



МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ
ФЕДЕРАЦИИ

**ФЕДЕРАЛЬНОЕ ГОСУДАРСТВЕННОЕ БЮДЖЕТНОЕ
ОБРАЗОВАТЕЛЬНОЕ УЧРЕЖДЕНИЕ ВЫСШЕГО ОБРАЗОВАНИЯ
«ДОНСКОЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»
(ДГТУ)**

Кафедра «Информационные системы в строительстве»

МЕТОДИЧЕСКИЕ УКАЗАНИЯ
по дисциплине

МЕТОДЫ И АЛГОРИТМЫ МАШИННОГО ОБУЧЕНИЯ

Ростов-на-Дону
ДГТУ
2025

Составитель: А.А. Ляпин

Методические указания для выполнения контрольной работы по дисциплине «Методы и алгоритмы машинного обучения» для обучающихся заочной формы / сост. А.А. Ляпин – Ростов-на-Дону: Донской государственный технический университет, 2025. – 30 с.

Содержат краткую теорию по изучаемым вопросам, примеры применения методов машинного обучения для решения задач классификации, кластеризации и регрессионного анализа. Приведены варианты индивидуальных заданий.

Предназначены для обучающихся магистратуры направления подготовки 09.04.02 заочной формы обучения

УДК 004.02

Ответственный за выпуск: зав. кафедрой «Информационные системы в строительстве» д-р ф.-м. наук, профессор А.А. Ляпин

Издательский центр ДГТУ
Адрес университета и полиграфического предприятия:
344000, г. Ростов-на-Дону, пл. Гагарина, 1

© Донской государственный технический университет, 2025

СОДЕРЖАНИЕ

1. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КОНТРОЛЬНОЙ РАБОТЫ.....	4
2. ПОДГОТОВКА ДАННЫХ ДЛЯ МАШИННОГО ОБУЧЕНИЯ	5
3. МЕТОД ОПОРНЫХ ВЕКТОРОВ.....	13
4. КЛАСТЕРИЗАЦИЯ	18
5. РЕГРЕССИЯ	23
6. ОЦЕНКА ПОЛОЖЕНИЯ ТЕЛА ЧЕЛОВЕКА С ПОМОЩЬЮ НЕЙРОННОЙ СЕТИ	26

1. МЕТОДИЧЕСКИЕ УКАЗАНИЯ ПО ВЫПОЛНЕНИЮ КОНТРОЛЬНОЙ РАБОТЫ

Задания контрольной работы выполняются по индивидуальным вариантам. Вариант соответствует последней цифре в зачетке обучающегося: цифра 1 – вариант 1, ..., цифра 0 – вариант 10.

Отчет по контрольной работе оформляется в виде электронного документа MS Word. Содержит:

- титульный лист;
- постановку задачи, незначительные сведения из теории, код программы или скриншоты выполнения в рекомендованной программе на компьютере, результаты вычислений и сформулированные выводы по каждой задаче;
- список использованной литературы, включая интернет-источники.

Отчет распечатывается и предоставляется на кафедру «Кибербезопасность информационных систем» для проверки и оценивания.

2. ПОДГОТОВКА ДАННЫХ ДЛЯ МАШИННОГО ОБУЧЕНИЯ

Перед обучением модели на данных важно учитывать несколько ключевых аспектов:

1. Качество данных: чем точнее используемые данные, тем лучших результатов можно ожидать.
2. Предварительная обработка данных: Иногда необходимо выполнить предварительную обработку данных, например, удаление выбросов, масштабирование или нормализацию признаков. Это может помочь улучшить процесс обучения и результаты модели.
3. Выбор признаков: Определите, какие признаки данных будут использоваться для обучения модели.

Соответствующие и информативные знаки помогут модели принимать правильные решения.

4. Разделение на обучающую и тестовую выборки: Важно разделить данные на две непересекающиеся группы — обучающую и тестовую выборки. Обучающая выборка используется для обучения модели, а тестовая — для оценки ее обобщающей способности.
5. Управление переобучением: Переобучение может возникнуть, когда модель «запоминает» обучающие данные и плохо обобщает результаты на новые данные. Для решения этой проблемы используйте методы регуляризации, такие как дропаут-регуляризация.

Обработаем набор данных "Опрос о психическом здоровье в сфере технологий" из Kaggle.

```
import pandas as pd
import numpy as np
df = pd.read_csv("survey.csv")
df.head()
```

Названия признаков набора данных:

- Отметка времени (время)

- Возраст (возраст)
- Пол (гендер)
- Страна страна)
- Штат (штат)
- self_employed (самозанятый)
- family_history: Есть ли в вашей семье случаи психических заболеваний?
- лечение: Обращались ли вы за лечением по поводу психического расстройства?

...

1. Определение характеристик набора данных

```
df.shape
```

```
df.info() # отобразит информацию
```

Визуализация пропущенных значений

```
import seaborn as sns
```

```
sns.heatmap(df.isnull(), yticklabels=False, cbar=False, cmap='viridis')
```

Далее мы представим статистические данные по числовым значениям признаков: количество, среднее значение, стандартное отклонение, минимум, 25%-50%-75% процентиль, максимум.

```
df.describe()
```

Количество уникальных значений для каждого столбца

```
df.nunique()
```

Количество уникальных значений для каждого столбца в наборе данных.

```
feature_names = df.columns.tolist()
for column in feature_names:
    print(column)
    print("-----")
    print(df[column].value_counts(dropna=False))
    print("=====")
```

2. Выбор целевой переменной

Выбор целевой переменной зависит от решаемой задачи. Например, мы решаем задачу классификации: будет ли применяться лечение к респонденту?

```
features = df.drop('treatment',1) labels =  
df['treatment'] print(features.head())
```

3. Очистка набора данных

В столбце «возраст» содержатся данные о людях, которые еще не родились (отрицательные числа). В столбце «возраст» содержатся данные о детях (например, 5-летних), которые вряд ли будут участвовать в опросе о своем рабочем месте. В столбце «возраст» указан возраст 9999999999 лет. «Мужской» и «мужской» означают одно и то же, но рассматриваются как две разные категории. В столбцах «Самозанятый» и «Вмешательство в работу» отсутствуют несколько полей. Необходимо исправить эти значения.

Один из самых простых способов решения этой проблемы — просто игнорировать или удалять строки, в которых отсутствуют данные, исключая их из анализа. Однако этот метод может быть неэффективным из-за потери информации. Другой способ — заполнить пробелы, заменив пропущенные значения каким-либо образом. В базовых реализациях все пропущенные значения просто заменяются средним значением, медианой или константой. Сначала нужно определить, что делать с пропущенными значениями в столбцах `self_employed` и `work_interfere`. В обоих случаях столбец содержит категориальные данные.

```
features['self_employed'] =  
features['self_employed'].fillna('undefined')  
features['work_interfere'] =  
features['work_interfere'].fillna('undefined')  
print(features.head)
```

Рассмотрим статистику пропущенных данных по различным переменным.

```
missing_counts = features.isnull().sum()  
print(missing_counts)
```

4. Поиск неявных дубликатов

Мы обнаружили 49 различных значений для свойства "пол". Некоторые из этих значений не следует рассматривать как разные категории. Мы можем составить два списка синонимов и заменить все варианты нужным нам значением для этой характеристики.

```

male_terms = ["male", "m", "mal", "msle", "malr", "mail",
"make"]

female_terms = ["female", "f", "woman", "femake",
"femaile", "femake"]

def clean_gender(response): if
response.lower().rstrip() in male_terms:
    return "Male" elif
response.lower().rstrip() in female_terms:
    return "Female"
else:
    return "Other"

features['Gender'] = features["Gender"].apply(lambda
x: clean_gender(x))

```

5. Выявление выбросов

В переменной Age встречаются значения, которые кажутся некорректными. Например, отрицательные значения или чрезвычайно большие целые числа могут негативно повлиять на результат работы алгоритма машинного обучения, и нам необходимо их исключить. Для этого возьмем эвристическую оценку возраста, в котором люди могут работать: от 14 до 100 лет.

```

features.Age.loc[(features.Age <14) | (features.Age>
100)] = np.nan

print(features.isnull().sum()['Age']) features['Age'] =
features['Age'].fillna(features['Age'].mean())

```

6. Кодирование данных

Многие алгоритмы машинного обучения ожидают числовые входные данные, поэтому вам нужно придумать способ представления категориальных данных в числовом виде. Одним из решений может быть случайное присвоение числового значения каждой категории и сопоставление набора данных из исходных категорий с соответствующим числом. Например, рассмотрим столбец «отпуск» (насколько легко вам взять больничный из-за психического расстройства?).

```
features['leave'].value_counts(dropna=False)
```

а) Ручное кодирование

```

features['leave'] = df['leave'].map({'Very difficult': 0,
                                     'Somewhat difficult': 1,

```



```
'Don\'t know': 2,  
'Somewhat easy': 3,  
'Very easy': 4})
```

b) **Метод однократного кодирования** Используется для кодирования номинальных данных. Для каждого значения создается отдельный столбец, а значения 1 и 0 используются для обозначения выражения каждого значения. Эти новые столбцы часто называют фиктивными переменными.

Метод One-Hot Encoding используется для преобразования номинальных (категориальных) данных в числовую форму, которая может быть использована для обучения нейронных сетей.

Суть метода заключается в следующем:

1. Каждая уникальная категория в номинальном признаке представлена как отдельный бинарный признак (столбец). То есть, если у вас есть атрибут «Цвет» с категориями «Красный», «Зеленый» и «Синий», то будут созданы три новых бинарных атрибута.
2. В каждом бинарном признаке значение 1 указывает на принадлежность объекта к соответствующей категории, а значение 0 указывает на то, что объект не принадлежит к этой категории.
3. Остальные числовые характеристики данных можно оставить без изменений или, при необходимости, преобразовать в числовой формат.

Преимущества Методика One-Hot Encoding:

- Позволяет представлять номинальные данные в удобной числовой форме без внесения искажений или упорядочивания между категориями.
- Подходит для использования с нейронными сетями, которые ожидают числовые входные данные. Однако следует также учитывать некоторые моменты:

Одноразовое кодирование (One-Hot Encoding) может привести к увеличению объема данных при большом количестве уникальных категорий. Это может вызвать проблему, известную как «проклятие одноразового кодирования».

В случае больших данных и множества категорий использование One-Hot Encoding может быть вычислительно затратным.

Важно правильно применять One-Hot Encoding в соответствии с контекстом и требованиями вашей конкретной задачи и данных.

```
pd.get_dummies(features['leave'])
```

7. Зависимости данных

Для построения модели в любом случае желательно исключить взаимозависимости из данных, поскольку переменные, зависящие друг от друга, как минимум увеличивают вычислительные затраты, а иногда и мешают построению качественной модели. Прежде чем задаваться вопросом об исключении ненужных переменных, необходимо разобраться, как обстоят дела с зависимостью в целом.

```
var_corr = round(features.corr(numeric_only=True), 2)
print(var_corr)
```

8. Нормализация данных

Алгоритмы машинного обучения, как правило, работают лучше или сходятся быстрее, когда различные функции (переменные) имеют меньший масштаб. Поэтому перед обучением моделей машинного обучения на таких данных обычно проводится их нормализация. Нормализация также делает процесс обучения менее чувствительным к масштабу признаков. Это приводит к улучшению коэффициентов после обучения.

Нормализация стандартного отклонения Параметр (StandardScaler) используется для изменения размера распределения значений таким образом, чтобы среднее значение наблюдаемых значений было равно 0, а стандартное отклонение — 1.

При нормализации по принципу «минимум-максимум» наименьшее наблюдаемое значение принимается равным 0, а наибольшее — равным 1.

Нормализация стандартного отклонения

```
from sklearn.preprocessing import StandardScaler
scale_features_std = StandardScaler()
features[['Age']] = scale_features_std.fit_transform(features[['Age']])
print(features.head())
```

Мин-макс нормализация

```
features["Age"] = df["Age"]
features['Age'] = features['Age'].fillna(features['Age'].mean())
print(features.head())

from sklearn.preprocessing import MinMaxScaler
scale_features_mm = MinMaxScaler()
features[["Age"]] = scale_features_mm.fit_transform(features[["Age"]])
print(features.head())
```

9. Разделение данных для обучения и тестирования.

Для обучения модели данные обычно делятся на три основные выборки: обучающий набор, проверочный набор и тестовый набор.

1. Тренировочный набор: Тренировочный набор используется для обучения нейронной сети. Он содержит набор выборок данных, на которых будет обучаться сеть. Обычно это самая большая часть доступных данных, например, 70-80% от общего объема.
2. Набор данных для валидации: Набор данных для валидации используется для настройки гиперпараметров модели, таких как количество слоев, количество нейронов и скорость обучения. Эта выборка помогает оценить производительность модели на данных, которые она ранее не видела. Обычно это около 10-20% от общего объема данных.
3. Тестовый набор: Тестовый набор используется для окончательной оценки производительности модели после ее обучения и настройки. Эта выборка должна быть независима от обучающей и валидационной выборок. Она помогает оценить обобщающую способность модели на основе новых, ранее не представленных данных. Обычно это составляет около 10-20% от общего объема данных.

При разделении данных важно помнить о следующих принципах:

Разделение данных должно быть случайным, чтобы избежать смещения или корреляции между выборками. Данные в каждой выборке должны быть репрезентативными для общего распределения данных. Применение хорошей стратегии разделения данных помогает оценить и сравнить производительность модели с внешними данными и предотвращает переобучение, когда модель показывает хорошие результаты только на обучающих данных, но плохо справляется с новыми данными.

```
features_train, features_test, labels_train, labels_test
= train_test_split(features, labels, test_size=0.2,
random_state = 0)

print(features.shape)
print(features_train.shape)
print(features_test.shape)
```

Индивидуальное задание:

1. Найдите набор данных на Kaggle или любом другом хранилище и скачайте его (структура набора данных должна быть похожа на приведенную в примере).

Например, Вы можете использовать набор данных о покупателях магазина из файла «Customers.csv».

Данные о покупателях магазина представляют собой подробный анализ идеальных клиентов креативного магазина. Они помогают бизнесу лучше понимать своих клиентов. Владелец магазина получает информацию о клиентах с помощью членских карт. Набор данных состоит из 2000 записей и 8 столбцов:

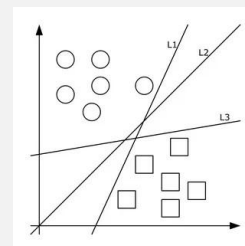
- Идентификатор клиента
 - Пол
 - Возраст
 - Годовой доход
 - Показатель покупательских расходов — оценка, присваиваемая магазином на основе поведения и характера расходов покупателя.
 - Профессия
 - Опыт работы - в годах
 - Размер семьи
2. Выполните описанные выше операции анализа и предварительной обработки набора данных для выбранного набора данных.

3. МЕТОД ОПОРНЫХ ВЕКТОРОВ

Методы опорных векторов (SVM) — это очень мощный и гибкий класс алгоритмов обучения с учителем, предназначенных как для классификации, так и для регрессии. Рассмотрим, как использовать метод опорных векторов в задачах классификации.

```
%matplotlib inline
import numpy as np
import matplotlib.pyplot as plt
from scipy import stats

# Давайте воспользуемся настройками
библиотеки Seaborn по умолчанию.
import seaborn as sns
sns.set()
```



Обучение линейного классификатора

```
# Загрузка библиотек из sklearn, импорт наборов данных
from sklearn.preprocessing import StandardScaler
from sklearn.svm import LinearSVC
```

```
# Загрузка данных только с двумя классами и двумя
признаками
iris = datasets.load_iris()
features = iris.data[:300,:2]
target = iris.target[:300]
# Стандартизация признаков
scaler = StandardScaler()
features_standardized = scaler.fit_transform(features)
# Создание классификатора опорных векторов
svc = LinearSVC(C=1.0)
# Обучение модели
model = svc.fit(features_standardized, target)
```

Визуализация полученных результатов

```
# Загрузка библиотеки
from matplotlib import pyplot as plt
```

```
# Построение графика точек данных и их раскрашивание в
# соответствии с их классом
color = ["red" if C == 0 else "blue" for C in target]
plt.scatter(features_standardized[:,0],features_standardized[:,
1], c=color)
```

```
# Создание гиперплоскости
w = svc.coef_[0] a = -w[0] / w[1]
xx = np.linspace(-2.5, 2.5)
yy = a * xx - (svc.intercept_[0]) / w[1]
```

```
# Нарисуем гиперплоскость
plt.plot(xx,yy)
plt.show()
```

Возможности метода SVM расширяются при его сочетании с ядрами.

Вам необходимо обучить классификатор на основе опорных векторов, но ваши классы не являются линейно разделимыми.

```
# Загрузка библиотек из sklearn.svm
import SVC from sklearn
import datasets from sklearn.preprocessing
import StandardScaler
import numpy as np
```

```
# Устанавливаем начальное случайное значение np.random.seed(0)
```

```
# Генерируем два признака
features = np.random.randn(200, 2)
# Генерируем линейно разделимые классы
target_xor = np.logical_xor(features[:, 0] > 0 ,
features[:, 1] > 0 )
target = np.where(target_xor, 0, 1)
```

```
# Создание машины опорных векторов с ядром на основе
радиальных базисных функций (ядро RBF)
svc= SVC(kernel="rbf", random_state=0, gamma=1, C=1)
```

```
# Обучение модели классификатора
model= svc.fit(features, target)
```

Используем функцию, которая строит график наблюдений и гиперплоскости границы решения в двумерном пространстве.

```
# Мы построим график наблюдений и границу принятия решения.
# гиперплоскость
from matplotlib.colors import ListedColormap import
matplotlib.pyplot as plt
def plot_decision_regions(X, y, classifier): cmap =
ListedColormap(("red", "blue"))
    xx1, xx2 = np.meshgrid(np.arange(-3, 3, 0.02), np.arange(-3,
3, 0.02))
    Z = classifier.predict(np.array([xx1.ravel(),
xx2.ravel()]).T)
    Z = Z.reshape(xx1.shape)
    plt.contourf(xx1, xx2, Z, alpha=0.1, cmap=cmap) for idx, cl
in enumerate(np.unique(y)):
    plt.scatter(x=X[y == cl, 0], y=X[y == cl, 1], alpha=0.8,
c=cmap(idx), label=cl)
```

Рассмотрим данные, содержащие две характеристики (т.е. два измерения) и вектор целевых значений, в каждом из которых указан класс. Обратите внимание, что классы назначены таким образом, что они линейно неразделимы. То есть, нет прямой линии, которую можно было бы провести, чтобы разделить два класса.

Давайте создадим классификатор на основе метода опорных векторов с линейным ядром:

```

# Создание классификатора опорных векторов с линейным
ядром
svc_linear= SVC(kernel="linear", random_state=0, C=1)
# Обучим модель
svc_linear.fit(features, target)
SVC(C=1,cache_size=200, class_weight=None, coef0=0.0,
decision_function_shape='ovr',degree=3, gamma='auto',
kernel='linear', max_iter=-1, probability=False,
random_state=0, shrinking=True,
tol=0.001,verbose=False)

plot_decision_regions(features,
target, classifier=svc_linear)
plt.show()

plt.scatter(x=X[y == c1, 0], y=X[y == c1, 1], alpha=0.8,
c=cm.map(idx), label=c1)

```

Очевидно, что линейная гиперплоскость очень плохо справлялась с разделением двух классов. Теперь давайте заменим линейное ядро на ядро радиальной базисной функции и используем его для обучения новой модели.

```

# Создание машины опорных векторов с ядром на основе
радиальных базисных функций (ядро RBF)
svc= SVC(kernel="rbf", random_state=0, gamma=1, C=1)
# Обучение модели классификатора
model= svc.fit(features, target)

# Построение графика наблюдений и гиперплоскости
plot_decision_regions(features, target, classifier=svc)
plt.show()

plt.scatter(x=X[y == c1, 0], y=X[y == c1, 1], alpha=0.8,
c=cm.map(idx), label=c1)

```

Используя ядро на основе радиальной базисной функции, мы смогли создать границу принятия решений, которая справлялась с разделением двух классов гораздо лучше, чем линейное ядро.

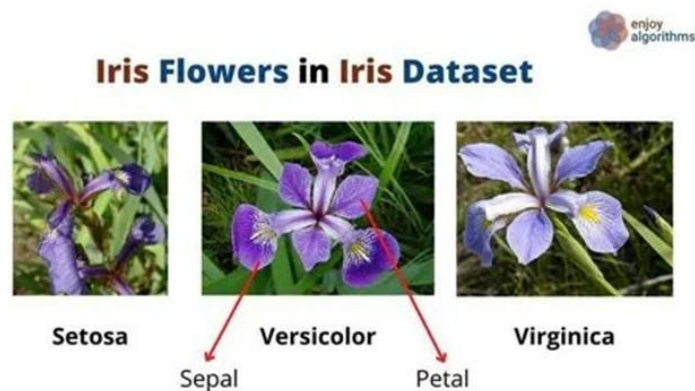
В библиотеке `scikit-learn` мы можем выбрать необходимое ядро с помощью параметра `kernel`. После выбора ядра необходимо указать соответствующие параметры ядра, такие как значение `d` (с помощью параметра `degree`) для

полиномиальных ядер и γ (с помощью параметра γ) для ядер на основе радиальных базисных функций. Также необходимо указать параметр штрафа C . В большинстве случаев во время обучения модели все эти параметры следует рассматривать как гиперпараметры, и для определения комбинации их значений, которая дает наилучшую модель, следует использовать методы выбора модели.

Индивидуальное задание:

1. Создайте и выполните на основе приведенных фрагментов кода программу на языке Питон. Опишите полученные результаты.
2. Реализуйте метод опорных векторов для модели линейного классификатора для двух признаков из датасета Iris библиотеки Scikit-learn в соответствии с вариантом индивидуального задания по последней цифре N Вашей зачетки.

N	Свойства из датасета	N	Свойства из датасета
0	SepalLength, PetalLength (0, 2)	5	PetalLength, SepalLength (2, 0)
1	SepalLength, PetalWidth (0, 3)	6	PetalWidth, SepalLength (3, 0)
2	SepalWidth, PetalLength (1, 2)	7	PetalLength, SepalWidth (2, 1)
3	SepalWidth, PetalWidth (1, 3)	8	PetalWidth, SepalWidth (3, 1)
4	PetalLength, PetalWidth (2, 3)	9	PetalWidth, PetalLength (3, 2)



4. КЛАСТЕРИЗАЦИЯ

Алгоритмы кластеризации направлены на поиск, на основе свойств данных, оптимального разбиения или дискретной маркировки групп точек.

Метод k-средних

В библиотеке Scikit-Learn представлено множество алгоритмов кластеризации. Рассмотрим алгоритм кластеризации k-средних, реализованный в классе `sklearn.cluster.KMeans`.

```
%matplotlib inline
import matplotlib.pyplot as plt
import seaborn as sns;
sns.set() # для стилизации диаграмм
import numpy as np
```

Алгоритм k-средних ищет заданное количество кластеров в немаркированном многомерном наборе данных. В основе модели k-средних лежат два предположения.

1. «Центр кластера» — это среднее арифметическое всех точек, принадлежащих данному кластеру.
2. Каждая точка находится ближе к центру своего кластера, чем к центрам других кластеров.

Процесс кластеризации:

1. Принимается гипотеза о центрах кластеров.
2. Повторяем следующий итерационный процесс до достижения сходимости:
 - шаг E (шаг ожидания): присваиваем точки ближайшим центрам кластеров;
 - шаг M (шаг максимизации): устанавливаем новые центры кластеров в соответствии со средними значениями.

Шаг E обновляет математическое ожидание принадлежности точек к тем или иным кластерам. Шаг M максимизирует некоторую целевую функцию, описывающую местоположение центров кластеров. В данном случае максимизация достигается простым усреднением данных в кластере.

Давайте создадим двумерный набор данных, содержащий четыре отдельных центра кластеров.

```
from sklearn.datasets import make_blobs
X,y_true = make_blobs(n_samples=300, centers=4,
cluster_std=0.60, random_state=0) plt.scatter(X[:,0], X[:, 1],
s=50)
```

Реализация метода k-средних с использованием приведенного выше алгоритма с максимизацией математического ожидания:

```
from sklearn.metrics import pairwise_distances_argmin
def find_clusters(X, n_clusters, rseed=2):
    rng = np.random.RandomState(rseed)
    # 1. Случайный выбор кластеров
    i = rng.permutation(X.shape[0])[:n_clusters] centers = X[i]
    while True:
        labels = pairwise_distances_argmin(X, centers)
    # 2a. Присваиваем метки в соответствии с ближайшим центром
        new_centers = np.array([X[labels == i].mean(0) for i in
range(n_clusters)])
    # 2b. Находим новые центры на основе средних значений точек
        if np.all(centers == new_centers):
            break
        centers = new_centers return centers, labels
    # 2c. Проверяем сходимость
centers, labels = find_clusters(X, 4)
#Количество кластеров следует выбрать заранее.
plt.scatter(X[:,0], X[:, 1], c=labels, s=50, cmap='viridis')
```

Кластеризацию неразмеченных данных можно выполнить с помощью модуля sklearn.cluster. Метод k-средних реализован в классе sklearn.cluster.KMeans.

Более подробно о параметрах метода читайте в <https://scikit-learn.org/stable/modules/generated/sklearn.cluster.KMeans.html#sklearn.cluster.KMeans>

```
from sklearn.cluster import KMeans
```

```
labels= KMeans(4, random_state=0).fit_predict(X)
plt.scatter(X[:,0], X[:, 1], c=labels, s=50, cmap='viridis')
```

DBSCAN

DBSCAN — это алгоритм кластеризации, основанный на плотности: получив набор точек в некотором пространстве, алгоритм группирует близко расположенные точки (точки со множеством близких соседей), помечая выбросами точки, находящиеся в одиночных областях с низкой плотностью (ближайшие соседи которых находятся далеко).

Алгоритм DBSCAN можно разложить на следующие этапы:

- Мы находим точки в эpsilon-окрестности каждой точки и выбираем основные точки, которые должны образовывать плотную область соседей minPts.
- Мы находим связные компоненты главных точек в графе соседей, игнорируя все неглавные точки.
- Если точка не является ядром кластера, мы присваиваем ей ближайший кластер, в противном случае считаем её шумом.

Параметры метода BSCAN можно найти [здесь](https://scikitlearn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluser.DBSCAN)

<https://scikitlearn.org/stable/modules/generated/sklearn.cluster.DBSCAN.html#sklearn.cluser.DBSCAN>

С помощью make_blobs мы генерируем 4 кластера. (make_blobs генерирует изотропные кластеры)

(сферические) гауссовы пятна)

```
from sklearn.datasets import make_blobs
from sklearn.preprocessing import tandardScaler
centers= [[1, 1], [-1, -1], [1, -1],[-1, 1]]
X, labels_true = make_blobs( n_samples=750, centers=centers,
cluster_std=0.4, random_state=0)
X = StandardScaler().fit_transform(X)
# Визуализация результатов
plt.scatter(X[:,0], X[:, 1])
plt.show()
```

```

from sklearn.cluster import DBSCAN from
sklearn import metrics

db = DBSCAN(eps=0.2, min_samples=10).fit(X) labels =
db.labels_
# Количество кластеров в метках, игнорируя шум, если он
присутствует.
n_clusters_ = len(set(labels)) - (1, if -1 in labels else 0)
n_noise_ = list(labels).count(-1)
print("Предполагаемое количество кластеров: %d"% n_clusters_)
print("Предполагаемое количество точек шума: %d"% n_noise_)

# Визуализация результатов unique_labels =
set(labels)
core_samples_mask = np.zeros_like(labels, dtype=bool)
core_samples_mask[db.core_sample_indices_] = True
colors = [plt.cm.Spectral(each) for each in np.linspace(0, 1,
len(unique_labels))]
for k, col in zip(unique_labels, colors):
    if k == -1:
# Точки, соответствующие шуму, выделены черным цветом
        col = [0, 0, 0, 1]
        class_member_mask = labels == k
        xy = X[class_member_mask & core_samples_mask] plt.plot(xy[:,
0], xy[:, 1], "o",
markerfacecolor=tuple(col), markeredgecolor="k",
markersize=14)
        xy = X[class_member_mask & ~core_samples_mask]
plt.plot( xy[:, 0], xy[:, 1], "o",
markerfacecolor=tuple(col), markeredgecolor="k",
markersize=6) plt.show()

```

Индивидуальное задание:

1. Модифицировать программу, исследовать процесс кластеризации для следующего набора параметров метода k-means.

Для функции <code>make_blobs</code>			Для функции <code>find_clusters</code>	N
<code>n_samples</code>	<code>centers</code>	<code>cluster_std</code>	<code>n_clusters</code>	по зачетке

300	4	1.0	4	0,1,3
200	6	0.5	3	2,6,7
300	3	1.0	6	4,5,8,9

2. Применить метод DBSCAN к следующей локализации центров кластеров:

centers= [[1, 1], [-1, -1], [1, 0],[0, 1]]	N =0,1,3 (по зачетке)
centers= [[0, 1], [-1, 0], [1, -1],[-1, 1]]	N =2,6,7
centers= [[2, 1], [-1, -2], [1, -1],[-1, 1]]	N = 4,5,8,9

Вычислить точность кластеризации.

5. РЕГРЕССИЯ

Линейная регрессия

Линейная регрессия и её расширения являются полезным методом прогнозирования, когда целевым вектором является количественное значение (например, цена дома, возраст). Такие модели популярны, потому что быстро учатся и дают очень легко интерпретируемые результаты.

```
matplotlib inline
import matplotlib.pyplot as plt

import numpy as np
import seaborn as sns

sns.set()
```

Самая простая форма линейной регрессионной модели — это подгонка разделяющей прямой линии к данным (приближение прямой линии), но такие модели можно расширить для моделирования более сложного поведения данных. Прямолинейное приближение — это модель вида $y = a * x + b$, где a называется наклоном, а b — y -пересечением.

```
rng= np.random.RandomState(1)
x= 10 * rng.rand(50)
y= 2 * x -5 + rng.randn(50)
plt.scatter(x,y)
```

<

Мы используем оценщик `LinearRegression` из библиотеки `Scikit-Learn` для обучения на этих данных и поиска оптимальной линии.

```
from sklearn.linear_model import LinearRegression
model= LinearRegression(fit_intercept=True)
model.fit(x[:,np.newaxis], y)
xfit= np.linspace(0, 10, 1000)
yfit= model.predict(xfit[:, np.newaxis])
plt.scatter(x,y)
plt.plot(xfit,yfit)
```

Параметры модели, которые нужно подгонять (в библиотеке Scikit-Learn они всегда заканчиваются подчёркивающей), включают наклон и пересечение. В этом случае соответствующие параметры — `coef_` и `intercept_`

```
print("Slope coefficient: ",model.coef_[0]) print("Point of  
intersection with the coordinate axis:", model.intercept_)
```

Возможности оценщика линейной регрессии гораздо шире: помимо прямолинейного приближения, он также может работать с многомерными линейными моделями вида $y = a_0 + a_1x_1 + a_2x_2 + \dots$ с несколькими значениями вектора X . Геометрически это похоже на подгонку плоскости к точкам в трёх измерениях или гиперплоскости к точкам в пространстве с ещё большим количеством измерений.

Полиномиальные базисные функции

```
from sklearn.pipeline import make_pipeline  
from sklearn.preprocessing import PolynomialFeatures  
poly_model= make_pipeline(PolynomialFeatures(7),  
LinearRegression())
```

После такого преобразования линейная модель может быть использована для сопоставления гораздо более сложных отношений между x и y .

```
rng= np.random.RandomState(1)  
x= 10 * rng.rand(50)  
y= np.sin(x) + 0.1 * rng.randn(50) #noisy sine wave  
poly_model.fit(x[:,np.newaxis], y)  
yfit= poly_model.predict(xfit[:, np.newaxis])  
plt.scatter(x,y)  
plt.plot(xfit,yfit)
```

Индивидуальное задание:

1. Создать программу для обработки данных в соответствии с предлагаемыми методами регрессии
2. Модифицировать программу, введя случайно генерируемый параметр рассеяния данных A для y .

Например:

```
y= 2 * x -5 + rng.randn(50)
```

заменить на

$A=2$

$y = 2 * x - 5 + \text{rng.randn}(50) * A$

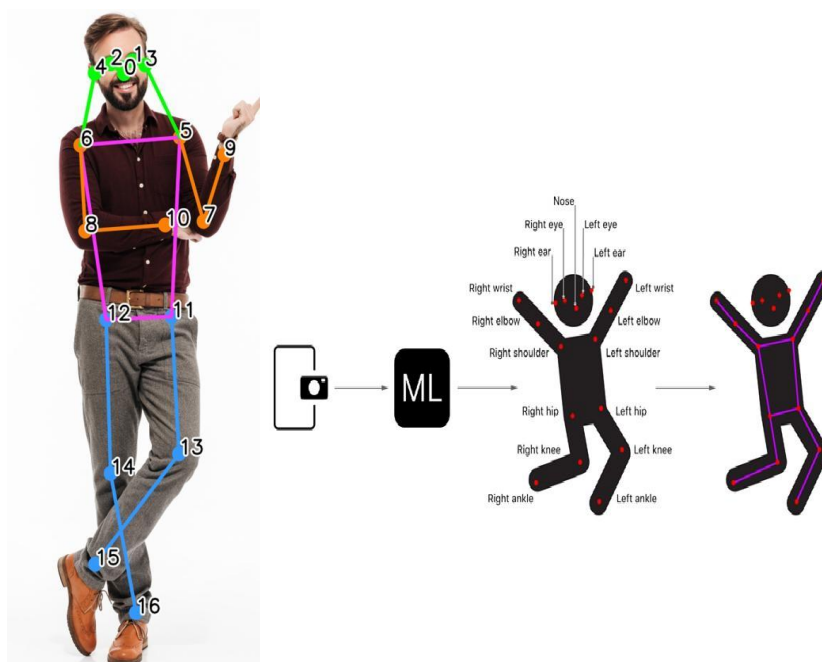
3. Исследовать метод машинного обучения (линейную и нелинейную регрессию) для нескольких значений параметра A
4. Сделать выводы о зависимости природы регрессии от параметра A .

6. ОЦЕНКА ПОЛОЖЕНИЯ ТЕЛА ЧЕЛОВЕКА С ПОМОЩЬЮ НЕЙРОННОЙ СЕТИ

Задача определения положения тела человека на изображении сводится к поиску всех людей на изображении и вычислении координат ключевых точек.

YOLO-Pose, разработанная компанией Ultralytics, является моделью для выявления ключевых точек человеческого тела. Эта модель предварительно обучена на наборе данных COCO8-Pose.

YOLO-Pose обучена определять положение 17 ключевых точек человеческого тела, показанных на рисунке.



В настоящее время текущая версия — YOLO-PoseV11. Эта модель представлена в 5 предварительно обученных версиях, которые различаются количеством параметров, скоростью работы, точностью и производительностью.

Перед началом нужно установить библиотеку Ultralytics, чтобы работать с командами YOLO

```
!pip3 install ultralytics
```

```
# Import Standard Libraries  
import numpy as np  
import cv2  
import matplotlib.pyplot as plt
```

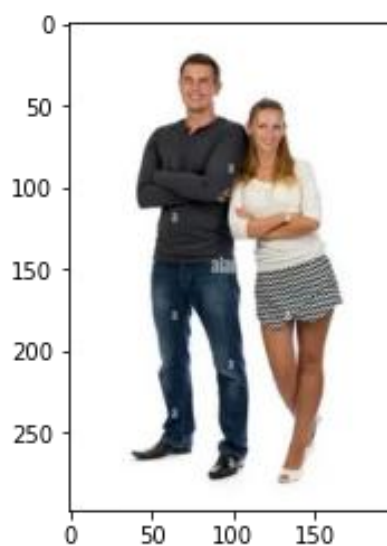
Анализ изображений

```
# Load a pre-trained model to detect key points in the body model
= YOLO("yolo11m-pose.pt")
```

```
list_point_names = ["nose", "left_eye", "right_eye", "left_ear",
"right_ear", "left_shoulder",
"right_shoulder", "left_elbow", "right_elbow", "left_wrist",
"right_wrist", "left_hip", "right_hip", "left_knee",
"right_knee", "left_ankle", "right_ankle"]
```

```
point_pairs = [
    (5, 11), (6, 12), # Torso
    (11, 12), # Between the legs
    (0, 1), (0, 2), (1, 3), (2, 4), # Head
    (5, 6), # Shoulder Connection
    (5, 7), (7, 9), # Left hand
    (6, 8), (8, 10), # Right hand
    (11, 13), (13, 15), # Left leg
    (12, 14), (14, 16), # Right leg
    (3, 5), (4, 6)
]
```

```
# Uploading and viewing an image for post-processing image =
cv2.imread("1--1.jpg")
image = cv2.cvtColor(image, cv2.COLOR_BGR2RGB)
plt.imshow(image)
```



Обработаем загруженное изображение с помощью нейронной сети и выведем результат

1. Параметр `iou` задаёт порог для немаксимального подавления (NMS)
2. Параметр `conf` устанавливает минимальную степень доверия для обнаружения. Объекты, обнаруженные с уверенностью ниже указанного значения, будут игнорированы
3. Если указать параметр `show=True`, результаты сети будут отображаться автоматически

```
results = model.predict(image, iou=0.4, conf=0.5, show=False)
print(results)
```

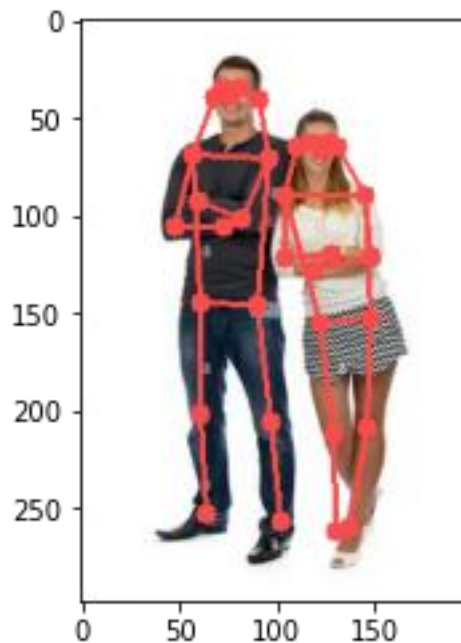
Результатом работы стал словарь. Для дальнейшего анализа нам понадобятся данные, содержащиеся с именами «boxes» и «keypoints».

```
print(results[0].boxes)
```

Поместим найденные точки на оригинальное изображение

```
img = image.copy()
for i in range(len(results[0].keypoints.xy)):
    keypoints = results[0].keypoints.xy[i]
    for j, point in enumerate(keypoints):
        (x, y) = point
        if x >= 0 and y >= 0:
            cv2.circle(img, (int(x), int(y)), 5, (249, 77, 77),
-1)
    for pair in point_pairs:
        start, end = pair
        if keypoints[start][0] > 0 and keypoints[start][1] > 0
and keypoints[end][0] > 0 and keypoints[end][1]:
            x1, y1 = int(keypoints[start][0]),
int(keypoints[start][1])
            x2, y2 = int(keypoints[end][0]),
int(keypoints[end][1])
            cv2.line(img, (x1, y1), (x2, y2), (249, 77, 77),
2)

plt.imshow(img)
```

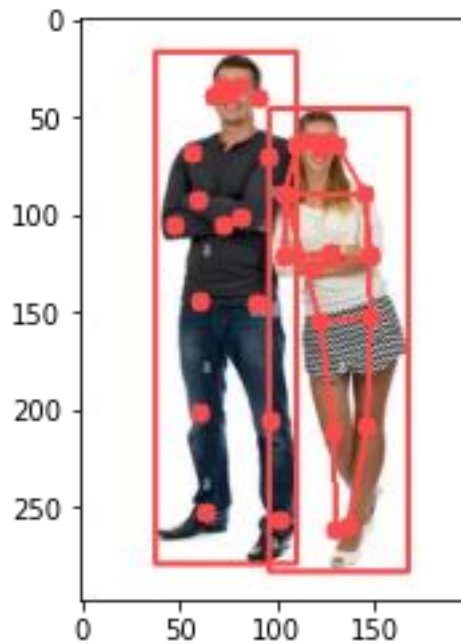


Добавим найденные ограничивающие рамки к изображению

```
results[0].boxes.xyxy[0]
```

```
img = image.copy()
for i in range(len(results[0].keypoints.xy)):
    keypoints = results[0].keypoints.xy[i]
    for j, point in enumerate(keypoints):
        (x, y) = point
        if x >= 0 and y >= 0:
            cv2.circle(img, (int(x), int(y)), 5, (249, 77, 77), -
1)
for i in range(len(results[0].boxes.xyxy)):
    bounding_box = results[0].boxes.xyxy[i]
    cv2.rectangle(img, (int(bounding_box[0]),
int(bounding_box[1])), (int(bounding_box[2]),
int(bounding_box[3])), (249, 77, 77), 2)
for pair in point_pairs:
    start, end = pair
    if keypoints[start][0] > 0 and keypoints[start][1] > 0 and
keypoints[end][0] > 0 and keypoints[end][1]:
        x1, y1 = int(keypoints[start][0]), int(keypoints[start][1])
        x2, y2 = int(keypoints[end][0]), int(keypoints[end][1])
        cv2.line(img, (x1, y1), (x2, y2), (249, 77, 77), 2)
```

```
plt.imshow(img)
```



Индивидуальное задание:

1. Используйте Yolo.Pose, чтобы определить ключевые точки для нескольких людей, сидящих, стоящих и лежащих (по 3 фотографии на каждую группу). Сохраните их в наборе данных.
2. Основываясь на этих данных, решите задачу кластеризации с помощью метода k-means для трех кластеров.
3. Вычислите средний вектор ключевых точек для каждого кластера и нарисуйте его. Сделайте выводы по точности кластеризации.