без использования сторонних библиотек, кроме NumPy и Pandas. Ваша реализация логистической регрессии должна включать в себя:
  + Функцию для вычисления гипотезы (sigmoid function).
  + Функцию для вычисления функции потерь (log loss).
  + Метод обучения, который включает в себя градиентный спуск.
  + Возможность варьировать гиперпараметры, такие как коэффициент обучения (learning rate) и количество итераций.

1. Исследование гиперпараметров:
   * Проведите исследование влияния гиперпараметров на производительность модели. Варьируйте следующие гиперпараметры:
     + Коэффициент обучения (learning rate).
     + Количество итераций обучения.
     + Метод оптимизации (например, градиентный спуск или оптимизация Ньютона).
2. Оценка модели:
   * Для каждой комбинации гиперпараметров оцените производительность модели на тестовом наборе данных, используя метрики, такие как accuracy, precision, recall и F1-Score.

Сделайте выводы о том, какие значения гиперпараметров наилучшим образом работают для данного набора данных и задачи классификации. Обратите внимание на изменение производительности модели при варьировании гиперпараметров.

## Описание метода

Метод логистической регрессии является статистическим алгоритмом машинного обучения, который используется для решения задач классификации. Логистическая регрессия широко применяется для решения задач бинарной классификации, где требуется разделение объектов на два класса на основе определенных признаков. Например, определение, является ли электронное письмо спамом или неспамом. Принцип работы логистической регрессии:

1️. Логистическая регрессия использует линейную комбинацию признаков и их весов, суммируя их значения с помощью линейной функции.

2️. Затем применяется логистическая функция (сигмоида), которая преобразует линейную комбинацию в вероятность. Вероятность показывает, насколько объект принадлежит определенному классу.

3️. Обучение модели происходит путем настройки весов признаков с использованием алгоритма градиентного спуска. Цель состоит в минимизации функции потерь, которая измеряет расхождение между предсказанными вероятностями и фактическими классами в обучающих данных.

4️. Когда модель обучена, она может использоваться для предсказания класса новых объектов, основываясь на вероятности, полученной от логистической функции. Логистическая регрессия обладает рядом преимуществ, таких как эффективность и вычислительная скорость, простота интерпретации результатов и способность обрабатывать большие объемы данных. Однако она хорошо работает только для линейно разделимых классов и может быть уязвима к выбросам в данных.

## Псевдокод метода

class LogisticRegression:

    def \_\_init\_\_(self, w, b):

      self.w = w

      self.b = b

    def fit(self, X, y, learning\_rate=0.005, epochs=40):

      self.losses = []

      self.m, self.n = X.shape

      for epoch in range(epochs):

        Z = X.values.reshape(self.m, self.n).dot(self.w) + self.b

        A = self.sigmoid(Z)

        dw = np.sum(X.values.reshape(self.m, self.n) \* (A.reshape(self.m, 1) - y.values.reshape(self.m, 1)), axis = 0) / len(X)

        db = np.sum((A.reshape(self.m, 1) - y.values.reshape(self.m, 1)), axis=0) / len(X)

        self.w = self.w - learning\_rate \* dw.reshape(self.n, 1)

        self.b = self.b - learning\_rate \* db

        # self.w = self.w - learning\_rate \* self.gr\_log\_loss(X, y)

        # self.losses.append(self.log\_loss(y, self.predict\_(X))) #Вычисление функции потерь

    # def gr\_log\_loss(self, X, y):

    #   return X.T.dot(self.predict\_(X) - y).values

    def predict\_(self, X):

      return np.array([self.sigmoid(x.reshape(1, self.n).dot(self.w) + self.b)[0][0]

                         for x in np.array(X)])

    def predict(self, X):

      y\_class = self.predict\_(X)

      return np.where(y\_class >= 0.5, 1, 0)

    def log\_loss(self, y\_true, y\_pred):

      y\_pred = np.clip(y\_pred, 1e-10, 1 - 1e-10)

      return -np.sum(y\_true \* np.log(y\_pred) + (1 - y\_true) \* np.log(1 - y\_pred), axis = 0) / len(y\_true)

    def sigmoid(self, x):

      return 1 / (1 + np.exp(-x))

    def confusion\_matrix(self, y\_true, y\_pred):

      res = np.zeros((2, 2))

      for pred, true in zip(y\_pred, y\_true):

        if pred == 1 and true == 1:

            res[0, 0] += 1

        elif pred == 1 and true == 0:

            res[0, 1] += 1

        elif pred == 0 and true == 1:

            res[1, 0] += 1

        elif pred == 0 and true == 0:

            res[1, 1] += 1

      return res

    def accuracy(self, confusion\_matrix):

      return (confusion\_matrix[0, 0] + confusion\_matrix[1, 1]) / (confusion\_matrix[0, 0] + confusion\_matrix[1, 1] + confusion\_matrix[0, 1] + confusion\_matrix[1, 0])

    def precision(self, confusion\_matrix):

      return confusion\_matrix[0, 0] / (confusion\_matrix[0, 0] + confusion\_matrix[0, 1])

    def recall(self, confusion\_matrix):

      return confusion\_matrix[0, 0] / (confusion\_matrix[0, 0] + confusion\_matrix[1, 0])

    def f1\_score(self, confusion\_matrix):

      return 2 \* self.precision(confusion\_matrix) \* self.recall(confusion\_matrix) / (self.precision(confusion\_matrix) + self.recall(confusion\_matrix))

## Результаты выполнения

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, программное обеспечение, число

Автоматически созданное описание

Изображение выглядит как текст, снимок экрана, Шрифт

Автоматически созданное описание

## Примеры использования метода

1️. Классификация электронных писем: Логистическая регрессия часто используется для определения, является ли электронное письмо спамом или не спамом. Она может анализировать различные признаки письма, такие как слова, фразы, заголовки и прочие характеристики, и предсказывать вероятность того, что письмо является спамом. Этот метод выбран, потому что он может обрабатывать большие объемы данных и быстро принимать решение на основе вероятностей.

2. Прогнозирование оттока клиентов: Логистическая регрессия может быть применена для прогнозирования вероятности оттока клиентов из компании на основе различных факторов, таких как длительность пользования услугами, история покупок, активность в сети и другие. Этот метод выбран, потому что он позволяет оценивать влияние различных факторов на вероятность оттока и принимать меры по удержанию клиентов.

Основная причина выбора метода логистической регрессии заключается в его способности обрабатывать вероятности и принимать решения на основе линейных комбинаций признаков. Он имеет простую интерпретацию результатов и может быть эффективно обучен на больших объемах данных.

# Сравнение методов

## Сравнительный анализ методов

Линейная регрессия:

Преимущества:

1. Простота интерпретации: Линейная регрессия легко интерпретируется и позволяет определить влияние каждого признака на результат.

2. Высокая производительность: Линейная регрессия обучается быстро и хорошо масштабируется для больших объемов данных.

Ограничения:

1. Линейная зависимость: Линейная регрессия не может моделировать сложные нелинейные взаимосвязи между признаками и результатом.
2. Чувствительность к выбросам: Линейная регрессия может быть чувствительна к выбросам в данных, что может искажать результаты.

Метод k-ближайших соседей (k-NN):

Преимущества:

1. Простота реализации: k-NN прост в реализации и понятен для понимания.
2. Адаптивность к нелинейным взаимосвязям: k-NN может моделировать сложные нелинейные отношения между признаками и результатом.

Ограничения:

1. Низкая производительность: в отличие от линейной регрессии, метод k-NN требует вычисления расстояний между объектами, что может быть вычислительно затратным на больших данных.
2. Неэффективность на данных большой размерности: Признаковое пространство большой размерности может привести к "разреженности" данных, что влияет на производительность метода k-NN.

Дерево решений:

Преимущества:

1. Хорошая интерпретируемость: Деревья решений можно легко понять и интерпретировать.
2. Универсальность: Деревья решений могут работать с категориальными и числовыми данными.

Ограничения:

1. Склонность к переобучению: Деревья решений могут быть склонны к переобучению на тренировочных данных.
2. Нестабильность: Малые изменения в данных могут привести к значительным изменениям в построенном дереве решений.

Логистическая регрессия:

Преимущества:

1. Интерпретируемость: Логистическая регрессия предоставляет интерпретируемые коэффициенты, которые выявляют влияние признаков на результат.
2. Вероятностная интерпретация: Логистическая регрессия моделирует вероятность принадлежности к классу, что полезно для классификации.

Ограничения:

1. Линейная разделимость: Логистическая регрессия может иметь ограничения в моделировании сложных нелинейных зависимостей между признаками и результатом.
2. Чувствительность к выбросам: Логистическая регрессия может быть чувствительна к выбросам в данных и дать искаженные результаты.

В итоге выбор метода зависит от конкретной задачи и свойств данных. Линейная регрессия хороша для моделирования линейных отношений, k-NN подходит для нелинейных связей и малого числа признаков, деревья решений пригодны для интерпретации, а логистическая регрессия хороша для вероятностного анализа и классификации.

## Примеры лучшего использования каждого метода

Линейная регрессия:

1. Когда взаимосвязи между признаками и результатом являются примерно линейными. В таких случаях линейная регрессия может достаточно точно моделировать отношения.
2. Когда требуется интерпретируемость модели и понимание влияния каждого признака на результат.
3. Когда объем данных велик, и требуется быстрая обучаемость модели.

Метод k-ближайших соседей:

1. Когда в данных присутствуют сложные нелинейные взаимосвязи между признаками и результатом. Метод k-NN способен улавливать такие закономерности.
2. Когда данных немного или признаковое пространство небольшой размерности. В таких случаях метод k-NN может быть быстрым и точным.

Дерево решений:

1. Когда требуется понимание и интерпретируемость модели. Дерево решений предоставляет простую и понятную визуализацию решений.
2. Когда в данных присутствуют как линейные, так и нелинейные взаимосвязи между признаками и результатом.
3. Когда данные содержат как числовые, так и категориальные признаки. Дерево решений может работать с различными типами данных.

Логистическая регрессия:

1. Когда задача связана с бинарной классификацией. Логистическая регрессия моделирует вероятности принадлежности к классу.
2. Когда данные линейно разделимы или близки к линейно разделимым.

# Заключение

В ходе выполнения данных лабораторных работ я познакомился с такими моделями машинного обучения, как линейная регрессия, k-ближайших соседей, дерево решений и логистическая регрессия. В итоге, каждый из этих методов имеет свои преимущества и ограничения, и выбор наиболее эффективного зависит от специфики данных и задачи.

# Приложения

Ссылка на google colab: <https://colab.research.google.com/drive/1mn1-KIaAlP5v8M_dE5oyd2bfNs-S0RnH?usp=sharing>