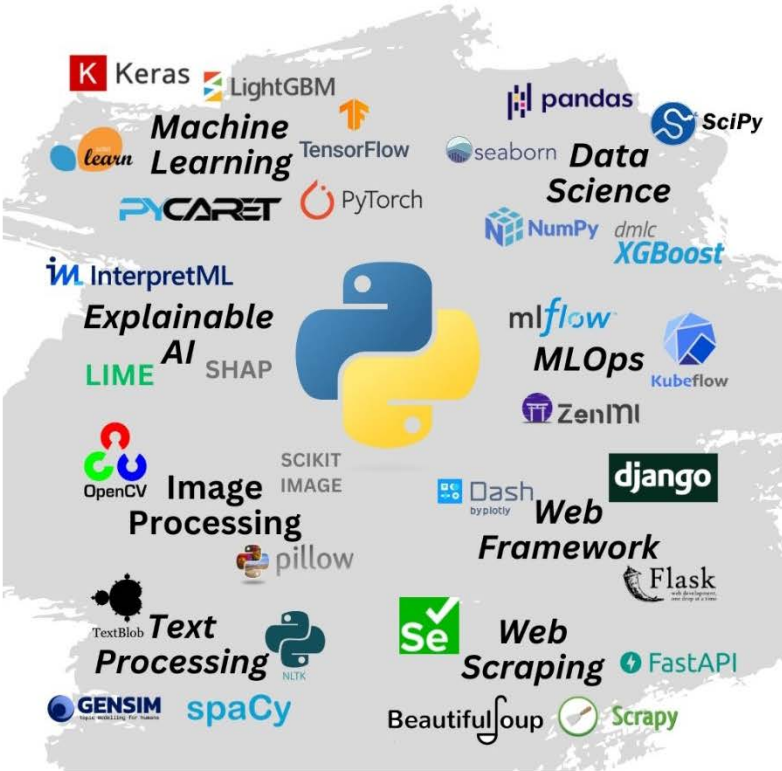
Мурат Дурмус

Практическое введение в

ОСНОВНЫЕ БИБЛИОТЕКИ И ФРЕЙМВОРКИ PYTHON

(с примерами кода)



Бумажный вариант доступен на Amazon:

https://www.amazon.com/dp/B0BW2MGYG4

Мурат Дурмус

Практическое введение в **основные библиотеки и фреймворки Python**

(с примерами кода)

Дизайн обложки:

Мурат Дурмус

Об авторе

Мурат Дурмус — генеральный директор и основатель AISOMA (компания из Франкфурта-на-Майне (Германия), специализирующаяся на разработке и консультировании технологий на основе ИИ) и автор книг "Mindful AI - Reflections on Artificial Intelligence" и "A Primer to the 42 Most commonly used Machine Learning Algorithms (With Code Samples)"

Связаться с автором можно через:

- LinkedIn: https://www.linkedin.com/in/ceosaisoma/

- E-Mail: murat.durmus@aisoma.de

Примечание:

Примеры кода и их описание в этой книге написаны при поддержке ChatGPT (OpenAI).

" Python — это не просто язык, это сообщество, где разработчики могут учиться, сотрудничать и творить чудеса."

- Гвидо ван Россум

(Создатель Python)

КРАТКАЯ ИСТОРИЯ ПРОГРАММИРОВАНИЯ НА ЯЗЫКЕ PYTHON

Python — популярный язык программирования высокого уровня для различных приложений, включая веб-разработку, научные вычисления, анализ данных и машинное обучение. Его простота, удобочитаемость и универсальность сделали его популярным среди программистов всех уровней квалификации. Вот краткая история языка программирования Python.

Python был создан в конце 1980-х Гвидо ван Россумом, работавшим в Национальном исследовательском институте математики и компьютерных наук в Нидерландах. Ван Россум искал язык программирования, который было бы легко читать и писать и который можно было бы использовать для различных приложений. Он назвал язык в честь британской комедийной группы «Монти Пайтон», так как был поклонником их телешоу.

Первая версия Python, Python 0.9.0, была выпущена в 1991 году. Эта версия включала в себя многие функции, которые до сих пор используются в Python, такие как модули, исключения и основные типы данных списков, словарей и кортежей.

Python 1.0 был выпущен в 1994 году и включал в себя множество новых функций, таких как лямбда, карта, фильтр и сокращение. Эти функции упростили написание функционального кода на

Python.

В 2000 году был выпущен Python 2.0, в котором были представлены генератор списков, новый сборщик мусора и сборщик мусора с обнаружением циклов. Понимание списков упростило написание кода, работающего со списками и другими итерируемыми объектами.

Python 3.0, значительное обновление языка, было выпущено в 2008 году. В эту версию было внесено множество изменений и улучшений, включая переработанную функцию печати, новый синтаксис форматирования строк и новый оператор деления. В последней версии также удалены некоторые функции, которые считались устаревшими или излишними.

С момента выпуска Python 3.0 было выпущено несколько второстепенных выпусков, каждый из которых содержал новые функции и улучшения, сохраняя при этом обратную совместимость с существующим кодом. Эти релизы включают в себя такие функции, как синтаксис async/await для асинхронного программирования, аннотации типов для улучшения читабельности и удобства сопровождения кода, а также усовершенствования сборщика мусора и стандартной библиотеки.

Популярность Python с годами неуклонно росла, и сейчас он является одним из самых популярных языков программирования в мире. Его широко используют веб-разработчики, специалисты по данным и инженеры по машинному обучению, среди прочих. Популярность Python обусловлена его простотой, удобочитаемостью и

универсальностью, а также большим и активным сообществом разработчиков, которые вносят свой вклад в язык и его экосистему библиотек и инструментов.

В заключение следует отметить, что язык программирования Python прошел долгий путь с момента своего появления в конце 1980-х годов. За прошедшие годы он претерпел множество изменений и улучшений, но его основные ценности — простота, удобочитаемость и универсальность — остались неизменными. Более того, популярность Python не собирается снижаться, и, скорее всего, он останется популярным среди программистов на долгие годы.

Если коротко:

- Python был создан Гвидо ван Россумом в конце 1980-х, когда он работал в Национальном исследовательском институте математики и информатики в Нидерландах.
- Первая версия Python, Python 0.9.0, была выпущена в 1991 г.
- Python 1.0 был выпущен в 1994 г. и включал множество новых функций, таких как лямбда-выражение, сопоставление, фильтрация и сокращение.
- В 2000 году был выпущен Python 2.0, в котором были представлены списки, новый сборщик мусора и сборщик мусора с обнаружением циклов.

- Python 3.0, крупное обновление языка, было выпущено в 2008 г. В эту версию было внесено множество изменений и улучшений, включая переработанную функцию печати, новый синтаксис форматирования строк и новый оператор деления.

- С момента выпуска Python 3.0 было выпущено несколько второстепенных выпусков, каждый из которых содержал новые функции и улучшения, сохраняя при этом обратную совместимость с существующим кодом.

- Python стал одним из самых популярных языков программирования в мире, который используется для самых разных приложений, таких как веб-разработка, научные вычисления, анализ данных и машинное обучение.

- **Популярность Python обусловлена его простотой, удобочитаемостью и универсальностью, а также большим и активным сообществом разработчиков, которые вносят свой вклад в язык и его экосистему библиотек и инструментов.**

НАУКА О ДАННЫХ

Наука о данных — это междисциплинарная область, которая включает в себя извлечение, анализ и интерпретацию больших и сложных наборов данных. Она сочетает в себе элементы статистики, компьютерных наук и знаний в предметной области для извлечения идей и знаний из данных.

Специалисты по данным используют различные инструменты и методы для сбора, обработки и анализа данных, включая статистический анализ, машинное обучение, интеллектуальный анализ данных и визуализацию данных. Они работают с большими и сложными наборами данных, чтобы выявить закономерности, взаимосвязи и идеи, которые могут помочь в принятии решений и повысить ценность бизнеса.

Наука о данных применяется в различных областях, включая бизнес, здравоохранение, финансы и социальные науки. Она информирует о различных решениях, от разработки продукта до маркетинга и принятия политических решений.

PANDAS

Python Pandas — это библиотека для обработки и анализа данных с открытым исходным кодом для языка программирования Python. Она предоставляет набор структур данных для эффективного хранения больших наборов данных и управления ими, а также различные инструменты для анализа, парсинга и предварительной обработки данных.

Некоторые из ключевых структур данных в Pandas включают Series, что представляет собой одномерный объект, похожий на массив, который может содержать данные любого типа; и DataFrame, представляющий собой двумерную табличную структуру данных со строками и столбцами, которую можно рассматривать как электронную таблицу или таблицу SQL.

Pandas также предоставляет ряд функций и методов обработки данных, таких как фильтрация, сортировка, слияние, группировка и агрегирование данных. Она также поддерживает инструменты визуализации данных, которые позволяют пользователям отображать и визуализировать данные различными способами.

Она широко используется в анализе данных и науке о данных и считается одним из основных инструментов для работы с данными в Python. Pandas также часто используется в сочетании с другими популярными библиотеками данных, такими как NumPy, Matplotlib и SciPy.

Пример того, как вы можете использовать Pandas для чтения файла CSV, обработки данных и вывода их в новый файл:

```
import pandas as pd

# Read in the CSV file
data = pd.read_csv('my_data.csv')

# Print the first few rows of the data

print(data.head()
)

# Filter the data to include only rows where
the 'score' column is greater than 90
filtered_data = data[data['score'] > 90]

# Create a new column that calculates the
average of the 'score' and 'time' columns
filtered_data['average'] =
(filtered_data['score'] +
filtered_data['time']) / 2

# Output the filtered data to a new CSV file
filtered_data.to_csv('my_filtered_data.csv',
index=False)
```

В этом примере мы сначала импортируем библиотеку Pandas, используя **import pandas as pd**. Затем мы читаем файл CSV с именем **my_data.csv**, используя функцию **pd.read_csv()**, которая создает объект DataFrame. Затем мы используем метод **head()**для вывода первых нескольких строк данных.

Затем мы фильтруем данные, чтобы включить только строки, в которых столбец 'score' больше 90, используя логическое индексирование. Затем мы создаем новый столбец под названием 'average',

который вычисляет среднее значение столбцов 'score' и 'time', используя основные арифметические операции.

Наконец, мы используем метод **to_csv()** для вывода отфильтрованных данных в новый CSV-файл с именем **my_filtered_data.csv** с параметром **index=False**, указывающим, что мы не хотим включать индекс DataFrame в качестве столбца в выходной файл.

За и против

За:

- Простая в использовании и очень универсальная библиотека для обработки и анализа данных.
- Предоставляет мощные инструменты для обработки больших наборов данных, включая быстрое индексирование, фильтрацию, группировку и операции слияния.
- Поддерживает широкий спектр форматов ввода и вывода, включая CSV, Excel, базы данных SQL и JSON.
- Предлагает богатый набор инструментов визуализации данных, включая линейные графики, точечные диаграммы, гистограммы и многое другое.
- Имеет большое и активное сообщество пользователей и разработчиков, а это означает, что доступно множество онлайн-ресурсов и поддержки.
- Может использоваться вместе с другими популярными библиотеками данных, такими как NumPy, SciPy и Matplotlib.

Против:

- Pandas может использовать большой объем памяти при работе с очень большими

наборами данных и может быть не лучшим выбором для приложений реального времени или данных очень большого объема.

- Некоторые функции и методы могут быть сложными и трудными для понимания, особенно для новых пользователей.
- Может быть медленным при выполнении определенных операций, таких как применение функций к большим наборам данных или выполнение нескольких слияний или конкатенаций.
- Может не всегда давать желаемые результаты, особенно при работе с беспорядочными или неструктурированными данными.
- Некоторые пользователи сообщали о проблемах совместимости и переносимости между разными версиями Pandas или между Pandas и другими библиотеками.

NUMPY

NumPy — это библиотека Python для численных вычислений. Она предоставляет мощные структуры данных, такие как n-мерные массивы или «ndarrays», и широкий спектр математических функций для эффективной работы с этими массивами.

Она широко используется в науке о данных, машинном обучении, научных вычислениях и инженерии, а также в других областях. Библиотека построена на основе низкоуровневых языков, таких как C и Fortran, что позволяет NumPy быть быстрым и эффективным даже при работе с большими наборами данных.

В дополнение к своей основной функциональности NumPy также предоставляет инструменты для интеграции с другими библиотеками научных вычислений на Python, такими как SciPy и Pandas. В целом, NumPy — важный инструмент для всех, кто работает с числовыми данными в Python.

Пример кода, который демонстрирует, как создать массив NumPy, выполнять над ним математические операции и нарезать его:

```
import numpy as np

# Create a 1-dimensional NumPy array
arr = np.array([1, 2, 3, 4, 5])

# Perform mathematical operations on the array
print("Original array:", arr)
print("Array multiplied by 2:", arr * 2)
```

```
print("Array squared:", arr ** 2)
print("Array sine values:", np.sin(arr))

# Create a 2-dimensional NumPy array

arr2d = np.array([[1, 2, 3], [4, 5, 6], [7, 8,
9]])

# Slice the array to get a subarray
sub_arr = arr2d[:2, 1:]
print("Original 2D array:\n", arr2d)
print("Subarray:\n", sub_arr)
```

Резуультат:

```
Original array: [1 2 3 4 5]
Array multiplied by 2: [ 2  4  6  8 10]
Array squared: [ 1  4  9 16 25]
Array sine values: [ 0.84147098  0.90929743
0.14112001 -0.7568025  -0.95892427]
Original 2D array:
 [[1 2 3]
 [4 5 6]
 [7 8 9]]
Subarray
 : [[2 3]
 [5 6]]
```

В этом примере мы импортируем библиотеку NumPy и создаем одномерный массив **arr** со значениями **[1, 2, 3, 4, 5]**. Затем мы выполняем несколько математических операций с массивом, такие как умножение на 2 и возведение значений в квадрат, используя функции NumPy.

Далее мы создаем двумерный массив **arr2d** со значениями **[[1, 2, 3], [4, 5, 6], [7, 8, 9]]**. Мы нарезаем этот массив, чтобы получить подмассив **sub_arr**, содержащий элементы в первых двух строках и

последних двух столбцах. Мы печатаем исходные массивы и подмассив, чтобы показать результаты.

За и против

За:

- Эффективность и быстрота: NumPy построен на основе языков низкого уровня, таких как C и Fortran, что делает его намного более быстрым и эффективным, чем чистый код Python при численных вычислениях.

- Мощные структуры данных: NumPy предоставляет мощные n-мерные массивы или «ndarrays», которые позволяют эффективно хранить большие наборы данных и управлять ими.

- Комплексные математические функции: NumPy предоставляет широкий спектр математических функций, таких как тригонометрические, логарифмические и статистические функции, которые упрощают выполнение сложных вычислений с массивами.

- Интеграция с другими библиотеками Python: NumPy легко интегрируется с другими библиотеками научных вычислений на Python, такими как SciPy, Pandas и Matplotlib, что позволяет выполнять более продвинутый анализ и визуализацию данных.

Против:

- Крутая кривая обучения: NumPy может быть сложным в освоении, особенно для новичков,

которые не знакомы с такими понятиями программирования, как массивы и векторизация.

- Использование памяти: массивы NumPy могут использовать много памяти, что может быть проблемой при работе с очень большими наборами данных.
- Отсутствие гибкости: NumPy оптимизирован для числовых вычислений и не так гибок, как чистый код Python в задачах программирования общего назначения.

В целом, плюсы NumPy намного перевешивают минусы, особенно при работе с большими наборами данных или выполнении сложных числовых вычислений. Однако важно помнить об ограничениях NumPy и выбирать правильный инструмент для работы.

SEABORN

Seaborn — это библиотека визуализации данных Python, созданная поверх Matplotlib. Он предоставляет высокоуровневый интерфейс для создания информативной и привлекательной статистической графики в Python.

Он предлагает ряд методов визуализации для статистической графики, в том числе:

- **Одномерные и двумерные графики**: гистограммы, оценки плотности ядер, коробчатые диаграммы, диаграммы скрипки и диаграммы рассеяния.
- **Графики регрессии и категорий**: линейная регрессия, логистическая регрессия, графики рассеяния по категориям и гистограммы.
- **Матричные графики**: карты интенсивности, кластерные карты и парные графики.
- **Графики временных рядов**: линейные графики, карты интенсивности временных рядов и сезонные графики.

Seaborn разработан для беспрепятственной работы с Pandas, популярной библиотекой обработки данных на Python, и может легко обрабатывать большие и сложные наборы данных. Он также предоставляет ряд параметров настройки для графиков, включая цветовые палитры, темы и стили.

В целом, Seaborn — это мощная и удобная библиотека для создания информативных и визуально привлекательных статистических графиков на Python.

Пример кода, использующего Seaborn для создания точечной диаграммы:

```
import seaborn as sns
import pandas as pd

# Load dataset
df = pd.read_csv('my_dataset.csv')

# Create scatter plot
sns.scatterplot(x='x_column', y='y_column',
data=df)

# Show plot
sns.plt.show()
```

В этом примере мы сначала импортируем библиотеку Seaborn и библиотеку Pandas для загрузки набора данных и управления им. Затем мы загружаем набор данных из файла CSV с помощью Pandas.

Затем мы создаем точечную диаграмму, используя функцию Seaborn **scatterplot()**, передавая имена столбцов x и y из набора данных в качестве аргументов. Мы также передаем аргумент **data**, чтобы указать набор данных, который мы хотим построить.

Наконец, мы используем функцию Seaborn **plt.show()** для отображения графика на экране. Seaborn автоматически оформляет график с помощью темы, принимаемой по умолчанию, и мы можем дополнительно настроить график, используя другие

функции и аргументы Seaborn.

Этот код создает точечную диаграмму, показывающую взаимосвязь между двумя переменными в наборе данных, где ось x представляет переменную «x_column», а ось y представляет переменную «y_column». Каждая точка на точечной диаграмме представляет одно наблюдение в наборе данных. Seaborn автоматически добавляет метки к осям и легенду, объясняющую значение различных цветов на графике.

За и против

За:

- Привлекательные и информативные визуализации: Seaborn предлагает ряд методов визуализации, оптимизированных для создания привлекательной и информативной статистической графики в Python. Она предлагает широкий спектр вариантов настройки цветов, стилей и тем, что позволяет легко создавать визуально привлекательные графики, адаптированные к потребностям пользователя.

- Дружественный интерфейс: Seaborn разработана так, чтобы ее было легко использовать, с простым и согласованным API, который позволяет легко создавать сложные визуализации с помощью всего нескольких строк кода. Она также предоставляет ряд встроенных наборов данных, которые можно использовать для практики или исследования.

- Интеграция с Pandas: Seaborn разработана для бесперебойной работы с Pandas, популярной библиотекой обработки данных на Python, которая упрощает обработку и визуализацию больших и сложных наборов данных.

- Универсальность: Seaborn предлагает широкий спектр методов визуализации, включая одномерные и двумерные графики,

регрессионные и категориальные графики, матричные графики и графики временных рядов, что делает ее универсальным инструментом для исследования и анализа данных.

Против:

- Ограниченный объем: Seaborn ориентирована на визуализацию статистических данных и не такая гибкая, как другие библиотеки визуализации для задач визуализации данных общего назначения.

- Крутая кривая обучения: хотя Seaborn разработана так, чтобы ее было легко использовать, некоторым пользователям может быть сложно учиться, особенно если они не знакомы с концепциями статистической визуализации или библиотекой Pandas.

- Ограниченные возможности настройки: хотя Seaborn предлагает широкий спектр возможностей настройки, некоторые пользователи могут обнаружить, что они ограничены в уровне настройки, которого они могут достичь, особенно по сравнению с более продвинутыми библиотеками визуализации, такими как Matplotlib.

В целом, плюсы Seaborn намного перевешивают минусы, особенно для пользователей, которым необходимо создавать информативные и привлекательные статистические графики в Python.

Однако важно помнить об ограничениях Seaborn и выбирать правильный инструмент для работы.

SCIPY

Scipy — это библиотека научных вычислений с открытым исходным кодом для Python, которая предоставляет набор функций для математики, науки и техники. Она построена на основе библиотеки NumPy, которая обеспечивает эффективные операции с массивами для числовых вычислений.

Она организована в подпакеты, которые обеспечивают различные функции, такие как:

- **Интеграция и оптимизация**
- **Обработка сигналов и изображений**
- **Статистика и вероятность**
- **Интерполяция и экстраполяция**
- **Разреженная матрица и линейная алгебра**
- **Специальные функции и числовые процедуры**

Scipy широко используется в научных исследованиях, инженерии, науке о данных и других областях, где требуются численные вычисления. Она предоставляет удобный и мощный способ выполнения сложных вычислений и анализа в Python с большим и активным сообществом пользователей и разработчиков, которые вносят свой вклад в его разработку и обслуживание.

Пример кода, использующего Scipy для выполнения численного интегрирования:

```
import numpy as np
from scipy.integrate import quad

# Define function to integrate
def f(x):
    return np.exp(-x ** 2)

# Perform numerical integration
result, error = quad(f, -np.inf, np.inf)

# Print result
print("Result:", result)
print("Error:", error)
```

В этом примере мы сначала импортируем библиотеку NumPy и функцию **quad** из подпакета Scipy **integrate**. Затем мы определяем функцию **f(x)**, которую хотим интегрировать.

Затем мы используем функцию **quad** для выполнения численного интегрирования **f(x)** в диапазоне от отрицательной бесконечности до положительной бесконечности. Функция **quad** возвращает результат интегрирования и оценку ошибки.

Наконец, мы выводим результат и ошибку на консоль. В этом случае результатом должен быть квадратный корень из числа пи (приблизительно 1,77245385091).

Этот код демонстрирует, как Scipy можно легко и эффективно использовать в Python для выполнения сложных математических вычислений, таких как численное интегрирование.

За и против

За:

- Предоставляет полный набор инструментов для научных вычислений и численного анализа, включая интеграцию, оптимизацию, обработку сигналов, линейную алгебру и многое другое.
- Построена на основе NumPy, что упрощает работу с массивами и выполнение эффективных числовых операций.
- Большое и активное сообщество пользователей и разработчиков с множеством пакетов и модулей с открытым исходным кодом, доступных для расширения ее функциональности.
- Хорошо документирована, множество примеров и учебных пособий доступны в Интернете.
- Портативность и кросс-платформенность, поддержка многих операционных систем и аппаратных архитектур.

Против:

- Может быть сложной и трудной в освоении для начинающих из-за большого количества доступных функций и подпакетов.
- Некоторые функции могут требовать

значительных вычислительных ресурсов и дополнительных знаний в области численного анализа и оптимизации производительности.

- Некоторые функции могут иметь ограничения или допущения, которые могут не подходить для всех приложений.

- Требует тщательного рассмотрения точности и правильности числовых расчетов, особенно для научных приложений, где точность имеет решающее значение.

- Некоторые функции могут работать не так быстро, как оптимизированный код, написанный на языках более низкого уровня, таких как C или Fortran.

В целом, Scipy — это мощная и широко используемая библиотека для научных вычислений на Python, но она может быть не лучшим выбором для всех приложений, и для ее эффективного использования необходимо тщательно учитывать ее сильные стороны и ограничения.

MATPLOTLIB

Matplotlib — популярная библиотека визуализации данных для языка программирования Python. Она предоставляет возможность создавать широкий спектр статических, анимированных и интерактивных визуализаций в Python.

Первоначально Matplotlib была разработана Джоном Д. Хантером в 2003 году и в настоящее время поддерживается командой разработчиков. Это программное обеспечение с открытым исходным кодом, доступное под лицензией типа BSD.

Matplotlib хорошо работает с NumPy, популярной библиотекой для числовых вычислений в Python, и часто используется в сочетании с другими библиотеками для научных вычислений, такими как SciPy и Pandas.

Она предоставляет широкий спектр функций построения графиков, включая линейные графики, точечные диаграммы, гистограммы, трехмерные графики и многое другое. Она также обеспечивает высокую степень настройки, позволяя пользователям изменять практически все аспекты своих графиков, включая оси, метки, цвета и стили.

Библиотеку можно использовать в различных условиях, от исследовательского анализа данных до научных исследований и создания графики публикационного качества. Она широко используется в научных кругах и промышленности и считается одним

из основных инструментов в экосистеме Python для обработки и анализа данных.

Пример фрагмента кода, который использует Matplotlib для создания простого линейного графика:

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

# Generate some sample data

x = np.linspace(0, 10, 100)
y = np.sin(x)

# Create a figure and axis object
fig, ax = plt.subplots()

# Plot the data
ax.plot(x, y)

# Add some labels and a title
ax.set_xlabel('X Label')
ax.set_ylabel('Y Label')
ax.set_title('Sinusoidal Plot')

# Show the plot
plt.show()
```

В этом примере мы сначала импортируем необходимые модули (Matplotlib и NumPy). Затем мы генерируем некоторые образцы данных (массив из 100 значений x, равномерно распределенных между 0 и 10, и массив соответствующих значений синуса).

Затем мы создаем объект фигуры и оси, используя функцию **subplots**. Затем мы используем функцию **plot** для построения данных на объекте оси.

Наконец, мы добавляем к графику несколько меток и заголовок, используя функции **set_xlabel**, **set_ylabel** и **set_title**. Затем мы используем функцию **show** для отображения графика.

Это всего лишь простой пример, а в Matplotlib есть много дополнительных функций для создания более сложных визуализаций.

За и против

За:

- Matplotlib — это широко используемая и хорошо зарекомендовавшая себя библиотека визуализации данных для Python с большим и активным сообществом разработчиков.
- Широкие возможности настройки, позволяющие пользователям изменять практически каждый элемент своих графиков, включая оси, метки, цвета и стили.
- Она предоставляет широкий спектр функций построения графиков, включая линейные графики, точечные диаграммы, гистограммы, трехмерные графики и многое другое.
- Может создавать высококачественные сюжеты, пригодные для публикации и презентации.
- Она хорошо интегрирована с другими библиотеками Python для анализа данных и научных вычислений, такими как NumPy, SciPy и Pandas.
- Она проста в использовании и изучении для выполнения основных задач построения графиков, что делает ее доступным для пользователей всех уровней.

Против:

- Синтаксис может быть многословным и сложным, особенно для более сложных задач настройки и построения графиков.
- Настройки по умолчанию для графиков могут не всегда быть эстетичными и могут потребовать дополнительной настройки.

- Не обеспечивает столько интерактивности или анимационных функций, как некоторые другие библиотеки визуализации данных, такие как Plotly или Bokeh.
- Она может быть медленнее в случае создания сложных или крупномасштабных визуализаций по сравнению с некоторыми другими библиотеками, такими как Seaborn.
- Для создания сложных или расширенных графиков может потребоваться больше кода и усилий, чем при использовании других библиотек, предоставляющих более специализированные функции построения графиков.

МАШИННОЕ ОБУЧЕНИЕ

Машинное обучение — это область искусственного интеллекта, которая разрабатывает алгоритмы, которые могут автоматически обучаться и улучшаться на основе данных.

В машинном обучении модель обучается на большом наборе данных пар ввода-вывода, называемом обучающим набором, а затем используется для прогнозирования новых, предполагаемых данных. Цель состоит в том, чтобы разработать модель, которая может хорошо обобщать новые данные, изучая закономерности и отношения в обучающих данных, которые можно применять к новым данным.

Существует несколько типов машинного обучения, в том числе **контролируемое**, **неконтролируемое** и **обучение с подкреплением**. В контролируемом обучении обучающий набор включает помеченные примеры пар вход-выход, а цель состоит в том, чтобы изучить функцию, которая может точно предсказать результат для новых входных данных. При неконтролируемом обучении обучающий набор не включает метки; цель состоит в том, чтобы обнаружить закономерности и взаимосвязи во входных данных. Наконец, при обучении с подкреплением агент учится взаимодействовать с окружающей средой для достижения цели, получая награды или штрафы в зависимости от действий.

Машинное обучение имеет множество применений, от распознавания изображений и обработки

естественного языка до систем рекомендаций и прогностической аналитики. Оно используется в различных отраслях, включая здравоохранение, финансы и электронную коммерцию, для автоматизации принятия решений, повышения эффективности и получения информации из данных.

SCIKIT-LEARN

Python scikit-learn (также известная как sklearn) — популярная библиотека машинного обучения для языка программирования Python. Она предоставляет ряд контролируемых и неконтролируемых алгоритмов обучения для различных типов задач анализа данных, таких как классификация, регрессия, кластеризация и уменьшение размерности.

Она был разработана Дэвидом Курнапо в рамках проекта Google Summer of Code в 2007 году и в настоящее время поддерживается командой разработчиков. Это программное обеспечение с открытым исходным кодом, доступное по разрешающей лицензии в стиле BSD.

Scikit-learn построена на основе других популярных научных вычислительных библиотек для Python, таких как NumPy, SciPy и matplotlib. Она также интегрируется с другими библиотеками машинного обучения и анализа данных, такими как TensorFlow и Pandas.

Scikit-learn предоставляет широкий спектр алгоритмов машинного обучения, в том числе:

- **Линейная и логистическая регрессия**
- **Метод опорных векторов (SVM)**
- **Деревья решений и случайные леса**
- **K-ближайшие соседи (KNN)**

- **Наивный Байес**
- **Алгоритмы кластеризации (например, K-Means)**
- **Методы уменьшения размерности (например, анализ главных компонентов)**

Она также предоставляет утилиты для выбора и оценки модели, такие как перекрестная проверка, поиск по сетке и показатели производительности.

Scikit-learn широко используется в научных кругах и промышленности для решения различных задач машинного обучения, таких как обработка естественного языка, распознавание изображений и прогностическая аналитика. Она считается одним из основных инструментов в экосистеме науки о данных Python.

Пример фрагмента кода, демонстрирующий, как использовать scikit-learn для обучения простой модели логистической регрессии:

```
from sklearn.linear_model
import
LogisticRegressio
n
from sklearn.datasets import
load_iris

# Load the iris
dataset
iris =
load_iris()

# Split the dataset into features (X) and
labels (y)
```

```
X, y = iris.data,
iris.target

# Create a LogisticRegression
object
logreg =
LogisticRegression()

# Fit the model using the iris
dataset
logreg.fit(X,
y)

# Predict the class labels for a new set of
features
new_X = [[5.1, 3.5, 1.4, 0.2], [6.2, 3.4,
5.4,
2.3]
]
predicted_y =
logreg.predict(new_X)

print(predicted_y
)
```

В этом примере мы сначала импортируем необходимые модули из scikit-learn (**LogisticRegression** для модели и **load_iris** для набора данных радужной оболочки). Затем мы загружаем набор данных радужной оболочки, который является хорошо известным набором данных в области машинного обучения, состоящим из 150 образцов цветов радужной оболочки, каждый из которых содержит четыре функции.

Затем мы разделяем набор данных на функции (переменная **X**) и метки (переменная **y**). Мы создаем объект **LogisticRegression** и подгоняем

модель к набору данных, используя функцию **fit**.

Наконец, мы используем обученную модель для прогнозирования меток классов для двух новых наборов функций (**new_X**). Предсказанные метки классов выводятся на консоль.

Это всего лишь простой пример, а у scikit-learn гораздо больше продвинутых функций и моделей для широкого круга задач машинного обучения.

За и против

За:

- Это мощная и всеобъемлющая библиотека машинного обучения, которая предлагает широкий спектр алгоритмов для различных задач.
- Scikit-learn проста в использовании и имеет относительно простой API по сравнению с другими библиотеками машинного обучения.
- Она построена на основе других популярных научных вычислительных библиотек для Python, таких как NumPy, SciPy и matplotlib, что упрощает интеграцию в существующие рабочие процессы анализа данных Python.
- Она предоставляет ряд инструментов для предварительной обработки данных, выбора функций и оценки модели, которые могут помочь оптимизировать рабочий процесс машинного обучения.

- Библиотека Scikit-learn хорошо документирована, содержит исчерпывающие руководства пользователя, справочники по API и большое онлайн-сообщество пользователей.

- Библиотека с открытым исходным кодом и бесплатна для использования, что делает ее доступной для широкого круга пользователей.

Против:

- Хотя scikit-learn предлагает широкий спектр алгоритмов, она может быть не лучшим выбором для некоторых конкретных задач или наборов данных, требующих более специализированных алгоритмов или моделей.

- Это может быть не самая эффективная библиотека для крупномасштабных или сложных задач машинного обучения, поскольку она в основном предназначена для небольших и средних наборов данных.

- Простота API-интерфейса scikit-learn может ограничивать уровень настройки и контроля, которые требуются более продвинутым пользователям.

- Она не включает некоторые новые или более продвинутые методы машинного обучения, разработанные совсем недавно, такие как глубокое обучение.

- Scikit-learn не включает встроенную поддержку некоторых популярных платформ машинного обучения, таких как TensorFlow или PyTorch, что может ограничивать ее гибкость в некоторых случаях использования.

PYTORCH

PyTorch — это популярная библиотека машинного обучения с открытым исходным кодом для языка программирования Python. Она в основном используется для разработки моделей глубокого обучения и предоставляет ряд инструментов и функций для создания, обучения и развертывания нейронных сетей.

Она была разработан исследовательской группой искусственного интеллекта Facebook в 2016 году и быстро стала одной из самых популярных библиотек глубокого обучения в экосистеме Python. Она известна своей гибкостью и простотой использования, что позволяет пользователям создавать и обучать сложные нейронные сети с относительно небольшим количеством строк кода.

PyTorch поддерживает ряд архитектур нейронных сетей, включая сверточные нейронные сети (CNN), рекуррентные нейронные сети (RNN) и преобразователи, а также предоставляет различные алгоритмы оптимизации для обучения этих моделей, в том числе стохастический градиентный спуск (SGD) и алгоритм Адама.

Некоторые из ключевых функций PyTorch включают:

- Автоматическое дифференцирование, которое позволяет пользователям легко вычислять градиенты для моделей нейронных сетей.

- Динамические вычислительные графы, обеспечивающие большую гибкость при построении и модификации нейронных сетей.

- Обширная тензорная библиотека, предоставляющая ряд операций для работы с многомерными массивами.

- Интеграция с популярными библиотеками Python, такими как NumPy и Pandas.

- Большое и активное сообщество пользователей и разработчиков.

PyTorch используется в широком спектре приложений, включая компьютерное зрение, обработку естественного языка и обучение с подкреплением. Она особенно популярна среди исследователей и разработчиков, которые ценят ее гибкость и простоту использования.

Пример фрагмента кода, который демонстрирует, как использовать PyTorch для определения и обучения простой нейронной сети для классификации рукописных цифр:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torchvision import datasets, transforms

# Define a neural network
class Net(nn.Module):
    def __init__(self):
        super(Net, self).__init__()
        self.fc1 = nn.Linear(784, 64)
        self.fc2 = nn.Linear(64, 10)
        self.relu = nn.ReLU()
```

```
    def forward(self, x):
        x = self.relu(self.fc1(x))
        x = self.fc2(x)
        return x

# Load the MNIST dataset
transform =
transforms.Compose([transforms.ToTensor(),

transforms.Normalize((0.1307,), (0.3081,))])
trainset = datasets.MNIST(root='./data',
train=True, download=True, transform=transform)
testset = datasets.MNIST(root='./data',
train=False, download=True,
transform=transform)
trainloader =
torch.utils.data.DataLoader(trainset,
batch_size=32, shuffle=True)
testloader =
torch.utils.data.DataLoader(testset,
batch_size=32, shuffle=False)

# Create a neural network object and an
optimizer
net = Net()
optimizer = optim.SGD(net.parameters(),
lr=0.01, momentum=0.9)

# Train the neural network
criterion = nn.CrossEntropyLoss()
for epoch in range(10):
    running_loss = 0.0
    for i, data in enumerate(trainloader, 0):
        inputs, labels = data
        inputs = inputs.view(-1, 28*28)
        optimizer.zero_grad()
        outputs = net(inputs)
        loss = criterion(outputs, labels)
        loss.backward()
        optimizer.step()
        running_loss += loss.item()
    print(f"Epoch {epoch+1}, Loss:
```

```
{running_loss / len(trainloader)}")

# Test the neural network
correct = 0
total = 0
with torch.no_grad():
    for data in testloader:
        inputs, labels = data
        inputs = inputs.view(-1, 28*28)
        outputs = net(inputs)
        _, predicted = torch.max(outputs.data,
1)
        total += labels.size(0)
        correct += (predicted ==
labels).sum().item()
print(f"Accuracy: {correct / total}")
```

В этом примере мы сначала импортируем необходимые модули из PyTorch, включая **torch**, **torch.nn**, **torch.optim** и **torchvision**. Затем мы определяем простую архитектуру нейронной сети, используя класс **nn.Module**, который включает в себя два полностью связанных слоя и функцию активации ReLU.

Затем мы загружаем набор данных MNIST с помощью модуля **torchvision.datasets** и создаем загрузчики данных для повторения данных во время обучения и тестирования. Мы создаем объект нейронной сети и оптимизатор с помощью модуля **optim** и определяем функцию потерь с помощью класса **nn.CrossEntropyLoss**.

Затем мы обучаем нейронную сеть в течение 10 эпох, используя размер пакета 32, вычисляя потери с использованием указанного критерия и применяя обратное распространение градиентов с помощью

оптимизатора.

Наконец, мы тестируем обученную нейронную сеть, используя тестовые данные, вычисляя точность модели на тестовом наборе.

Это всего лишь простой пример, а у PyTorch гораздо больше продвинутых функций и моделей для широкого круга задач глубокого обучения.

За и против

За:

- Она известна своей простотой использования и гибкостью, что позволяет пользователям быстро создавать и обучать сложные нейронные сети с относительно небольшим количеством строк кода.
- Она включает в себя ряд инструментов и функций глубокого обучения, в том числе автоматическое дифференцирование, динамические вычислительные графы и обширную тензорную библиотеку.
- Хорошо интегрируется с другими популярными библиотеками Python, такими как NumPy и Pandas, что упрощает их использование в сочетании с другими инструментами анализа данных и машинного обучения.
- Имеет большое и активное сообщество пользователей и разработчиков, а это означает, что существует серьезная поддержка и

множество ресурсов для пользователей, которые впервые знакомятся с библиотекой или которым нужна помощь в решении более сложных задач.

- PyTorch широко используется как в научных кругах, так и в промышленности, а также для достижения самых современных результатов в различных задачах глубокого обучения.

Против:

- Она может быть не такой быстрой и эффективной, как другие библиотеки глубокого обучения, такие как TensorFlow или Keras, особенно для крупномасштабного распределенного обучения.
- Может быть менее стабильной, чем другие библиотеки глубокого обучения, что может затруднить отладку ошибок или воспроизведение результатов.
- Для эффективного использования может потребовать больше знаний и опыта, чем для других библиотек глубокого обучения, особенно для пользователей, которые плохо знакомы с машинным обучением или не знакомы с программированием на Python.

TENSORFLOW

TensorFlow — популярная библиотека машинного обучения с открытым исходным кодом, разработанная Google. В основном она используется для создания и обучения глубоких нейронных сетей, хотя она также включает в себя ряд инструментов и функций, предназначенных для других задач машинного обучения.

Первоначально была разработана командой Google Brain для внутреннего использования, но позже, в 2015 году, была опубликована как библиотека с открытым исходным кодом. С тех пор она стала одной из наиболее широко используемых и уважаемых библиотек машинного обучения с большим и активным сообществом пользователей и разработчиков.

TensorFlow создана так, чтобы быть гибкой и масштабируемой, позволяя пользователям создавать и обучать глубокие нейронные сети на широком спектре оборудования: от ноутбуков и мобильных устройств до крупномасштабных распределенных кластеров. Она включает в себя обширную тензорную библиотеку для эффективных численных вычислений, а также ряд API-интерфейсов высокого уровня для построения и обучения нейронных сетей.

Она также включает в себя ряд инструментов и функций для предварительной обработки, визуализации и анализа данных, что делает ее комплексной и мощной платформой машинного

обучения. TensorFlow широко используется как в научных кругах, так и в промышленности для достижения самых современных результатов в различных задачах машинного обучения, включая распознавание изображений, обработку естественного языка и многое другое.

В целом, TensorFlow — это мощная и гибкая библиотека машинного обучения, которая широко используется и пользуется уважением в сообществе машинного обучения.

Пример фрагмента кода, который использует TensorFlow для обучения простой нейронной сети классификации рукописных цифр из набора данных MNIST:

```
import tensorflow as tf
from tensorflow.keras.datasets import mnist

# Load the MNIST dataset
(x_train, y_train), (x_test, y_test) =
mnist.load_data()

# Preprocess the data
x_train = x_train / 255.0
x_test = x_test / 255.0

# Define the model architecture
model = tf.keras.models.Sequential([
    tf.keras.layers.Flatten(input_shape=(28,
28)),
    tf.keras.layers.Dense(128,
activation='relu'),
    tf.keras.layers.Dense(10)
])

# Compile the model
model.compile(optimizer='adam',

loss=tf.keras.losses.SparseCategoricalCrossentr
opy(from_logits=True),
              metrics=['accuracy'])

# Train the model
model.fit(x_train, y_train, epochs=10,
validation_data=(x_test, y_test))

# Evaluate the model
test_loss, test_acc = model.evaluate(x_test,
y_test, verbose=2)
print('Test accuracy:', test_acc)
```

Этот код сначала загружает набор данных MNIST и предварительно обрабатывает данные, масштабируя их до диапазона [0, 1]. Затем он определяет простую архитектуру нейронной сети с использованием API Keras в TensorFlow с одним скрытым слоем, содержащим 128 нейронов и активацией ReLU, и выходным слоем, содержащим 10 нейронов (по одному на каждую возможную цифру). Затем модель компилируется с использованием оптимизатора Adam и разреженной категориальной кроссэнтропийной потери и обучается в течение 10 эпох на обучающих данных, при этом проверка выполняется на тестовых данных после каждой эпохи. Наконец, модель оценивается на основе тестовых данных, а точность теста распечатывается.

Обратите внимание, что это всего лишь простой пример — TensorFlow способен создавать гораздо более сложные модели и архитектуры для широкого спектра задач машинного обучения.

За и против

За

- Широко считается одной из самых мощных и гибких доступных библиотек машинного обучения с набором инструментов и функций для создания и обучения сложных нейронных сетей.

- У нее большое и активное сообщество пользователей и разработчиков, а это означает, что существует серьезная поддержка и множество ресурсов для пользователей, которые впервые познакомились с библиотекой или которым нужна помощь в решении более сложных задач.

- Она создана так, чтобы быть масштабируемой и эффективной, позволяя пользователям создавать и обучать модели на широком спектре оборудования, от ноутбуков до крупномасштабных распределенных кластеров.

- Включает комплексную тензорную библиотеку для эффективных численных вычислений, а также ряд API-интерфейсов высокого уровня для построения и обучения нейронных сетей.

- Широко используется как в научных кругах, так и в промышленности для достижения самых современных результатов в различных задачах машинного обучения, включая распознавание изображений, обработку естественного языка и многое другое.

Против

- Может иметь более крутую кривую обучения, чем другие библиотеки машинного обучения, особенно для пользователей, которые плохо знакомы с

глубоким обучением или не знакомы с программированием на Python.

- Она может быть менее интуитивно понятной, чем другие библиотеки машинного обучения, с более многословным и сложным синтаксисом.

- Низкоуровневый API TensorFlow может потребовать больше кода для выполнения простых задач, чем другие библиотеки машинного обучения, что может сделать его менее привлекательным для прототипирования или экспериментирования.

- Архитектура вычислительного графа может затруднить отладку, особенно для пользователей, не знакомых с внутренним устройством библиотеки.

- Архитектура вычислительных графов TensorFlow может затруднить интеграцию с другими библиотеками Python, хотя в последних релизах ситуация улучшилась.

XGBOOST

XGBoost — это программная библиотека с открытым исходным кодом, которая обеспечивает среду повышения градиента для машинного обучения. Он был разработан Тяньци Ченом и его коллегами из Вашингтонского университета и в настоящее время поддерживается DMLC. XGBoost спроектирован так, чтобы быть масштабируемым, портативным и эффективным, что делает его популярным для использования в широком спектре приложений, включая задачи прогнозирования, классификации и ранжирования в промышленности и научных кругах.

В Python XGBoost можно использовать через библиотеку **xgboost**, которая предоставляет API для определения, обучения и оценки моделей XGBoost. Библиотека построена на основе базовой библиотеки C++ XGBoost, которая обеспечивает быструю и эффективную реализацию повышения градиента.

Он работает путем итеративного добавления деревьев решений в модель, при этом каждое дерево обучено исправлять ошибки предыдущих деревьев. Алгоритм объединяет прогнозы всех деревьев для получения окончательного прогноза. XGBoost использует различные методы оптимизации, включая регуляризацию и параллельную обработку, для повышения точности и быстродействия модели.

Она стала популярным инструментом для использования в соревнованиях по машинному обучению благодаря своей способности достигать самых современных результатов в широком спектре

задач.

Пример использования библиотеки XGBoost на Python для обучения простой модели повышения градиента на популярном наборе данных Iris:

```
import numpy as np
import pandas as pd
import xgboost as xgb
from sklearn.datasets import load_iris
from sklearn.model_selection import
train_test_split

# Load the iris dataset and split into training
and testing sets
iris = load_iris()
X_train, X_test, y_train, y_test =
train_test_split(iris.data, iris.target,
test_size=0.2, random_state=42)

# Define the XGBoost model
xgb_model =
xgb.XGBClassifier(objective="multi:softmax",
num_class=3)

# Train the model on the training data
xgb_model.fit(X_train, y_train)

# Make predictions on the testing data
y_pred = xgb_model.predict(X_test)

# Evaluate the accuracy of the model
accuracy = np.sum(y_pred == y_test) /
len(y_test)
print("Accuracy: {:.2f}%".format(accuracy *
100))
```

В этом примере мы начинаем с загрузки набора данных Iris с помощью встроенной функции **load_iris** scikit-learn. Затем мы разделяем набор данных на наборы для обучения и тестирования, используя функцию **train_test_split** в scikit-learn.

Далее мы определяем модель классификатора XGBoost, используя класс **xgb.XGBClassifier**. В этом случае мы устанавливаем для параметра **objective** значение **"multi:softmax"**, а для параметра **num_class** — значение **3**, поскольку в наборе данных Iris у нас есть три класса.

Затем мы обучаем модель XGBoost на обучающих данных, используя метод **fit**. После обучения модели мы используем ее для прогнозирования данных тестирования с помощью метода **predict**.

Наконец, мы оцениваем точность модели, сравнивая предсказанные метки с истинными метками в тестовом наборе и распечатываем показатель точности.

За и против

За

- Известна своей точностью и производительностью, особенно в задачах структурированных данных, таких как классификация, регрессия и ранжирование.
- Она предоставляет несколько методов регуляризации, таких как регуляризация L1 и L2, чтобы помочь предотвратить переобучение и улучшить обобщение модели.
- Поддерживает параллельную обработку на нескольких процессорах, что позволяет эффективно обрабатывать большие наборы данных.

- Библиотека предоставляет множество гиперпараметров, которые можно настроить для повышения производительности модели и адаптации к различным вариантам использования.
- XGBoost — это библиотека с открытым исходным кодом, имеющая активное сообщество разработчиков, что означает, что она постоянно обновляется и улучшается.

Против

- Предназначена в первую очередь для структурированных данных и может быть не столь эффективна для неструктурированных данных, таких как текстовые или графические данные.
- Процесс настройки гиперпараметров может занять много времени и требует определенного уровня знаний для эффективной оптимизации модели.
- Может не подойти для обучения в режиме реального времени или онлайн-обучения, поскольку требует переобучения всей модели каждый раз при добавлении новых данных.
- Поскольку XGBoost является алгоритмом повышения градиента, она подвержена тем же проблемам, что и другие алгоритмы повышения градиента, например, склонность к переобучению и требование к тщательной регуляризации, чтобы предотвратить это.

LIGHTGBM

Python LightGBM — это повышение градиента среды, использующее старые алгоритмы обучения. Это мощная библиотека машинного обучения, разработанная Microsoft и предназначенная для эффективной и быстрой работы. LightGBM означает «Light Gradient Boosting Machine». Она была разработана для обработки крупномасштабных данных и может обрабатывать миллионы строк и тысячи функций.

LightGBM отличается от других библиотек повышения градиента, таких как XGBoost, использованием новой техники, называемой «Односторонняя выборка на основе градиента» (GOSS) и «Объединение эксклюзивных функций» (EFB). Эти методы помогают сократить вычислительные ресурсы, необходимые для обучения модели, и ускорить процесс обучения.

Она поддерживает различные типы учебных задач, такие как регрессия, классификация и ранжирование. Она также имеет множество полезных функций, таких как встроенная перекрестная проверка, ранняя остановка и поддержка категориальных функций.

В целом, LightGBM — это мощная библиотека для повышения градиента и отличный выбор для обработки крупномасштабных структурированных данных.

Пример использования кода LightGBM на Python для задачи двоичной классификации:

```
import lightgbm as lgb
```

```
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import
train_test_split
from sklearn.metrics import accuracy_score

# Load the breast cancer dataset
data = load_breast_cancer()

# Split the data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(data.data, data.target,
test_size=0.2, random_state=42)

# Convert the data into LightGBM's dataset
format
train_data = lgb.Dataset(X_train,
label=y_train)
test_data = lgb.Dataset(X_test, label=y_test)

# Set the hyperparameters for the LightGBM
model
params = {
    'objective': 'binary',
    'metric': 'binary_logloss',
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9
}

# Train the LightGBM model on the training data
num_rounds = 100
model = lgb.train(params, train_data,
num_rounds)

# Make predictions on the testing data
y_pred = model.predict(X_test)
y_pred = [1 if x >= 0.5 else 0 for x in y_pred]

# Evaluate the accuracy of the model
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: {:.2f}%".format(accuracy *
100))
```

В этом примере мы начинаем с загрузки набора данных о раке молочной железы с помощью функции **load_breast_cancer** scikit-learn. Затем мы разделяем набор данных на наборы для обучения и тестирования, используя функцию **train_test_split** в scikit-learn.

Затем мы конвертируем данные обучения и тестирования в формат набора данных LightGBM, используя класс **lgb.Dataset**. Затем мы устанавливаем гиперпараметры для модели LightGBM, включая целевую функцию, метрику оценки, количество листьев в каждом дереве, скорость обучения и долю признаков.

Затем мы обучаем модель LightGBM на обучающих данных с помощью функции **lgb.train**. После обучения модели мы используем ее для прогнозирования данных тестирования, вызывая метод **predict**. Затем мы конвертируем предсказанные вероятности в двоичные предсказания, устанавливая порог 0.5.

Наконец, мы оцениваем точность модели, сравнивая предсказанные метки с истинными метками в тестовом наборе и распечатываем показатель точности.

За и против

За

- LightGBM предназначена для обработки крупномасштабных данных и может эффективно обрабатывать миллионы строк и

тысячи функций.

- Она использует новую технику под названием «Односторонняя выборка на основе градиента» (GOSS) и «Объединение эксклюзивных функций» (EFB) для сокращения вычислительных ресурсов, необходимых для обучения модели, и ускорения процесса обучения.

- Она поддерживает различные типы учебных задач, такие как регрессия, классификация и ранжирование.

- Она имеет множество полезных функций, таких как встроенная перекрестная проверка, ранняя остановка и поддержка категориальных функций.

- Она имеет хорошую точность и производительность по сравнению с другими системами повышения градиента.

Против

- LightGBM может потребовать некоторой настройки гиперпараметров для достижения оптимальных результатов.

- Ее может быть сложнее использовать и понимать по сравнению с более простыми алгоритмами машинного обучения, особенно для новичков.

- Библиотека по умолчанию не поддерживает ускорение графического процессора, что может быть недостатком в некоторых случаях

использования, когда желательно ускорение графического процессора. Однако есть способы использовать LightGBM с ускорением графического процессора через сторонние библиотеки.

KERAS

Keras — это API нейронных сетей высокого уровня, написанный на Python и способный работать поверх популярных фреймворков глубокого обучения, таких как TensorFlow. Keras был разработан для быстрого экспериментирования с глубокими нейронными сетями и стал одной из самых популярных библиотек глубокого обучения. Он особенно хорошо подходит для создания и обучения моделей глубокого обучения для задач компьютерного зрения и обработки естественного языка (НЛП). Keras имеет открытый исходный код и поддерживается сообществом участников на GitHub.

Пример кода для построения простой нейронной сети с использованием Keras:

```
import numpy as np
from keras.models import Sequential
from keras.layers import Dense

# Generate some dummy data for training and
testing
x_train = np.random.random((1000, 10))
y_train = np.random.randint(2, size=(1000, 1))
x_test = np.random.random((100, 10))
y_test = np.random.randint(2, size=(100, 1))

# Build the model
model = Sequential()
model.add(Dense(32, input_dim=10,
activation='relu'))
model.add(Dense(1, activation='sigmoid'))

# Compile the model
model.compile(optimizer='rmsprop',
loss='binary_crossentropy',
metrics=['accuracy'])
```

```
# Train the model
model.fit(x_train, y_train, epochs=20,
batch_size=32)

# Evaluate the model on the test data
score = model.evaluate(x_test, y_test,
batch_size=128)

# Print the test loss and accuracy
print('Test loss:', score[0])
print('Test accuracy:', score[1])
```

Этот код определяет простую нейронную сеть с одним скрытым слоем из 32 нейронов и выходным слоем с одним нейроном, который используется для бинарной классификации. Модель компилируется с использованием двоичной функции кроссэнтропийных потерь и оптимизатора RMSprop. Затем она обучается на случайно сгенерированных обучающих данных для 20 эпох с размером пакета 32. Наконец, модель оценивается на некоторых случайно сгенерированных тестовых данных, и распечатываются тестовые потери и точность.

За и против

За

- Удобный API: Keras предоставляет простой и интуитивно понятный интерфейс, который упрощает использование и понимание, особенно для новичков в глубоком обучении.
- Модульная и гибкая архитектура: Keras позволяет пользователям создавать модели путем объединения нескольких слоев, которые

можно легко добавлять или удалять, что позволяет быстро экспериментировать и создавать прототипы.

- Широкий спектр приложений: Keras поддерживает множество задач глубокого обучения, таких как классификация изображений, обработка естественного языка и прогнозирование временных рядов.

- Эффективные вычисления: Keras может работать как на центральных, так и на графических процессорах, обеспечивая быстрые вычисления для больших наборов данных.

Против

- Ограниченная гибкость: хотя Keras отлично подходит для создания прототипов и экспериментов, он может не обеспечивать тот уровень гибкости и контроля, который необходим для более сложных моделей глубокого обучения.

- Меньше настроек: Keras абстрагирует многие детали реализации более низкого уровня, что может ограничить возможности настройки для опытных пользователей.

- Ограниченная обратная совместимость: Keras со временем претерпел некоторые существенные изменения, из-за чего может быть сложно поддерживать обратную совместимость между различными версиями.

- Ограниченная поддержка распределенного обучения. Хотя Keras можно использовать для распределенного обучения, он может быть не таким эффективным, как другие среды глубокого обучения, специально разработанные для распределенных вычислений.

PYCARET

PyCaret — это библиотека машинного обучения с открытым исходным кодом на Python, которая автоматизирует сквозной процесс машинного обучения. Она спроектирована как простая в использовании библиотека, требующая минимальных усилий по кодированию и одновременно обеспечивающая максимальную гибкость и контроль для пользователя. PyCaret имеет широкий спектр функций, включая предварительную обработку данных, классификацию, регрессию, кластеризацию, обнаружение аномалий, обработку естественного языка, прогнозирование временных рядов и развертывание моделей.

Она построена на основе популярных библиотек машинного обучения, таких как scikit-learn, XGBoost, LightGBM, CatBoost и spaCy. Она предоставляет API высокого уровня, который упрощает сложные рабочие процессы машинного обучения за счет автоматизации повторяющихся задач, таких как предварительная обработка данных, настройка гиперпараметров, выбор модели и построение ансамбля.

PyCaret особенно полезен для специалистов по данным и специалистов по машинному обучению, которые хотят быстро создавать и прототипировать модели машинного обучения, не тратя много времени на предварительную обработку данных и выбор модели. Он также подходит для бизнес-аналитиков и инженеров данных, которые хотят исследовать и анализировать данные с использованием методов

машинного обучения.

Пример использования кода PyCaret:

```
from pycaret.datasets import get_data
from pycaret.classification import *

# load data
data = get_data('diabetes')

# setup model
clf = setup(data, target='Class variable')

# compare models
compare_models()
```

В этом примере мы сначала импортируем функцию **get_data** из **pycaret.datasets** и функцию **setup**, а также функцию **compare_models** из **pycaret.classification**. Затем мы загружаем набор данных 'diabetes', используя **get_data**, и настраиваем модель классификации с помощью **setup**, указав в качестве целевой переменной 'Class variable'. Наконец, мы сравниваем производительность различных моделей классификации, используя **compare_models**.

Обратите внимание, что **setup** автоматически предварительно обрабатывает данные, выполняет разработку и выбор функций, а также настраивает среду обучения и тестирования. **compare_models** возвращает таблицу перекрестно проверенных показателей производительности для каждой модели, которую можно использовать для выбора наиболее эффективной модели для дальнейшей настройки и оценки.

За и против

За

- Предоставляет широкий спектр встроенных функций для подготовки данных, обучения модели, настройки гиперпараметров и развертывания модели, что позволяет легко создавать и тестировать модели машинного обучения.

- Поддерживает широкий спектр моделей и алгоритмов машинного обучения, включая методы обучения с учителем и без учителя.

- Предоставляет обширную документацию и примеры, упрощающие изучение и использование как новичками, так и опытными специалистами по машинному обучению.

- PyCaret предоставляет веб-интерфейс для создания и развертывания моделей машинного обучения, что может быть особенно полезно для нетехнических пользователей, которые хотят использовать машинное обучение без необходимости писать какой-либо код.

Против

- Простота использования и встроенные функциональные возможности PyCaret могут достигаться за счет гибкости и настраиваемости, особенно для сложных задач машинного обучения, требующих более сложных методов обработки данных или

моделирования.

- Может представлять не лучший выбор для крупномасштабных или высокопроизводительных задач машинного обучения, требующих специализированного оборудования или программного обеспечения.

- Это относительно новая библиотека, поэтому она может не иметь такого же уровня поддержки сообщества или сторонней интеграции, как более известные библиотеки машинного обучения.

MLOPS

MLOps (Machine Learning Operations) — это набор практик и инструментов, которые оптимизируют жизненный цикл разработки машинного обучения (ML), от разработки до развертывания и обслуживания.

Это похоже на DevOps — набор методов разработки, развертывания и обслуживания программных приложений. Однако MLOps адаптирован к конкретным потребностям и задачам разработки и внедрения моделей машинного обучения.

Он включает в себя различные задачи, в том числе подготовку и парсинг данных, обучение и проверку модели, развертывание и обслуживание модели, а также мониторинг и обслуживание. Это также требует сотрудничества между различными командами, такими как специалисты по данным, инженеры по машинному обучению, разработчики программного обеспечения и рабочие группы.

Инструменты и методы MLOps включают системы контроля версий, конвейеры непрерывной интеграции и развертывания (CI/CD), контейнеризацию, инструменты оркестрации, а также инструменты мониторинга и квитирования. Внедряя MLOps, организации могут повысить производительность, масштабируемость и надежность своих систем машинного обучения, а также снизить риск ошибок или сбоев в производстве.

MLFLOW

MLflow — это платформа с открытым исходным кодом для управления и отслеживания экспериментов по машинному обучению. Он предоставляет простой и гибкий интерфейс для отслеживания экспериментов, упаковки кода в воспроизводимые прогоны, а также совместного использования и развертывания моделей.

MLflow был разработан и выпущен в 2018 году, как проект Databricks с открытым исходным кодом. Цель MLflow — упростить жизненный цикл машинного обучения, предоставляя стандартизированный способ управления и отслеживания экспериментов, а также упаковки и развертывания моделей. MLflow можно использовать с различными библиотеками и платформами машинного обучения, включая TensorFlow, PyTorch и scikit-learning.

MLflow состоит из нескольких компонентов:

1. Отслеживание: модуль для регистрации и отслеживания экспериментов, включая параметры, метрики и артефакты.

2. Проекты: формат для упаковки кода науки о данных многоразовым и воспроизводимым способом.

3. Модели: формат упаковки моделей машинного обучения таким образом, чтобы их можно было легко развернуть в различных производственных средах.

4. Реестр моделей: централизованное хранилище для управления моделями, включая управление версиями, переходы между этапами и контроль доступа.

MLflow также предоставляет интерфейс командной строки и API для интеграции с другими инструментами и рабочими процессами. В целом, MLflow стремится упростить процесс разработки, обучения и развертывания моделей машинного обучения, одновременно улучшая сотрудничество и воспроизводимость.

Пример фрагмента кода, использующего MLflow для отслеживания и регистрации показателей во время эксперимента по машинному обучению:

```
import mlflow
import numpy as np
from sklearn.linear_model import
LinearRegression

# Start an MLflow experiment
mlflow.set_experiment("linear-regression")

# Generate some random data
x = np.random.rand(100, 1)
y = 2*x + np.random.randn(100, 1)

# Define a model
model = LinearRegression()

# Train the model
model.fit(x, y)

# Log some metrics
mlflow.log_metric("r2_score", model.score(x,
y))
mlflow.log_metric("mse",
np.mean((model.predict(x) - y) ** 2))
```

```
# Save the model
mlflow.sklearn.log_model(model, "model")

# End the experiment
mlflow.end_experiment()
```

В этом примере мы сначала запускаем эксперимент MLflow, вызывая **mlflow.set_experiment** с наименованием эксперимента. Затем мы генерируем некоторые случайные данные и определяем модель линейной регрессии с помощью scikit-learn. Мы обучаем модель на данных, а затем используем MLflow для регистрации некоторых показателей (оценка R-квадрат и среднеквадратическая ошибка) с помощью **mlflow.log_metric**. Мы также сохраняем обученную модель, используя **mlflow.sklearn.log_model**. Наконец, мы завершаем эксперимент, используя **mlflow.end_experiment**.

Запустив этот код, мы можем использовать пользовательский интерфейс MLflow для просмотра и сравнения результатов нескольких экспериментов, включая зарегистрированные метрики и обученные модели.

За и против

За

1. Воспроизводимость: MLflow предоставляет стандартизированный способ отслеживания экспериментов, пакетов и моделей развертывания, что может помочь гарантировать воспроизводимость экспериментов.

2. Гибкость. MLflow можно использовать с различными библиотеками и платформами машинного обучения, включая TensorFlow, PyTorch и scikit-learn, что делает его универсальным инструментом для управления проектами машинного обучения.

3. Сотрудничество: MLflow предоставляет централизованную платформу для обмена экспериментами, моделями и кодом, что может улучшить сотрудничество между учеными и разработчиками данных.

4. Визуализация: MLflow предоставляет веб-интерфейс для визуализации и сравнения экспериментов, который может помочь в отладке и оптимизации.

Против

1. Кривая обучения: для эффективного использования MLflow требуется некоторое обучение, включая знание API MLflow и способов его интеграции с существующими рабочими процессами.

2. Накладные расходы: использование MLflow требует некоторых дополнительных накладных расходов по сравнению с простым проведением экспериментов и отслеживанием результатов вручную, хотя эти накладные расходы обычно минимальны.

3. Ограничения. Хотя MLflow является мощным инструментом, он может не отвечать всем потребностям конкретного проекта, например специальным требованиям к развертыванию

модели или обучению.

MLflow может быть ценным инструментом для управления и отслеживания проектов машинного обучения, особенно в средах, где важны сотрудничество и воспроизводимость. Однако, как и любой инструмент, он имеет свои сильные и слабые стороны, и его следует оценивать с учетом конкретных потребностей проекта.

KUBEFLOW

Kubeflow — это платформа с открытым исходным кодом для запуска рабочих процессов машинного обучения в Kubernetes. Kubernetes — это платформа оркестрации контейнеров, которая обеспечивает масштабируемую и отказоустойчивую инфраструктуру для развертывания распределенных приложений и управления ими. Kubeflow построен на основе Kubernetes и предоставляет платформу для развертывания, масштабирования и управления рабочими процессами машинного обучения.

Kubeflow предоставляет ряд инструментов и платформ для создания и развертывания моделей машинного обучения, в том числе:

1. Блокноты Jupyter: веб-среда для интерактивного анализа данных и разработки моделей.
2. TensorFlow: популярная библиотека машинного обучения для создания и обучения глубоких нейронных сетей.
3. PyTorch: популярная библиотека машинного обучения для создания и обучения глубоких нейронных сетей.
4. Apache Spark: среда распределенных вычислений для обработки больших наборов данных.
5. Apache Beam: унифицированная модель программирования для пакетной и потоковой

обработки данных.

Kubeflow также предоставляет ряд компонентов для управления рабочим процессом машинного обучения, в том числе:

1. Конвейеры: инструмент для создания, развертывания и управления конвейерами машинного обучения.

2. Обучение: инструмент для управления распределенными учебными заданиями в Kubernetes.

3. Обслуживание: инструмент для развертывания и обслуживания обученных моделей в виде веб-сервисов.

4. Метаданные: инструмент для отслеживания и управления метаданными, связанными с экспериментами по машинному обучению.

Kubeflow предоставляет мощную платформу для создания и развертывания рабочих процессов машинного обучения в Kubernetes. Используя масштабируемость и отказоустойчивость Kubernetes, Kubeflow может помочь оптимизировать рабочий процесс машинного обучения и улучшить воспроизводимость и масштабируемость моделей машинного обучения.

Пример фрагмента кода, демонстрирующий, как использовать Kubeflow для обучения модели TensorFlow в кластере Kubernetes:

```
import kfp
import kfp.dsl as dsl
import kfp.components as comp
```

```
# Define the pipeline
@dsl.pipeline(name='train-tf-model',
description='Trains a TensorFlow model on
Kubernetes')
def train_pipeline(
    data_path: str,
    model_path: str,
    epochs: int,
    batch_size: int,
    learning_rate: float
):
    # Load the data
    load_data = dsl.ContainerOp(
        name='load_data',
        image='my-registry/my-image',
        command=['python',
'/app/load_data.py'],
        arguments=[
            '--data-path', data_path,
            '--output-path',
'/mnt/data/raw_data.csv'
        ]
    )

    # Preprocess the data
    preprocess = dsl.ContainerOp(
        name='preprocess',
        image='my-registry/my-image',
        command=['python',
'/app/preprocess.py'],
        arguments=[
            '--data-path',
'/mnt/data/raw_data.csv',
            '--output-path',
'/mnt/data/cleaned_data.csv'
        ]
    ).after(load_data)

    # Train the model
    train = dsl.ContainerOp(
        name='train',
        image='my-registry/my-image',
        command=['python', '/app/train.py'],
        arguments=[
```

```
            '--train-data',
'/mnt/data/cleaned_data.csv',
            '--model-dir', model_path,
            '--epochs', epochs,
            '--batch-size', batch_size,
            '--learning-rate', learning_rate
        ]
    ).after(preprocess)

# Compile the pipeline
pipeline_func = train_pipeline
pipeline_filename = pipeline_func.  name   +
'.yaml'
kfp.compiler.Compiler().compile(pipeline_func,
pipeline_filename)

# Define the Kubeflow experiment
experiment_name = 'train-tf-model'
run_name = pipeline_func.  name__ + ' run'
client = kfp.Client()
```

```
# Define the pipeline parameters
params = {
    'data_path': 'gs://my-bucket/my-
    data.csv',
    'model_path': 'gs://my-bucket/my-model',
    'epochs': 10,
    'batch_size': 32,
    'learning_rate': 0.001
}

# Submit the pipeline to the Kubeflow cluster
try:
    experiment =
client.create_experiment(name=experiment_name
) except kfp.errors.ApiException:
    experiment =
client.get_experiment(experiment_name
) run =
client.run_pipeline(experiment.id,
run_name, pipeline_filename, params)
```

В этом примере мы определяем конвейер Kubeflow, который состоит из двух компонентов: компонента для предварительной обработки данных и компонента для обучения модели. Затем мы определяем сам конвейер, который принимает в качестве входных данных путь к необработанным данным, путь, по которому будет сохранена обученная модель, и различные гиперпараметры для процесса обучения. Конвейер сначала предварительно обрабатывает данные с помощью компонента **preprocess_op**, затем обучает модель с помощью компонента **train_op**. Наконец, мы компилируем конвейер и отправляем его в кластер Kubeflow, используя класс **kfp.Client**.

Запустив этот код, мы можем обучить модель TensorFlow в кластере Kubernetes с помощью Kubeflow, а также воспользоваться преимуществами масштабируемости, отказоустойчивости и воспроизводимости, обеспечиваемыми Kubernetes.

За и против

За

1. Масштабируемость. Kubeflow предназначен для работы с Kubernetes, который обеспечивает масштабируемую и распределенную среду для выполнения рабочих функций машинного обучения. Это означает, что Kubeflow можно легко масштабировать для обработки больших наборов данных и сложных моделей.

2. Воспроизводимость. Kubeflow позволяет создавать воспроизводимые конвейеры для рабочих процессов машинного обучения, что гарантирует повторяемость ваших экспериментов и надежность результатов. Это связано с тем, что Kubeflow позволяет легко отслеживать и управлять версиями ваших данных, кода и конфигураций.

3. Переносимость. Kubeflow позволяет создавать конвейеры машинного обучения, которые можно запускать в любом

кластере Kubernetes, локальном или облачном. Это означает, что вы можете легко перемещать рабочие функции машинного обучения между различными средами без необходимости изменения кода или конфигурации.

4. Настраиваемость. Kubeflow предоставляет ряд готовых компонентов для общих задач машинного обучения, но также позволяет создавать собственные компоненты с использованием любого языка программирования или инструмента. Это позволяет легко адаптировать конвейеры машинного обучения к вашим конкретным потребностям.

Против

1. Сложность. Kubeflow — это сложная система, для запуска которой требуется значительный объем конфигурации и настройки. Это может стать барьером для входа в систему для небольших команд или организаций, у которых нет выделенных ресурсов DevOps.

2. Кривая обучения. Kubeflow — это относительно новая технология, поэтому ее кривая обучения требует интенсивного подхода. Это означает, что командам может потребоваться некоторое время, чтобы освоить использование Kubeflow для своих рабочих процессов машинного обучения.

3. Требования к ресурсам. Поскольку Kubeflow предназначен для работы в Kubernetes, для его эффективной работы требуется значительный объем ресурсов. Это означает, что командам потребуется доступ к кластеру Kubernetes, что может стать проблемой для небольших организаций или команд без выделенных ресурсов DevOps.

4. Управление версиями. Хотя Kubeflow предоставляет инструменты для управления версиями данных и кода, могут возникнуть проблемы с моделями и конфигурациями управления версиями. Это может затруднить отслеживание изменений в моделях с течением времени и обеспечение воспроизводимости моделей.

ZENML

ZENML — это платформа MLOps с открытым исходным кодом, которая обеспечивает конвейерный подход для управления сквозными рабочими процессами машинного обучения. ZENML предназначена для упрощения разработки и развертывания моделей машинного обучения, предоставляя API высокого уровня для распространенных задач машинного обучения.

Она построена на основе TensorFlow и предназначена для полной интеграции с популярными библиотеками машинного обучения, такими как scikit-learn и PyTorch. ZENML поддерживает ряд источников данных и методов предварительной обработки, а также предоставляет ряд предварительно созданных компонентов для общих задач машинного обучения, таких как проверка данных, разработка функций и обучение моделей.

Она также предоставляет ряд функций для управления развертыванием и мониторингом моделей машинного обучения, включая поддержку управления версиями моделей, A/B-тестирование и автоматическое переобучение моделей.

ZENML создана так, чтобы быть гибкой и настраиваемой, позволяя пользователям создавать собственные компоненты с использованием любого языка программирования или инструмента. ZENML также предоставляет обширную документацию и ряд учебных пособий, которые помогут пользователям начать работу с платформой.

Пример использования кода ZENML для рабочего процесса MLOps:

```
from zenml.core import SimplePipeline
from zenml.datasources import CSVDatasource
from zenml.steps.evaluator.tf_evaluator import TFEvaluator
from zenml.steps.preprocesser.standard_scaler import StandardScaler
from zenml.steps.splitter.random_split import RandomSplit
from zenml.steps.trainer.tf_trainer import TFTrainer
from zenml.backends.orchestrator.tf_local_orchestrator import TFLocalOrchestrator

# Define data source
ds = CSVDatasource(name='my-csv-datasource', path='./my-dataset.csv')

# Define splitter
split = RandomSplit(split_map={'train': 0.7, 'eval': 0.2, 'test': 0.1})

# Define preprocesser
preprocesser = StandardScaler()

# Define trainer
trainer = TFTrainer(
    loss='categorical_crossentropy',
    last_activation='softmax',
    epochs=10,
    batch_size=32
)

# Define evaluator
evaluator = TFEvaluator()
```

```
# Define pipeline
pipeline = SimplePipeline(
    datasource=ds,
    splitter=split,
    preprocesser=preprocesser
    , trainer=trainer,
    evaluator=evaluator,
    name='my-pipeline'
)

#              Define
orchestrator
orchestrator                          =
TFLocalOrchestrator()

#          Run
pipeline
orchestrator.run(pipeline)
```

В этом примере мы сначала определяем **CSVDatasource**, который указывает на CSV-файл, содержащий наш набор данных. Затем мы определяем разделитель **RandomSplit**, чтобы разделить набор данных на наборы обучения, оценки и тестирования.

Далее мы определяем препроцессор **StandardScaler** для стандартизации функций в наборе данных. Затем мы определяем **TFTrainer** для обучения модели TensorFlow на предварительно обработанных данных.

Мы также определяем **TFEvaluator** для оценки обученной модели по набору оценок.

Наконец, мы создаем объект **SimplePipeline**, который включает в себя все определенные шаги, и определяем **TFLocalOrchestrator** для локального

запуска конвейера.

Затем мы запускаем конвейер с помощью команды **orchestrator.run(pipeline)**. Это приведет к выполнению шагов конвейера в определенном порядке и выведению результатов конвейера. Затем этот конвейер можно управлять версиями, развертывать и управлять им с помощью фреймворка ZENML.

За и против

За

- Конвейерный подход: ZENML обеспечивает конвейерный подход к управлению сквозными рабочими процессами машинного обучения, упрощая создание, тестирование и развертывание моделей машинного обучения.

- Гибкость: ZENML создан так, чтобы быть гибким и настраиваемым, позволяя пользователям создавать собственные компоненты с использованием любого языка программирования или инструмента. Это упрощает интеграцию ZENML с другими инструментами и библиотеками, которые вы, возможно, уже используете.

- Масштабируемость: ZENML создан таким образом, чтобы его можно было масштабировать, и его можно запускать в различных вычислительных средах, от одной машины до распределенного кластера.

- Интеграция с TensorFlow: ZENML построен на основе TensorFlow, одной из самых популярных библиотек глубокого обучения. Это упрощает включение моделей TensorFlow в ваши конвейеры ZENML и предоставляет ряд готовых компонентов TensorFlow, которые можно использовать в ваших конвейерах.

- Открытый исходный код: ZENML — это платформа с открытым исходным кодом, что означает, что каждый может свободно использовать, изменять и вносить свой вклад в развитие.

Против

- Кривая обучения: как и любой новый инструмент или библиотека, использование ZENML может потребовать обучения, особенно если вы не знакомы с конвейерным подходом к управлению рабочими процессами машинного обучения.

- Ограниченная поддержка сообщества. Будучи относительно новым проектом с открытым исходным кодом, ZENML может иметь ограниченную поддержку сообщества по сравнению с более устоявшимися платформами MLOps, такими как Kubeflow.

- Ограниченное количество готовых компонентов: хотя ZENML предоставляет ряд готовых компонентов для общих задач машинного обучения, таких как

предварительная обработка данных и обучение моделей, выбор компонентов более ограничен по сравнению с некоторыми другими фреймворками MLOps.

- Зависимость от TensorFlow. Хотя интеграция ZENML с TensorFlow является преимуществом, она также может быть недостатком для пользователей, которые предпочитают использовать другие библиотеки или инструменты машинного обучения.

EXPLAINABLE AI

Explainable AI (XAI) или Объяснимый ИИ — это набор методов и практик, целью которых является сделать модели машинного обучения и их решения более прозрачными и понятными для людей.

Цель XAI — предоставить представление о том, как работает модель машинного обучения, как она принимает решения и какие факторы влияют на ее прогнозы. Это важно, поскольку многие современные модели машинного обучения сложны и трудны для интерпретации, а их выбор может существенно повлиять на отдельных людей и общество.

Методы XAI включают анализ важности функций, интерпретируемость локальной и глобальной модели, контрфактический анализ и визуализацию модели. Эти методы могут помочь выявить наиболее важные факторы, влияющие на прогнозы модели, дать объяснения конкретным прогнозам и выявить потенциальные отклонения или неточности в модели.

Объяснимый ИИ особенно важен в приложениях, где решения, принимаемые с помощью моделей машинного обучения, имеют серьезные последствия, например в здравоохранении, финансах и правосудии. Делая модели машинного обучения более прозрачными и понятными, XAI может помочь укрепить доверие к этим системам и гарантировать, что они принимают справедливые и этичные решения.

SHAP

SHAP (SHapley Additive exPlanations) — популярная библиотека с открытым исходным кодом для интерпретации и объяснения прогнозов моделей машинного обучения. SHAP основан на концепции ценностей Шепли, которая представляет собой метод теории кооперативных игр, используемый для определения вклада каждого игрока в кооперативную игру. В контексте машинного обучения SHAP вычисляет вклад каждой функции в конкретный прогноз, обеспечивая представление о том, как модель делает свои прогнозы.

Он предоставляет ряд инструментов для визуализации и интерпретации прогнозов модели, включая сводные графики, графики сил и графики зависимостей. Его можно использовать с широким спектром моделей машинного обучения, включая модели «черного ящика» и «белого ящика».

В целом, Python SHAP — это мощный инструмент для понимания того, как модели машинного обучения делают прогнозы, и который может быть полезен в ряде приложений, включая выбор функций, отладку модели и управление моделями.

Пример использования кода Python SHAP:

```
import shap
from sklearn.ensemble import
RandomForestClassifier
from sklearn.datasets import load_breast_cancer

# Load the Breast Cancer Wisconsin dataset
data = load_breast_cancer()
```

```
# Create a random forest classifier
clf = RandomForestClassifier(n_estimators=100,
random_state=0)

# Train the classifier on the breast cancer
dataset
clf.fit(data.data, data.target)

# Initialize the SHAP explainer
explainer = shap.Explainer(clf)

# Generate SHAP values for the first 5
instances in the dataset
shap_values = explainer(data.data[:5])

# Plot the SHAP values for the first instance
shap.plots.waterfall(shap_values[0])
```

В этом примере мы сначала загружаем набор данных рака молочной железы, штат Висконсин, и создаем случайный классификатор леса, используя класс **RandomForestClassifier** из scikit-learn. Затем мы обучаем классификатор на наборе данных.

Далее мы инициализируем эксплейнер SHAP, используя класс **Explainer** из библиотеки **shap**. Затем мы генерируем значения SHAP для первых 5 экземпляров набора данных, используя эксплейнер.

Наконец, мы отображаем значения SHAP для первого экземпляра, используя функцию **waterfall** из модуля **shap.plots**. В результате создается водопадный график, показывающий вклад каждой функции в прогноз модели для первого экземпляра.

Это всего лишь простой пример того, как SHAP можно использовать для интерпретации прогнозов модели

машинного обучения. На практике SHAP можно использовать с широким спектром моделей и наборов данных машинного обучения и может предоставить ценную информацию о том, как эти модели делают свои прогнозы.

За и против

За

- Предоставляет мощный инструмент для интерпретации и объяснения прогнозов моделей машинного обучения.
- Работает с широким спектром моделей машинного обучения, включая модели черного ящика.
- Может использоваться для решения различных задач, включая выбор функций, отладку модели и управление моделью.
- Предоставляет ряд визуализаций для изучения и интерпретации прогнозов модели.
- Основано на устоявшейся концепции теории кооперативных игр (вектор Шепли).
- Поддерживается активным сообществом и широко используется в промышленности и научных кругах.

Против

- Может требовать больших вычислительных ресурсов, особенно для больших наборов данных или сложных моделей.

- Может быть сложной для интерпретации и понимания, особенно для пользователей, которые не знакомы с основными концепциями и методами.
- Для эффективного использования требуются некоторые знания Python и концепций машинного обучения.
- Может быть чувствительна к выбору гиперпараметров и других настроек.
- Не всегда может давать четкие или окончательные объяснения предсказаний модели.

SHAP — мощный и широко используемый инструмент для интерпретации и объяснения моделей машинного обучения. Однако, как и любой инструмент, он имеет свои ограничения и требует определенного опыта для эффективного использования. Важно тщательно учитывать компромиссы и ограничения любого инструмента интерпретации модели при выборе подходящего для конкретного приложения.

LIME

Python LIME (Local Interpretable Model-Agnostic Explanations или локальные интерпретируемые модельно-независимые объяснения) — это библиотека с открытым исходным кодом для объяснения предсказаний моделей машинного обучения. Как и Python SHAP, LIME предоставляет возможность понять, как модель делает прогнозы, создавая объяснения для отдельных экземпляров. Однако в то время как SHAP предоставляет глобальные оценки важности функций, LIME генерирует локальные объяснения, специфичные для конкретного экземпляра.

Она работает путем обучения интерпретируемой модели (например, линейной модели или дерева решений) на выборке случаев, похожих на объясняемый пример. Интерпретируемая модель затем используется для создания объяснений предсказаний исходной модели. Этот процесс повторяется для каждого объясняемого экземпляра, в результате чего появляются локальные объяснения, специфичные для конкретного экземпляра.

LIME можно использовать с различными моделями машинного обучения и получать полезную информацию о том, как эти модели делают свои прогнозы. Это может быть особенно полезно при работе с моделями типа «черный ящик» или когда глобальных показателей важности признаков недостаточно для понимания отдельных прогнозов.

В целом, Python LIME — мощный инструмент для

интерпретации и объяснения моделей машинного обучения, особенно в тех случаях, когда SHAP и другие методы глобальной интерпретации могут оказаться недостаточными. Его можно использовать в различных приложениях, включая отладку моделей, управление моделями и выбор функций.

Пример использования кода Python LIME:

```
from lime import lime_text
from sklearn.pipeline import make_pipeline
from sklearn.ensemble import
RandomForestClassifier
from sklearn.feature_extraction.text import
TfidfVectorizer

# Define a dataset of text documents and
corresponding labels
docs = ['The quick brown fox', 'Jumped over the
lazy dog', 'The dog chased the cat', 'The cat
ran away']
labels = [1, 0, 1, 0]

# Define a random forest classifier and a TF-
IDF vectorizer
clf = RandomForestClassifier(n_estimators=100,
random_state=0)
vectorizer = TfidfVectorizer()

# Train the classifier on the text data
X_train = vectorizer.fit_transform(docs)
clf.fit(X_train, labels)

# Define a LIME explainer for text data
explainer =
lime_text.LimeTextExplainer(class_names=['negat
ive', 'positive'])

# Generate an explanation for the first
document
exp = explainer.explain_instance(docs[0],
clf.predict_proba, num_features=6)

# Print the explanation
print(exp.as_list())
```

В этом примере мы определяем набор данных текстовых документов и соответствующих меток. Затем мы определяем классификатор случайного леса и векторизатор TF-IDF и обучаем классификатор на текстовых данных.

Далее мы определяем блок объяснения LIME для текстовых данных, используя класс **LimeTextExplainer** из библиотеки **lime**. Затем мы генерируем объяснение для первого документа, используя блок объяснения.

Наконец, мы выводим объяснение, используя метод объяснений объектов **as_list**. При этом приводится список объектов и их веса, указывающий вклад каждого объекта в модели прогноза для первого документа.

Это всего лишь простой пример того, как LIME можно использовать для интерпретации прогнозов модели машинного обучения по текстовым данным. На практике LIME можно использовать с широким спектром моделей машинного обучения и типов данных и она может предоставить ценную информацию о том, как эти модели делают свои прогнозы.

За и против:

За

- Локальная интерпретируемость: LIME предоставляет объяснения для конкретных

случаев, позволяя понять, как модель делает прогнозы в каждом конкретном случае.

- Независимость от модели: LIME можно использовать с широким спектром моделей машинного обучения, включая модели типа «черный ящик», которые трудно интерпретировать с помощью других методов.

- Гибкость: LIME можно использовать с различными типами данных, включая текст, изображения и табличные данные.

- Интуитивно понятный результат: LIME генерирует объяснения, которые легко понять даже неспециалистам.

- Открытый исходный код: LIME — это библиотека с открытым исходным кодом, которая находится в свободном доступе и может настраиваться и расширяться по мере необходимости.

Против

- Ограничение локальными объяснениями: LIME предназначена для создания пояснений для конкретных случаев и может не подходить для понимания глобальных закономерностей или тенденций данных.

- Опирается на выборку: LIME генерирует объяснения, обучая интерпретируемую модель на выборке примеров, похожих на объясняемый пример. Это означает, что качество объяснений может зависеть от

качества и репрезентативности обучающих данных.

- Требует знаний в предметной области: для эффективного использования LIME важно хорошо понимать данные и объясняемую модель машинного обучения. Это может потребовать некоторых знаний в конкретной области.

- Интенсивные вычисления: создание объяснений LIME может потребовать больших вычислительных ресурсов, особенно для больших наборов данных или сложных моделей. Это может ограничить ее полезность в некоторых случаях использования.

- Результаты не всегда единообразны: поскольку объяснения LIME основаны на выборке случаев, они могут быть несогласованными в разных выборках или сериях. Это может затруднить сравнение и анализ различных объяснений.

INTERPRETML

InterpretML — это библиотека Python с открытым исходным кодом для интерпретации и объяснения моделей машинного обучения. Она предоставляет ряд инструментов и методов для понимания того, как модель делает свои прогнозы, включая глобальную значимость функций, локальные объяснения и контрфактические рассуждения. Библиотека не зависит от модели и может использоваться с широким спектром моделей машинного обучения, включая модели регрессии, классификации и временных рядов.

InterpretML предоставляет ряд методов интерпретируемости, в том числе:

- **Значимость функции**: предоставляет инструменты для понимания относительной важности различных функций в модели как на глобальном, так и на локальном уровне.
- **Локальные объяснения**: предоставляет инструменты для создания объяснений для конкретного экземпляра, которые могут помочь понять, почему был сделан тот или иной прогноз.
- **Противоречивые объяснения**: предоставляет инструменты для создания противоречащих фактам объяснений, которые показывают, как изменение значения признака повлияет на прогноз модели.
- **Графики частичной зависимости**: InterpretML

предоставляет инструменты для создания графиков частичной зависимости, которые показывают, как изменение значения признака влияет на прогноз модели, одновременно контролируя значения других признаков.

InterpretML можно использовать для решения различных задач, в том числе:

- **Отладка модели**: может помочь выявить и диагностировать проблемы с моделью, такие как смещение или переобучение.
- **Выбор модели**: может использоваться для сравнения и оценки различных моделей машинного обучения на основе их интерпретируемости и производительности.
- **Развертывание модели**: InterpretML может помочь объяснить и обосновать решения, принятые с помощью модели машинного обучения, заинтересованным сторонам и регулирующим органам.

Это мощный инструмент для понимания и интерпретации моделей машинного обучения, который можно использовать для повышения прозрачности, подотчетности и надежности моделей.

Пример использования кода InterpretML для создания глобальной важности функций и локальных объяснений для модели бинарной классификации:

```
# Import the necessary libraries
from interpret.glassbox import
ExplainableBoostingClassifier
from interpret import show
```

```
# Load the dataset
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer()
X, y = data.data, data.target

# Train an ExplainableBoostingClassifier model
ebm =
ExplainableBoostingClassifier(random_state=42)
ebm.fit(X, y)

# Generate global feature importances
global_explanation = ebm.explain_global()
show(global_explanation)

# Generate local explanations for a specific
instance
local_explanation = ebm.explain_local(X[:5])
show(local_explanation)
```

В этом примере мы сначала загружаем набор данных о раке молочной железы из scikit-learn и разделяем его на функции (X) и цели (y). Затем мы обучаем модель ExplainableBoostingClassifier на наборе данных и используем InterpretML для генерации глобальных значимостей функций и локальных объяснений.

Метод **explain_global()** генерирует глобальные значения функций для модели, что может помочь определить наиболее важные функции для прогнозирования. Метод **show()** используется для визуализации результатов.

Метод **explain_local()** генерирует локальные объяснения для конкретного экземпляра (в данном случае первых 5 экземпляров в наборе данных). Локальные объяснения помогают понять, почему тот или иной прогноз был сделан для конкретного экземпляра, и могут быть полезны для отладки и

уточнения модели.

В целом, этот пример демонстрирует, как InterpretML можно использовать для понимания и интерпретации моделей машинного обучения, а также для получения информации, которую можно использовать для повышения производительности и прозрачности модели.

За и против

За

- Независимость от модели: InterpretML можно использовать с широким спектром моделей машинного обучения, что делает его очень гибким и адаптируемым к различным сценариям использования.
- Интерпретируемость: InterpretML предоставляет ряд инструментов и методов для понимания того, как модель делает прогнозы, включая глобальную значимость признаков, локальные объяснения и контрфактические рассуждения.
- Комплексность: InterpretML предоставляет ряд методов интерпретации, включая значимость функций, локальные объяснения, контрфактические объяснения и графики частичной зависимости.
- Простота использования: InterpretML прост в использовании благодаря простому и интуитивно понятному API.
- Открытый исходный код: InterpretML — это библиотека с открытым исходным кодом, что означает, что ее можно использовать бесплатно, а сообщество может изменять и расширять ее.

Против

- Ограниченная масштабируемость:

InterpretML может быть дорогостоящим и медленным в вычислительном отношении при работе с большими наборами данных или сложными моделями.

- Ограниченная поддержка глубокого обучения: InterpretML в первую очередь разработан для интерпретируемых моделей машинного обучения и может не подходить для моделей глубокого обучения, которые по своей сути менее интерпретируемы.

- Ограниченная поддержка некоторых вариантов использования. Хотя InterpretML предоставляет широкий спектр методов интерпретации, в некоторых случаях могут потребоваться более специализированные методы.

ОБРАБОТКА ТЕКСТА

Обработка текста — это анализ и манипулирование текстовыми данными для извлечения полезной информации или идей. Она включает в себя различные методы и инструменты, включая обработку естественного языка (NLP), машинное обучение и статистический анализ.

Обработку текста можно использовать для различных задач, включая классификацию текста, анализ настроений, распознавание объектов, тематическое моделирование и поиск информации. Она используется во многих отраслях, включая здравоохранение, финансы и электронную коммерцию, для анализа больших объемов текстовых данных и получения информации о поведении клиентов, тенденциях рынка и других жизненно важных факторах. Обычно обработка текста подразделяется на несколько этапов, включая парсинг и предварительную обработку данных, извлечение признаков, обучение и проверку модели, а также развертывание модели. Методы NLP, такие как токенизация, маркировка частей речи и распознавание именованных объектов, часто используются для предварительной обработки данных и извлечения функций.

Алгоритмы машинного обучения, такие как деревья решений, машины опорных векторов и нейронные сети, часто используются для построения моделей, которые могут классифицировать, кластеризовать или анализировать текстовые данные. Методы статистического анализа, такие как регрессия и

кластеризация, также могут использоваться для получения более глубокого понимания данных.

Обработка текста — это быстро развивающаяся область, в которой постоянно разрабатываются новые инструменты и методы. Это важная область исследований и разработок, поскольку объем генерируемых текстовых данных растет в геометрической прогрессии.

SPACY

Spacy — это библиотека с открытым исходным кодом для расширенной обработки естественного языка (NLP) в Python. Она предоставляет широкий спектр возможностей NLP, включая токенизацию, тегирование частей речи, распознавание именованных объектов, анализ зависимостей и многое другое. Spacy спроектирован так, чтобы быть быстрым, эффективным и удобным для пользователя, что делает его популярным выбором для разработки приложений NLP.

Она также включает предварительно обученные модели для нескольких языков, что позволяет быстро приступить к выполнению задач NLP на разных языках. Кроме того, Spacy позволяет пользователям обучать свои модели на пользовательских наборах данных, что позволяет им создавать решения NLP, адаптированные к их конкретным потребностям.

В целом, Spacy — мощный инструмент для решения задач NLP на Python, предлагающий ряд функций и предварительно обученных моделей для оптимизации разработки NLP.

Пример использования Spacy для извлечения именованных объектов:

```
import spacy

# Load a pre-trained model
nlp = spacy.load('en_core_web_sm')

# Text to process
text = "Apple is looking at buying U.K. startup
```

```
for $1 billion"

# Process the text with the loaded model
doc = nlp(text)

# Print each token with its part-of-speech
(POS) tag and named entity recognition (NER)
label
for token in doc:
    print(token.text, token.pos_,
token.ent_type_)
```

Этот код использует библиотеку SpaCy для загрузки предварительно обученной модели английского языка (**en_core_web_sm**) и обработки текстовой строки. Затем он перебирает каждый токен в обработанном документе и выводит его текст, тег части речи (POS) и метку распознавания именованного объекта (NER). Вывод может выглядеть следующим образом:

```
Apple PROPN ORG
is AUX
looking VERB
at ADP
buying VERB
U.K. GPE
startup NOUN
for ADP
$ NUM
1 NUM
billion NUM
```

За и против

За

- Высокая степень оптимизация по скорости и использованию памяти, что делает ее эффективной даже при работе с большими

наборами данных.

- Обеспечивает высочайшую производительность при решении различных задач NLP, включая распознавание именованных объектов, маркировку частей речи, анализ зависимостей и многое другое.

- Предлагает простую интеграцию с другими библиотеками и платформами Python, такими как scikit-learn, PyTorch и TensorFlow.

- Предоставляет удобный и согласованный API для выполнения задач NLP.

- Включает предварительно обученные модели для нескольких языков, что упрощает начало работы с NLP для языков, отличных от английского.

- Имеет активное сообщество разработчиков и хорошую документацию.

Против

- Имеет более крутую кривую обучения по сравнению с некоторыми другими библиотеками NLP.

- Могут не так эффективно выполнять некоторые конкретные задачи NLP по сравнению с другими библиотеками.

- Хотя основная библиотека имеет открытый исходный код, некоторые предварительно обученные модели доступны только на коммерческой основе.

NLTK

NLTK означает Natural Language Toolkit или «Инструментарий естественного языка». Это популярная библиотека с открытым исходным кодом для задач обработки естественного языка (NLP) на Python. Он предоставляет широкий спектр функций для обработки человеческого языка, таких как токенизация, стемминг, лемматизация, POS-теги и многое другое. Она также включает в себя ряд готовых корпусов и ресурсов для обучения моделей машинного обучения задачам NLP. NLTK широко используется для различных способов применения, таких как классификация текста, анализ настроений, машинный перевод и извлечение информации.

Простой пример использования кода Python NLTK для токенизации:

```
import nltk
from nltk.tokenize import word_tokenize

# sample text
text = "This is an example sentence for
tokenization."

# tokenize the text
tokens = word_tokenize(text)

# print the tokens
print(tokens)
```

Результат:

```
['This', 'is', 'an', 'example', 'sentence',
'for', 'tokenization', '.']
```

За и против

За

- Предоставляет широкий спектр инструментов и модулей обработки естественного языка, включая токенизацию, стемминг, тегирование, синтаксический анализ и классификацию.
- Имеет большое сообщество пользователей и разработчиков, что позволяет легко найти помощь и ресурсы в Интернете.
- Предлагает поддержку нескольких языков.
- Поставляется с различными наборами данных и корпусами для обучения и тестирования моделей.
- Обеспечивает удобный интерфейс для начинающих.
- Может быть интегрирована с другими библиотеками Python, такими как NumPy и Pandas.

Против

- Может работать медленнее, чем другие библиотеки обработки естественного языка, поскольку использует структуры данных Python.
- Документация может оказаться слишком сложной для новичков, и в ней может быть

сложно ориентироваться.

- Некоторые алгоритмы и модели могут быть не такими продвинутыми и точными, как в других библиотеках.

- NLTK может не подойти для крупномасштабных задач обработки естественного языка из-за ограничений памяти.

- Код может быть более многословным и трудным для чтения по сравнению с другими библиотеками обработки естественного языка.

TEXTBLOB

Python TextBlob — это популярная библиотека Python с открытым исходным кодом, используемая для обработки текстовых данных. Она предоставляет простой API для задач обработки естественного языка, таких как анализ настроений, маркировка частей речи, извлечение именной фразы и многое другое. Она построена на основе библиотеки Natural Language Toolkit (NLTK) и предоставляет простой в использовании интерфейс для обработки текста.

Пример использования кода Python TextBlob:

```
from textblob import TextBlob

# Creating a TextBlob object
text = "I am really enjoying this course on
natural language processing."
blob = TextBlob(text)

# Sentiment Analysis
sentiment_polarity = blob.sentiment.polarity
sentiment_subjectivity =
blob.sentiment.subjectivity
print("Sentiment Polarity:",
sentiment_polarity)
print("Sentiment Subjectivity:",
sentiment_subjectivity)

# Parts of Speech Tagging
pos_tags = blob.tags
print("Parts of Speech Tags:", pos_tags)

# Named Entity Recognition
ner_tags = blob.noun_phrases
print("Named Entity Recognition:", ner_tags)

# Text Translation
```

```
translation = blob.translate(to='fr')
print("Translation to French:", translation)
```

Этот код выполняет анализ настроений, маркировку частей речи, распознавание именованных объектов и перевод текста с помощью TextBlob. Результат будет зависеть от используемого входного текста.

За и против

За

- TextBlob прост в использовании и имеет простой синтаксис, что делает его доступным для новичков в обработке естественного языка.
- Она имеет встроенные возможности анализа настроений, которые полезны для таких задач, как мониторинг социальных сетей и сбор мнений.
- TextBlob также включает в себя другие задачи обработки естественного языка, такие как извлечение именной фразы, разметка частей речи и классификация.
- Библиотека построена на основе библиотеки NLTK, поэтому имеет доступ к широкому спектру инструментов и ресурсов, доступных в NLTK.

Против

- TextBlob не такая мощная и настраиваемая, как другие библиотеки обработки естественного языка, такие как SPACY.

- Библиотека может быть не такой эффективной и масштабируемой, как другие варианты, особенно для больших наборов данных.
- Встроенный анализ настроений TextBlob не всегда может быть точным, особенно для более сложного и тонкого текста.

В целом TextBlob — полезный инструмент для начинающих, а также для простых задач обработки естественного языка, но он может быть не лучшим выбором для более сложных или крупномасштабных проектов.

CORENLP

Python CoreNLP — это оболочка Python для Stanford CoreNLP, набора инструментов обработки естественного языка на основе Java, разработанного Стэнфордским университетом. Она предоставляет набор инструментов для различных задач обработки естественного языка, таких как разметка частей речи, распознавание именованных объектов, анализ зависимостей, анализ настроений и многое другое. Его можно использовать для анализа и извлечения информации из текстовых данных в различных форматах, таких как обычный текст, HTML и XML.

Пример кода, который использует Python CoreNLP для анализа предложения и извлечения именованных сущностей:

```
from stanfordcorenlp import StanfordCoreNLP

nlp = StanfordCoreNLP(r'/path/to/corenlp',
memory='8g'
)

sentence = "John works at Google in
California."
output       =       nlp.annotate(sentence,
properties={
                        'annotators': 'ner',
                        'outputFormat': 'json',
                        'timeout': 1000,
                    })
for entity in
output['sentences'][0]['entitymentions']:
    print(entity['text'], entity['ner'])
```

Результат:

```
John PERSON
```

```
Google ORGANIZATION
California STATE_OR_PROVINCE
```

В этом примере мы сначала импортируем класс **StanfordCoreNLP** из пакета **stanfordcorenlp**. Затем мы создаем объект **StanfordCoreNLP** и указываем путь к установке CoreNLP и объем памяти, который будет использоваться.

Затем мы определяем предложение и используем метод **annotate()** объекта **StanfordCoreNLP** для анализа предложения и извлечения именованных сущностей. Мы указываем аннотатор 'ner' для распознавания именованного объекта и устанавливаем выходной формат 'json'. Мы также установили таймаут 1000 миллисекунд.

Наконец, мы просматриваем именованные объекты на выходе и печатаем их текст и тег NER

За и против

За

- CoreNLP предоставляет широкий спектр задач NLP, таких как разметка частей речи, распознавание именованных объектов, анализ настроений и анализ зависимостей.
- Она написана на Java и может быть легко интегрирована с Python и другими языками программирования.
- Она может обрабатывать большие наборы текстовых данных и предоставлять точные и надежные результаты.

- Она также поддерживает различные языки, наряду с английским, такие как китайский, испанский, французский, немецкий и арабский.

Против

- Процесс установки и настройки CoreNLP может быть сложным и трудоемким.
- CoreNLP требует значительного объема памяти и вычислительных ресурсов для выполнения задач с большими наборами данных, что может оказаться невозможным на машинах начального уровня.
- Результаты CoreNLP не всегда могут быть идеальными и для улучшения результатов может потребоваться некоторое ручное вмешательство.
- Она может не подходить для приложений реального времени или онлайн-приложений из-за высоких вычислительных требований.

GENSIM

Gensim — это библиотека с открытым исходным кодом для неконтролируемого тематического моделирования и обработки естественного языка. Она предоставляет набор алгоритмов и моделей для таких задач, как анализ сходства документов, кластеризация документов и тематическое моделирование. Библиотека спроектирована так, чтобы быть масштабируемой и эффективной, с поддержкой потоковой передачи данных и распределенных вычислений.

Gensim построена на основе NumPy, SciPy и других библиотек научных вычислений и обладает простым и интуитивно понятным интерфейсом для задач анализа текста. Она поддерживает различные форматы файлов входных данных, включая обычный текст, HTML и XML, а также обеспечивает встроенную поддержку общих этапов предварительной обработки текста, таких как токенизация, стемминг и удаление стоп-слов.

В целом, Gensim — это мощный инструмент для изучения и анализа больших коллекций текстовых данных, который можно использовать для широкого спектра приложений, включая поиск информации, системы рекомендаций и анализ контента.

Пример кода, который использует Gensim для создания простой тематической модели из образца набора текстовых данных:

```
import gensim
```

```
from gensim import corpora
from pprint import pprint

# Define the dataset
data = [
    "I like to eat broccoli and bananas.",
    "I ate a banana and spinach smoothie for
breakfast.",
    "Chinchillas and kittens are cute.",
    "My sister adopted a kitten yesterday.",
    "Look at this cute hamster munching on a
piece of broccoli."
]

# Tokenize the dataset
tokenized_data =
[gensim.utils.simple_preprocess(text) for text
in data]

# Create a dictionary from the tokenized data
dictionary = corpora.Dictionary(tokenized_data)

# Create a corpus from the dictionary and
tokenized data
corpus = [dictionary.doc2bow(text) for text in
tokenized_data]

# Train the LDA model
lda_model = gensim.models.ldamodel.LdaModel(
    corpus=corpus,
    id2word=dictionary,
    num_topics=2,
    random_state=100,
    update_every=1,
    chunksize=10,
    passes=10,
    alpha='auto',
    per_word_topics=True
)

# Print the topics
pprint(lda_model.print_topics())
```

Результат:

```
[(0,
  '0.082*"and" + 0.082*"broccoli" + 0.082*"eat"
+ 0.082*"to" + 0.082*"bananas" + 0.060*"i" +
0.057*"a" + 0.035*"for" + 0.035*"breakfast" +
0.035*"smoothie"'),
 (1,
  '0.077*"kitten" + 0.056*"and" + 0.056*"are" +
0.056*"chinchillas" + 0.056*"cute" + 0.056*"my"
+ 0.056*"sister" + 0.056*"adopted" +
0.056*"yesterday" + 0.056*"look"')]
```

В этом примере мы используем Gensim для токенизации образца набора данных, создания словаря и корпуса и обучения тематической модели LDA с двумя темами. В результате показаны самые популярные слова, связанные с каждой темой.

За и против

За

- Простота использования API для создания и обучения тематических моделей
- Поддерживает несколько алгоритмов тематического моделирования, таких как скрытое распределение Дирихле (LDA) и скрытый семантический анализ (LSA)
- Может эффективно обрабатывать большие наборы данных
- Предоставляет инструменты для предварительной обработки текста, такие как токенизация и удаление стоп-слов

- Может генерировать встраивания слов с использованием популярных алгоритмов, таких как Word2Vec и FastText

Против

- Ограниченная поддержка методов глубокого обучения по сравнению с другими библиотеками, такими как TensorFlow или PyTorch
- Для эффективного использования могут потребоваться некоторые знания в области статистического анализа и концепций машинного обучения
- Некоторые функции могут работать медленнее, чем в других библиотеках, поскольку они ориентированы на эффективность использования памяти и масштабируемость

REGEX

Библиотека Python Regex (Regular Expression или регулярные выражения) — мощный инструмент, используемый для сопоставления шаблонов и обработки текста. Он предоставляет набор функций и метасимволов, которые позволяют нам искать строки и манипулировать ими, используя сложные шаблоны. Регулярное выражение представляет собой последовательность символов, определяющую шаблон поиска. Модуль re, встроенный в Python, обеспечивает поддержку регулярных выражений Python. Это широко используемая библиотека для выполнения различных задач по манипулированию текстом, таких как сопоставление строк, поиск, анализ и замена.

Пример использования библиотеки регулярных выражений **re** в Python для извлечения информации из сложной строки:

```
import re

# Example string to search through
text = "My phone number is (123) 456-7890 and
my email is example@example.com."

# Define regex patterns to search for
phone_pattern = re.compile(r'\(\d{3}\)\s\d{3}-
\d{4}')  # Matches phone numbers in (123) 456-
7890 format
email_pattern = re.compile(r'\b[\w.-
]+?@\w+?\.\w+?\b')  # Matches email addresses

# Search for matches in the text
phone_match = phone_pattern.search(text)
```

```
email_match = email_pattern.search(text)

# Print out the results
if phone_match:
    print("Phone number found:",
phone_match.group())

else:
    print("Phone number not found.")

if email_match:
    print("Email found:", email_match.group())
else:
    print("Email not found.")
```

Результат:

```
Phone number found: (123) 456-7890
Email found: example@example.com
```

В этом примере мы используем регулярные выражения для определения шаблонов поиска номера телефона и адреса электронной почты в сложной строке. Шаблоны компилируются с помощью функции **re.compile()**, а затем ищутся с помощью функции **search()**. Функция **group()** используется для получения фактического совпавшего текста.

За и против

За

- Мощные возможности: регулярные выражения – это мощный способ поиска текста и управления им.
- Эффективность: библиотека Python Regex

оптимизирована по производительности и может быстро обрабатывать большие объемы текста.

- Универсальность: регулярные выражения можно использовать для широкого круга задач; от простого сопоставления строк до сложного анализа и манипулирования текстом.

- Гибкость: библиотека Python Regex обеспечивает широкие возможности настройки, позволяя создавать сложные шаблоны и сопоставлять определенные шаблоны в тексте.

Против

- Крутая кривая обучения: регулярные выражения могут быть трудными для изучения, особенно для новичков в программировании.

- Легко использовать неправильно: из-за своей сложности регулярные выражения могут быть подвержены ошибкам и их сложно отлаживать.

- Ограниченная функциональность: хотя библиотека Python Regex является мощной, она имеет некоторые ограничения и может подходить не для всех задач обработки текста.

- Менее читабельные: регулярные выражения могут быть менее читабельными, чем другие формы кода обработки текста, что затрудняет поддержку и обновление кода.

ОБРАБОТКА ИЗОБРАЖЕНИЙ

Обработка изображений анализирует и манипулирует цифровыми изображениями для извлечения полезной информации или улучшения их качества. Она включает в себя различные методы и инструменты, включая компьютерное зрение, машинное обучение и обработку сигналов.

Обработка изображений может использоваться для решения различных задач, включая обнаружение и распознавание объектов, сегментацию изображений, улучшение изображений и распознавание образов. Она используется во многих отраслях, включая здравоохранение, производство и развлечения, для анализа цифровых изображений и манипулирования ими, а также для получения информации об основных данных.

Обработка изображений обычно включает в себя несколько этапов, включая получение изображений, предварительную обработку, извлечение признаков, обучение и проверку модели, а также развертывание модели. Методы компьютерного зрения, такие как обнаружение границ, распознавание объектов и сегментация изображений, часто используются для предварительной обработки данных и извлечения функций.

Алгоритмы машинного обучения, такие как сверточные нейронные сети (CNN), часто используются для создания моделей, которые могут классифицировать, обнаруживать или анализировать цифровые изображения. Кроме того, для повышения

качества цифровых изображений можно использовать методы обработки сигналов, такие как фильтрация и анализ Фурье.

Обработка изображений — быстро развивающаяся область, в которой постоянно разрабатываются новые инструменты и методы. Это важная область исследований и разработок, поскольку использование цифровых изображений продолжает расти во многих областях.

OPENCV

OpenCV (Open-Source Computer Vision Library или библиотека компьютерного зрения с открытым исходным кодом) — это библиотека функций программирования, в основном предназначенная для компьютерного зрения в реальном времени. Она предоставляет множество полезных и мощных алгоритмов и методов для приложений компьютерного зрения и машинного обучения, включая обработку изображений и видео, обнаружение и распознавание объектов, калибровку камеры и многое другое.

OpenCV написана на C++ и предоставляет привязки для Python, что упрощает его использование в приложениях Python. Она также включает графический интерфейс пользователя (GUI) для обработки изображений и видео, что упрощает визуализацию данных и взаимодействие с ними.

Некоторые из ключевых особенностей OpenCV включают в себя:

- **Обработка изображений и видео**: предоставляет множество функций для базовой и расширенной обработки изображений и видео, включая фильтрацию, обнаружение функций, сегментацию изображений и многое другое.
- **Обнаружение и распознавание объектов**: OpenCV предоставляет несколько методов обнаружения и распознавания объектов,

включая каскады Хаара, HOG (гистограммы ориентированных градиентов) и подходы, основанные на глубоком обучении.

- **Калибровка камеры**: включает функции для калибровки камер, включая оценку внутренних и внешних параметров, коррекцию искажений и многое другое.

- **Машинное обучение**: предоставляет несколько алгоритмов машинного обучения для классификации, регрессии, кластеризации и многого другого.

В целом, OpenCV — это мощный инструментарий компьютерного зрения и машинного обучения, который широко используется как в академических, так и в промышленных условиях.

Пример кода, который использует OpenCV для захвата видео с веб-камеры и отображения его на экране:

```
import cv2

# Create a VideoCapture object
cap = cv2.VideoCapture(0)

while True:
    # Read a frame from the camera
    ret, frame = cap.read()

    # Display the frame
    cv2.imshow('frame', frame)

    # Exit if the 'q' key is pressed
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# Release the VideoCapture object and close the
window
```

```
cap.release()
cv2.destroyAllWindows()
```

В этом коде мы сначала импортируем модуль **cv2**, который предоставляет функции и классы, необходимые для работы с OpenCV. Затем мы создаем объект **VideoCapture** для захвата видео с веб-камеры по умолчанию (индекс устройства 0).

Внутри цикла **while** мы используем метод **cap.read()** для чтения кадра с камеры. Переменная **ret** указывает, была ли операция чтения успешной, а переменная **frame** содержит данные изображения для текущего кадра.

Затем мы используем функцию **cv2.imshow()** для отображения кадра на экране. Первый аргумент этой функции — это имя окна (которое может быть любым), а второй аргумент — данные изображения.

Наконец, мы используем функцию **cv2.waitKey()** для ожидания нажатия клавиши. Если нажата клавиша 'q', мы выходим из цикла, освобождаем объект **VideoCapture** и закрываем окно.

За и против

За

- OpenCV — это библиотека с открытым исходным кодом, что означает, что ее можно бесплатно использовать и модифицировать.
- Имеет большое сообщество разработчиков, что гарантирует постоянное улучшение библиотеки и добавление новых функций.

- OpenCV поддерживает несколько языков программирования, включая Python, C++ и Java.
- Она имеет широкий спектр функций обработки изображений и видео, что делает ее универсальным инструментом для различных приложений.
- Она поддерживает несколько платформ, включая Windows, Linux и MacOS.

Против

- OpenCV может потребовать из-за большого количества функций и сложных API сложного обучения для новичков.
- Для эффективного использования требуются некоторые знания компьютерного зрения и методов обработки изображений.
- Производительность OpenCV на некоторых устройствах может быть низкой, особенно при использовании сложных алгоритмов.
- Возможно, это не лучший выбор для приложений, требующих обработки больших объемов данных в реальном времени из-за высоких требований к вычислительным ресурсам.

SCIKIT-IMAGE

Python scikit-image — это библиотека обработки изображений с открытым исходным кодом, которая предоставляет алгоритмы для обработки изображений и решения задач компьютерного зрения, таких как фильтрация, сегментация, обнаружение объектов и многое другое. Она построена на основе научной экосистемы Python, включая NumPy, SciPy и matplotlib.

Она проста в использовании и обладает простым и интуитивно понятным интерфейсом для быстрого выполнения задач по обработке изображений. Она поддерживает различные форматы изображений и совместима с Python 3.x.

Некоторые из ключевых особенностей scikit-image включают в себя:

- **Коллекцию алгоритмов обработки изображений и компьютерного зрения**
- **Поддержку различных форматов изображений, включая JPEG, PNG, BMP, TIFF и других**
- **Простой и интуитивно понятный API для легкой интеграции с другими библиотеками Python**
- **Совместимость с NumPy и SciPy для задач научных вычислений**

- **Полную документацию и примеры**

В целом, scikit-image — это мощный инструмент для обработки изображений и задач компьютерного зрения на Python.

Пример использования кода Python scikit-image для обработки изображений:

```
from skimage import io, filters

# Load image
image = io.imread('example.jpg', as_gray=True)

# Apply Gaussian blur
image_blur = filters.gaussian(image, sigma=1)

# Apply Sobel filter
sobel = filters.sobel(image)

# Display the images
io.imshow_collection([image, image_blur, sobel])
io.show()
```

В этом примере мы загружаем изображение с помощью функции **io.imread** и применяем к изображению размытие по Гауссу с помощью функции **filters.gaussian** со значением **sigma**, равным 1. Затем мы применяем фильтр Собеля к изображению с помощью функции **filters.sobel**.

Наконец, мы используем функцию **io.imshow_collection** для отображения исходного изображения, размытого изображения и изображения, отфильтрованного Собелем.

Обратите внимание, что **as_gray=True** используется для преобразования изображения в оттенки серого. Также функция **io.show()** используется для отображения изображений.

За и против

За

- scikit-image — мощная библиотека обработки изображений, предоставляющая широкий набор функций для управления и анализа изображений.
- Она построена на основе популярных научных библиотек Python NumPy и SciPy, что упрощает интеграцию с другими инструментами научных вычислений.
- scikit-image имеет обширную документацию и активное сообщество, а это означает, что найти помощь и поддержку относительно легко.
- Библиотека имеет открытый исходный код и доступна бесплатно по разрешительной лицензии, что делает ее доступной для всех.

Против

- Некоторые из более продвинутых функций scikit-image могут быть сложными в использовании и требуют глубокого понимания концепций обработки изображений.
- Библиотека может быть не такой производительной, как другие библиотеки обработки изображений, такие как OpenCV, для определенных задач.
- Некоторые пользователи сообщали о проблемах с установкой и совместимостью с некоторыми версиями Python и другими условиями.
- scikit-image не поддерживает обработку 3D-изображений «из коробки», что может быть ограничением для некоторых применений.

PILLOW

Pillow — популярная библиотека Python, используемая для задач обработки изображений. Это ответвление библиотеки изображений Python (PIL) и поддерживает многие ее функции, а также включает дополнительные функции и исправления ошибок. Pillow предоставляет полный набор функций для открытия, управления и сохранения файлов изображений в самых разных форматах, включая BMP, PNG, JPEG, TIFF и GIF.

Некоторые из ключевых функций Pillow включают поддержку различных форматов изображений, функции улучшения изображений и манипулирования ими, функции рисования и рендеринга текста, а также поддержку основных операций фильтрации и преобразования изображений. Она также включает в себя различные алгоритмы обработки изображений, такие как обнаружение краев, обнаружение контуров и сегментация изображения.

Pillow широко используется в различных приложениях для обработки изображений, включая компьютерное зрение, машинное обучение и веб-разработку. Эта библиотека известна своей простотой использования, а также гибкостью и масштабируемостью. Кроме того, Pillow — это программное обеспечение с открытым исходным кодом, что означает, что оно свободно доступно для использования и модификации кем угодно.

В целом Pillow — это мощная и универсальная

библиотека для работы с данными изображений в Python, а также важный инструмент для всех, кто работает с изображениями в своих проектах Python.

Пример использования кода Pillow с использованием фильтров:

```
from PIL import Image, ImageFilter

# Open the image
img = Image.open('image.jpg')

# Apply a Gaussian blur filter
blurred_img =
img.filter(ImageFilter.GaussianBlur(radius=10))

# Apply a sharpen filter
sharpened_img = img.filter(ImageFilter.SHARPEN)

# Display the original image and the filtered
images
img.show()
blurred_img.show()
sharpened_img.show()
```

В этом примере мы открыли изображение и применили к нему два разных фильтра с помощью Pillow. Сначала мы применили фильтр размытия по Гауссу радиусом 10 пикселей, который создает эффект размытия изображения. Затем мы применили к исходному изображению фильтр повышения резкости, который усиливает края и детали изображения. Наконец, мы отобразили все три изображения (исходное, размытое и резкое) с помощью метода **show()**.

За и против

За

- Pillow – это хорошо документированная и простая в использовании библиотека для обработки изображений в Python.
- Она поддерживает широкий спектр форматов изображений и позволяет выполнять ряд задач по манипулированию изображениями, включая обрезку, изменение размера и фильтрацию.
- Pillow пользуется мощной поддержкой сообщества и активно поддерживается частыми обновлениями и исправлениями ошибок.
- Pillow совместим как с Python 2, так и с Python 3, что делает ее универсальным выбором для обработки изображений в Python.

Против

- Хотя Pillow — мощная библиотека, она может не подойти для очень сложных задач обработки изображений, требующих более специализированных инструментов или алгоритмов.
- Pillow может работать относительно медленно при обработке больших или сложных изображений, особенно по сравнению с более оптимизированными библиотеками,

написанными на языках более низкого уровня, таких как C или C++.

- Pillow может осуществлять ограниченную поддержку некоторых менее распространенных форматов изображений, что может стать проблемой в некоторых случаях использования.

MAHOTAS

Python Mahotas — это библиотека обработки изображений, предоставляющая набор алгоритмов для обработки изображений и задач компьютерного зрения. Она построена на основе numpy и scipy и предоставляет функции для выполнения таких операций, как фильтрация, сегментация, извлечение признаков, морфология и другие задачи обработки изображений.

Эта библиотека предназначена для работы с массивами numpy, что упрощает интеграцию с другими библиотеками обработки изображений, такими как OpenCV и scikit-image. Она обеспечивает быструю и эффективную реализацию многих распространенных алгоритмов обработки изображений и поддерживает многомерные массивы, что делает ее пригодной для работы с объемными данными.

Некоторые из особенностей Mahotas включают в себя:

- **Фильтрация и сегментация изображений**
- **Извлечение признаков и распознавание объектов**
- **Морфологические операции, такие как эрозия и расширение**
- **Пороговое значение и обнаружение границ**
- **Сегментация границы перехода**

- **Свойства и маркировка области**

Пример использования кода:

```
import mahotas as mh
import numpy as np
from skimage import data

# Load example image
image = data.coins()

# Convert image to grayscale
image = mh.colors.rgb2gray(image)

# Apply thresholding
thresh = mh.thresholding.otsu(image)

# Label regions
labeled, nr_objects = mh.label(image > thresh)

# Calculate region properties
regions = mh.regionprops(labeled,
intensity_image=image)

# Display results
print("Number of objects:", nr_objects)
for region in regions:
    print("Object:", region.label)
    print("Area:", region.area)
    print("Perimeter:", region.perimeter)
    print("Eccentricity:", region.eccentricity)
    print("Intensity mean:",
region.mean_intensity)
    print("")
```

Этот код загружает пример изображения, преобразует его в оттенки серого, применяет метод определения порога Оцу для разделения пикселей переднего плана и фона, помечает связанные компоненты в результирующем двоичном изображении и вычисляет

различные свойства области для каждого объекта. В выводе отображается количество найденных объектов и их свойства.

За и против

За

- Mahotas предоставляет ряд мощных функций обработки изображений и извлечения признаков, что делает его пригодным для различных задач компьютерного зрения.
- Библиотека хорошо документирована и содержит ряд примеров для начала работы.
- Mahotas предназначена для эффективной работы с большими наборами данных изображений, что позволяет пользователям быстро обрабатывать и анализировать большие объемы данных изображений.
- Mahotas проста в установке и использовании благодаря простому и понятному API.

Против

- Mahotas не предоставляет столько функций или расширенных возможностей, как некоторые из более известных библиотек компьютерного зрения, таких как OpenCV или scikit-image.
- Некоторые функции, предоставляемые Mahotas, могут работать медленно и выполнять определенные задачи не так хорошо, как другие библиотеки.

- Хотя Mahotas имеет относительно активное сообщество пользователей, оно может не так широко использоваться и поддерживаться, как другие библиотеки обработки изображений.

SIMPLEITK

SimpleITK — это высокоуровневый интерфейс к набору инструментов Insight Segmentation and Registration Toolkit (ITK). Это библиотека Python, используемая для обработки изображений, анализа и задач компьютерного зрения. SimpleITK позволяет легко манипулировать изображениями, например фильтровать, сегментировать, регистрировать и извлекать признаки.

Некоторые распространенные задачи, которые можно выполнить с помощью SimpleITK, включают выравнивание изображений, регистрацию нескольких изображений, сегментацию интересующих областей и анализ особенностей изображения. Библиотека также предоставляет доступ ко многим алгоритмам и методам анализа изображений, таким как обнаружение краев, обнаружение объектов и классификация.

SimpleITK — популярная библиотека для обработки и анализа медицинских изображений, поскольку она предоставляет инструменты для анализа медицинских изображений, таких как КТ, МРТ и ультразвуковые изображения. Она широко используется в сфере здравоохранения и в научных исследованиях.

В целом, SimpleITK предоставляет удобный интерфейс для комплекта инструментов ITK, упрощая пользователям выполнение сложных задач по обработке и анализу изображений. Он также имеет широкий спектр применений в различных областях, включая медицинскую визуализацию, компьютерное

зрение и машинное обучение.

Пример использования кода Python SimpleITK:

```
import SimpleITK as sitk

# Read an image
image = sitk.ReadImage("image.nii")

# Get the image size
size = image.GetSize()

# Get the image origin
origin = image.GetOrigin()

# Get the image spacing
spacing = image.GetSpacing()

# Get the image direction
direction = image.GetDirection()

# Print the image information
print("Size:", size)
print("Origin:", origin)
print("Spacing:", spacing)
print("Direction:", direction)

# Display the image
sitk.Show(image)
```

Этот код считывает изображение в формате NIfTI с помощью SimpleITK, получает размер изображения, начало координат, расстояние и направление, а затем отображает изображение с помощью функции **sitk.Show()**.

За и против

За

- SimpleITK — мощная библиотека для обработки и анализа изображений с широким набором функций для 2D, 3D и многомерных изображений.

- Она предоставляет простой и интуитивно понятный API для выполнения различных задач, таких как чтение и запись файлов изображений, применение фильтров изображений и сегментирование изображений.

- SimpleITK построена на основе ITK (Insight Segmentation and Registration Toolkit), который представляет собой хорошо зарекомендовавшую себя и широко используемую в исследовательском сообществе библиотеку анализа изображений.

- SimpleITK можно использовать с различными языками программирования, включая Python, C++, Java и Tcl.

Против

- SimpleITK имеет более сложную кривую обучения по сравнению с некоторыми другими библиотеками обработки изображений Python из-за более сложного API и того факта, что он построен на основе ITK.

- SimpleITK может не подходить для всех типов задач анализа изображений, поскольку она в

первую очередь предназначена для анализа медицинских изображений.

- Некоторые из более продвинутых функций SimpleITK, такие как регистрация и сегментация, требуют хорошего понимания основных концепций и алгоритмов.

- SimpleITK может работать медленнее по сравнению с некоторыми другими библиотеками обработки изображений Python из-за более сложных алгоритмов и структур данных.

ВЕБ-ФРЕЙМВОРК

Веб-фреймворк — это программная платформа, предназначенная для упрощения разработки веб-приложений путем предоставления набора повторно используемых компонентов и инструментов для создания веб-проектов и управления ими. Она обеспечивает стандартизированный способ создания и развертывания веб-приложений, предоставляя структуру, библиотеки и предварительно написанный код для выполнения повседневных задач, таких как обработка запросов, маршрутизация, обработка форм, проверка данных и доступ к базе данных.

Веб-фреймворк обычно включает в себя инструменты и библиотеки программирования, такие как шаблоны, промежуточное программное обеспечение и механизмы маршрутизации, которые позволяют разработчикам писать чистый, удобный в обслуживании и масштабируемый код для веб-проектов. Кроме того, он абстрагирует большую часть низкоуровневых деталей веб-разработки, позволяя разработчикам сосредоточиться на высокоуровневой функциональности своих приложений.

Существует множество веб-фреймворков, доступных на различных языках программирования, включая Python (Django, Flask), Ruby on Rails, PHP (Laravel, Symfony) и JavaScript (React, Angular, Vue.js). Эти платформы различаются по функциям, производительности, простоте использования и поддержке сообщества.

Веб-фреймворки стали необходимы для веб-разработки, поскольку они предоставляют стандартизированный способ создания и поддержки веб-приложений, упрощая разработчикам создание сложных веб-проектов за меньшее время и с меньшим количеством ошибок.

FLASK

Flask — это микровеб-фреймворк, написанный на Python. Он классифицируется как микрофреймворк, поскольку не требует определенных инструментов или библиотек. В нем нет уровня абстракции базы данных, проверки формы или каких-либо других компонентов, в которых уже существующие сторонние библиотеки предоставляют общие функции. Однако Flask поддерживает расширения, которые могут добавлять функции приложения, как если бы они были реализованы в самом Flask. Существуют расширения для пплатформ объектно-реляционного отображения, проверки форм, обработки загрузки, различных технологий открытой аутентификации и многого другого.

Пример использования кода Flask:

```
from flask import Flask

app = Flask(  name  )

@app.route('/')
def hello():
    return 'Hello, World!'

if   name__      == '  main  ':
    app.run()
```

Здесь создается простое веб-приложение Flask, которое прослушивает запросы по корневому URL-адресу (/) и возвращает строку **'Hello, World!'** в качестве ответа. Когда вы запустите этот код и перейдете по адресу **http://localhost:5000/** в своем веб-браузере, вы должны увидеть сообщение "Hello, World!", отображаемое на странице.

За и против

За

- Представляет собой легкий веб-фреймворк, который легко настроить и использовать.
- Имеет простой и интуитивно понятный API, упрощающий разработку веб-приложений.
- Обеспечивает большую гибкость при интеграции баз данных, позволяя разработчикам использовать любую базу данных по своему выбору.
- Платформа обладает широкими возможностями настройки, доступно большое количество сторонних расширений, позволяющих добавить функциональность вашему приложению.
- Имеет хорошую поддержку сообщества, имеется большое количество учебных пособий, ресурсов и примеров.

Против

- Не такая мощная, как некоторые более крупные веб-фреймворки, такие как Django, что может сделать его менее подходящим для более крупных и сложных проектов.

- Требует от разработчиков принятия большего количества решений о том, как структурировать свое приложение, что может усложнить работу новичкам.

- Не обеспечивает встроенную поддержку таких задач, как проверка формы или аутентификация пользователя, что может увеличить время реализации этих функций.

- Поскольку Flask не является самодостаточной средой, она требует дополнительной настройки и установки, что может оказаться сложной задачей для разработчиков, не знакомых с веб-разработкой.

- Она не подходит для разработки высокопроизводительных веб-приложений, требующих большого количества параллелизма, из-за ее однопоточной природы.

FASTAPI

FastAPI — это современная, быстрая (высокопроизводительная) веб-инфраструктура для создания API с помощью Python 3.6+ на основе стандартных подсказок типов Python. Она разработана так, чтобы быть простым в использовании и понимании, с упором на производительность разработчиков и качество кода.

FastAPI предлагает множество готовых функций, в том числе:

- **Автоматическое создание документации по OpenAPI и JSON Schema**
- **Быстрая асинхронная поддержка с помощью Starlette**
- **Внедрение зависимостей с помощью системы внедрения зависимостей FastAPI**
- **Проверка данных с помощью Pydantic**
- **Интерактивная документация по API с пользовательским интерфейсом Swagger и ReDoc**
- **Встроенная поддержка GraphQL с Graphene**
- **Поддержка WebSocket с помощью Flask-Sockets и поддержка WebSocket**

Все эти функции упрощают создание и поддержку высококачественных API, минимизируя при этом время разработки и уменьшая количество ошибок.

Пример использования кода FastAPI для создания простой конечной точки API:

```
from fastapi import FastAPI

app = FastAPI()

@app.get("/")

async def root():

  return {"message": "Hello World"}
```

При этом создается экземпляр FastAPI под названием **app**. Затем, используя декоратор **@app.get()**, мы создаем конечную точку GET для корневого URL-пути /, которая возвращает объект JSON с ключом "message" и значением "Hello World".

Чтобы запустить приложение, мы можем сохранить этот код в файле, например **main.py**, а затем использовать интерфейс командной строки для запуска сервера:

```
$ uvicorn main:app –reload
```

Эта команда запускает сервер с модулем **main** и экземпляром **app** в качестве приложения. Опция **--reload** автоматически перезагрузит сервер при изменении кода.

После запуска сервера мы можем получить доступ к конечной точке по адресу **http://localhost:8000/** в веб-браузере или сделать запрос GET к URL-адресу, используя такой инструмент, как **curl**, или библиотеку **requests** языка программирования.

В целом, FastAPI предоставляет простой и интуитивно понятный способ создания конечных точек API и обработки HTTP-запросов и ответов.

За и против

За

- FastAPI — одна из самых быстрых веб-платформ Python, скорость которой сопоставима с Node.js и Go.
- Имеет встроенную поддержку синтаксиса async/await, что упрощает написание быстрых, масштабируемых и отзывчивых API.
- FastAPI имеет отличную документацию, включая интерактивный инструмент документации API, который позволяет разработчикам тестировать конечные точки непосредственно из браузера.
- Она поддерживает автоматическую проверку и сериализацию данных, что уменьшает объем стандартного кода, необходимого для создания надежных API.
- FastAPI осуществляет строгую проверку типов и автодополнение кода, что помогает предотвратить ошибки и уменьшает время разработки.
- FastAPI имеет большое и растущее сообщество участников, а это означает, что существует множество плагинов, инструментов и учебных

пособий, которые помогут разработчикам начать работу.

Против

- FastAPI — относительно новая платформа, поэтому при ее использовании с другими библиотеками и инструментами могут возникнуть проблемы со стабильностью и совместимостью.
- Кривая обучения требует сложного подхода, особенно для разработчиков, которые не знакомы с синтаксисом async/await или аннотациями типов.
- FastAPI, возможно, не лучший выбор для небольших проектов, поскольку его преимущества в производительности наиболее заметны в больших и сложных API.
- Поскольку FastAPI построен на основе Starlette, платформы ASGI нижнего уровня, разработчикам может потребоваться изучить обе платформы, чтобы эффективно использовать ее.
- Сильный акцент FastAPI на производительности и проверке типов может быть необходим не для всех проектов и может привести к чрезмерному проектированию и увеличению времени разработки.

DJANGO

Django — это веб-фреймворк Python высокого уровня, который позволяет быстро разрабатывать безопасные и удобные в обслуживании веб-сайты. Он соответствует архитектурному шаблону модель-представление-контроллер (MVC) и предоставляет обширный набор инструментов и библиотек для решения общих задач веб-разработки, таких как маршрутизация URL-адресов, проверка форм и миграция схем базы данных.

Философия дизайна Django подчеркивает возможность повторного использования и «подключаемости» компонентов, что означает, что отдельные части проекта Django можно легко заменять и настраивать в соответствии с конкретными потребностями. Это делает его особенно подходящим для сложных веб-приложений с множеством различных функций и требований.

Одной из ключевых особенностей Django является встроенный интерфейс администрирования, который обеспечивает мощный и настраиваемый веб-интерфейс для управления содержимым сайта и учетными записями пользователей. Django также включает встроенную поддержку различных серверных баз данных, включая PostgreSQL, MySQL и SQLite, а также интеграцию с популярными интерфейсными платформами, такими как React и Angular.

В целом, Django — популярный выбор для веб-разработчиков, стремящихся быстро и эффективно

создавать масштабируемые и поддерживаемые веб-приложения, особенно для тех, кто работает над большими и сложными проектами со множеством различных компонентов и требований.

Пример использования кода для создания простого приложения Django, отображающего сообщение "Hello, World!":

1. Установите Django, выполнив команду **pip install Django** в командной строке или терминале.

2. Создайте новый проект Django, выполнив команду **django-admin startproject myproject** в командной строке или терминале. Это создаст новый каталог под названием **myproject**.

3. Создайте новое приложение Django, выполнив команду **python manage.py startapp myapp** в командной строке или терминале. Это создаст новый подкаталог с именем **myapp** внутри каталога **myproject**.

4. Откройте файл **views.py** в каталоге **myapp** и добавьте следующий код:

```
from django.http import HttpResponse

def hello(request):
    return HttpResponse("Hello, World!")
```

5. Откройте файл **urls.py** в каталоге **myapp** и добавьте следующий код:

```
from django.urls import path
from . import views
```

```
urlpatterns = [
    path('hello/', views.hello, name='hello'),
]
```

6. Откройте файл **urls.py** в каталоге **myproject** и добавьте следующий код:

```
from django.contrib import admin
from django.urls import include, path

urlpatterns = [
    path('admin/', admin.site.urls),
    path('myapp/', include('myapp.urls')),
]
```

7. Запустите сервер Django, выполнив команду **python manage.py runserver** в командной строке или терминале.

8. Откройте веб-браузер и перейдите по адресу **http://127.0.0.1:8000/myapp/hello/**. Вы должны увидеть сообщение "Hello, World!", отображаемое в вашем браузере.

Это очень простой пример приложения Django, но он демонстрирует, как можно создать простую веб-страницу с помощью Python и платформы Django.

За и против

За

- Django предоставляет надежную основу для быстрого и эффективного создания веб-приложений.
- У него большое и активное сообщество, а это значит, что существует множество ресурсов и инструментов, которые помогут разработчикам.

- Django имеет встроенный интерфейс администратора, который упрощает управление данными и контентом.

- Платформа по умолчанию безопасна, что помогает защититься от распространенных уязвимостей веб-приложений.

- Django имеет отличную документацию и хорошо структурированную архитектуру, которая упрощает понимание и поддержку кода.

Против

- Django — относительно тяжелая среда, а это означает, что она может быть менее производительной и более ресурсоемкой, чем некоторые другие варианты.

- Встроенный интерфейс администратора обладает мощными возможностями, но для некоторых проектов он может оказаться недостаточно настраиваемым.

- Кривая обучения Django может быть сложной, особенно для разработчиков, которые плохо знакомы с веб-разработкой или с самим Python.

- Самодостаточный характер платформы иногда может ограничивать возможности, особенно для разработчиков, которые предпочитают большую гибкость и контроль над своим кодом.

DASH

Dash — это платформа веб-приложений для создания интерактивных веб-панелей. Он построен на основе Flask, Plotly.js и React.js, что позволяет легко создавать сложные веб-приложения, управляемые данными. Dash позволяет пользователям создавать интерактивные информационные панели с интерактивными графиками, таблицами и виджетами без необходимости знания HTML, CSS или JavaScript.

С помощью Dash вы можете создавать динамические веб-приложения, способные обрабатывать миллионы точек данных и обновления в реальном времени. Она имеет простой синтаксис и может использоваться с любым стеком обработки данных Python, включая NumPy, Pandas и Scikit-learn. Dash также поддерживает развертывание в облаке с использованием таких сервисов, как Heroku и AWS.

В целом, Dash — это мощный и гибкий инструмент для создания управляемых данными веб-приложений и информационных панелей, которые можно использовать в различных областях, включая финансы, здравоохранение и государственные структуры.

Пример использования кода Python Dash:

```
import dash
import dash_core_components as dcc
import dash_html_components as html

app = dash.Dash()
```

```
app.layout = html.Div(children=[
    html.H1(children='Hello Dash'),

    html.Div(children='''
        Dash: A web application framework for
Python.
    '''),

    dcc.Graph(
        id='example-graph',
        figure={
            'data': [
                {'x': [1, 2, 3], 'y': [4, 1,
2], 'type': 'bar', 'name': 'SF'},
                {'x': [1, 2, 3], 'y': [2, 4,
5], 'type': 'bar', 'name': u'Montréal'},
            ],
            'layout': {
                'title': 'Dash Data
Visualization'
            }
        }
    )
])

if __name__ == '__main__':
```

```
app.run_server(debug=True)
```

Этот код создает простое приложение Dash, отображающее гистограмму. Запустив приложение, вы увидите веб-страницу с заголовком «Hello Dash» и гистограмму, на которой отображаются два набора данных для городов Сан-Франциско и Монреаль.

За и против

За

- Простота изучения и использования, особенно для тех, кто знаком с Python.
- Широкие возможности настройки информационных панелей и визуализаций.
- Обеспечивает интерактивность и обновление данных в режиме реального времени.
- Может быть интегрирован с другими библиотеками и платформами Python.
- Поддерживает как локальное, так и облачное развертывание.

Против

- Ограниченные возможности оформления панели мониторинга и визуализаций.
- Может работать медленнее в случае крупномасштабных приложений.
- Для расширенной настройки требуются знания HTML, CSS и JavaScript.

- Ограниченная поддержка некоторых библиотек визуализации данных.

PYRAMID

Pyramid — это веб-фреймворк, предназначенный для того, чтобы сделать разработку веб-приложений более доступной, предоставляя простой и гибкий подход к созданию веб-приложений. Pyramid — это легкий фреймворк, который легко изучать и использовать. Он основан на стандарте WSGI и предоставляет множество функций, включая маршрутизацию URL-адресов, шаблоны, аутентификацию и интеграцию с базами данных.

Pyramid задуман как модульный и расширяемый. Он предоставляет основные функции, которые можно расширить с помощью надстроек и сторонних библиотек. Он также обладает широкими возможностями настройки, что позволяет разработчикам настраивать поведение платформы в соответствии со своими конкретными потребностями.

Pyramid построен на основе веб-фреймворка Pylons и включает в себя множество функций. Другие популярные веб-фреймворки, включая Django и Ruby on Rails, также влияют на его.

В целом Pyramid — отличный выбор для создания сложных веб-приложений, требующих высокой гибкости и настройки. Его модульность и расширяемость позволяют легко адаптировать его к различным сценариям использования. В то же время основные функции обеспечивают прочную основу для создания надежных и масштабируемых веб-приложений.

Простой пример веб-приложения Pyramid:

Во-первых, вам нужно установить Pyramid, запустив **pip install pyramid** в терминале.

Затем создайте новый файл с именем **app.py** и добавьте следующий код:

```
from wsgiref.simple_server import make_server
from pyramid.config import Configurator
from pyramid.response import Response

def home(request):
    return Response('Hello, Pyramid!')

if __name__ == '__main__':
    with Configurator() as config:
        config.add_route('home', '/')
        config.add_view(home,
route_name='home')
        app = config.make_wsgi_app()
    server = make_server('localhost', 8000,
app
)    print('Server running at
```

```
http://localhost:8000')
    server.serve_forever()
```

Этот код настраивает очень простое веб-приложение Pyramid с одним адресом, который отвечает "Hello, Pyramid!".

Чтобы запустить приложение, просто запустите **python app.py** в своем терминале и перейдите по адресу **http://localhost:8000** в свом веб-браузере. Вы должны увидеть сообщение "Hello, Pyramid!", отображаемое в вашем браузере.

Обратите внимание, что это всего лишь простой пример для начала работы с Pyramid. С его помощью вы можете делать гораздо больше, например использовать шаблоны, работать с базами данных и многое другое.

За и против

За

- Гибкость и простота использования как для небольших, так и для крупномасштабных веб-приложений.
- Предоставляет множество готовых функций, таких как маршрутизация URL-адресов, шаблоны и аутентификация.
- Может использоваться с различными базами данных, такими как PostgreSQL, MySQL, SQLite и Oracle.

- Поддерживает различные функции безопасности, включая предотвращение межсайтового скриптинга (XSS), защиту CSRF и безопасное хеширование паролей.
- Имеет большое и активное сообщество, которое обеспечивает поддержку и обновления.

Против

- Может иметь более крутую кривую обучения по сравнению с другими веб-фреймворками Python, особенно для новичков.
- Имеет более минималистичный подход к веб-разработке, который может потребовать большего количества ручной настройки и установки.
- Может быть менее подходящим для быстрого прототипирования или небольших проектов, поскольку требует больше усилий для установки и настройки.
- Документация может быть менее полной по сравнению с другими веб-платформами Python.

ВЕБ-ПАРСИНГ

Веб-парсинг — это процесс автоматического извлечения данных с веб-сайтов с помощью программного обеспечения или сценария. Он включает в себя получение веб-страниц, анализ содержимого HTML или XML и извлечение полезной информации с веб-страниц, такой как текст, изображения, ссылки и другие данные.

Веб-парсинг можно использовать для различных целей, таких как сбор данных, исследования, мониторинг цен и агрегирование контента. Обычно его используют предприятия; исследователи и аналитики данных собирают данные из нескольких источников, анализируют их и используют для принятия решений.

Парсинг веб-страниц можно выполнять вручную, но чаще всего его автоматизируют с помощью специализированного программного обеспечения или инструментов, известных как веб-парсеры или веб-сканеры. Эти инструменты можно запрограммировать для посещения веб-сайтов, перехода по ссылкам и извлечения определенных данных с веб-страниц в структурированном или неструктурированном формате.

Веб-парсинг вызывает этические и юридические проблемы, в основном при извлечении данных с защищенных авторским правом или частных веб-сайтов. Кроме того, на некоторых веб-сайтах могут быть ограничения на сбор веб-страниц, например

условия обслуживания или файлы robots.txt, которые ограничивают или запрещают сбор веб-страниц. Поэтому важно понимать правовые и этические последствия парсинга веб-страниц и использовать его ответственно и этично.

BEAUTIFULSOUP

BeautifulSoup — это библиотека Python, используемая для парсинга веб-страниц с целью извлечения данных из файлов HTML и XML. Она создает дерево синтаксического анализа из исходного кода страницы, которое можно использовать для извлечения данных в иерархической и более читаемой форме.

BeautifulSoup предоставляет несколько простых методов и идиом Pythonic для навигации, поиска и изменения дерева синтаксического анализа. Она располагается поверх парсера HTML или XML и предоставляет идиомы Python для итерации, поиска и изменения дерева синтаксического анализа.

Это мощный инструмент для парсинга веб-страниц, который можно использовать для различных приложений, таких как интеллектуальный анализ данных, машинное обучение и веб-автоматизация.

Пример использования кода BeautifulSoup:

Предположим, мы хотим получить заголовок и ссылки 5 самых популярных статей с главной страницы New York Times.

```
import
requests
from        bs4        import
BeautifulSoup

url                          =
"https://www.nytimes.com/"
response                     =
```

```
requests.get(url)
soup                                    =
BeautifulSoup(response.content,
'html.parser'
)

articles                                =
soup.find_all('article')[:5]

for article in articles:
    title =
article.find('h2').get_text().strip()
    link = article.find('a')['href']
    print(title)
    print(link)
    print()
```

Этот код отправляет запрос GET на домашнюю страницу New York Times, извлекает HTML-содержимое с помощью библиотеки BeautifulSoup, а затем находит все элементы статьи на странице. Для каждой статьи он извлекает заголовок и ссылку и выводит их на консоль.

Результат
:

```
New York City Vaccine Mandate Takes Effect
for
Private
Employers

https://www.nytimes.com/2022/01/20/nyregion/new
-york-city-vaccine-mandate.html

Wall Street Is Bracing for a Reshuffle

https://www.nytimes.com/2022/01/20/business/wal
l-street-banks-q4-earnings.html

Biden Administration Plans to Move Afghans to
```

```
Third Countries, but Fewer Will Qualify

https://www.nytimes.com/2022/01/20/us/politics/
afghanistan-refugees.html

E.U. Chief Has a Warning for Russia Over
Its Actions in Ukraine

https://www.nytimes.com/2022/01/20/world/europe
/eu-russia-
ukraine.html

Elliott Abrams, Who Oversaw U.S. Policy
in
Latin America, Dies at
73
https://www.nytimes.com/2022/01/20/us/politics/
elliott-abrams-dead.html
```

В этом примере мы используем библиотеку запросов Python для отправки запроса HTTP GET на указанный URL-адрес. Затем мы передаем HTML-содержимое ответа в BeautifulSoup, который анализирует HTML и создает дерево анализа. Мы используем метод **find_all()** для поиска всех элементов **article**, а затем извлекаем информацию о заголовке и ссылке из каждого элемента **article** с помощью метода **find()**. Наконец, мы печатаем заголовок и информацию о ссылке на консоль.

За и против

За

1. Легко освоить: BeautifulSoup — это интуитивно понятная библиотека, которую легко освоить и использовать для парсинга веб-страниц.

2. Гибкость: он может обрабатывать все типы файлов HTML и XML и позволяет работать с различными парсерами.

3. Поддержка селекторов CSS. Вы можете использовать селекторы CSS для поиска определенных элементов HTML, что упрощает сбор данных с веб-страниц.

4. Широкое сообщество. BeautifulSoup имеет большое сообщество пользователей, которые регулярно вносят свой вклад в библиотеку и оказывают поддержку коллегам-разработчикам.

Против

1. Медленная: BeautifulSoup может работать медленно при работе с большими веб-страницами или наборами данных.

2. Ограниченная поддержка JavaScript: она не поддерживает рендеринг JavaScript, что может быть недостатком при парсинге динамических веб-страниц.

3. Ограниченная обработка ошибок. Она не очень хорошо обрабатывает ошибки или исключения, что может затруднить отладку.

4. Нет встроенного сохранения данных: вам придется использовать другие библиотеки или инструменты для хранения данных парсинга.

SCRAPY

Scrapy — это платформа веб-сканирования с открытым исходным кодом, которая используется для извлечения данных с веб-сайтов. Она построена на основе платформы Twisted и предоставляет простой в использовании API для сканирования веб-страниц и извлечения информации. Scrapy предназначена для решения крупномасштабных задач сканирования веб-страниц и может использоваться для извлечения данных для широкого спектра применения, включая интеллектуальный анализ данных, обработку информации и даже для создания интеллектуальных агентов.

Scrapy использует конвейерную архитектуру, которая позволяет пользователям писать повторно используемый код для обработки очищенных данных. Она также включает встроенную поддержку обработки распространенных веб-протоколов, таких как HTTP и HTTPS, а также обработки асинхронных запросов.

В дополнение к мощным возможностям сканирования веб-страниц Scrapy также включает в себя функции парсинга, фильтрации и нормализации данных. Это делает ее отличным инструментом для извлечения структурированных данных из неструктурированных веб-страниц, что может быть сложно сделать с помощью других инструментов веб-парсинга.

Платформа обладает широкими возможностями

настройки и может быть расширена с помощью плагинов и сторонних библиотек. Ее сообщество также очень активно: пользователям доступен широкий спектр ресурсов, на которых они могут учиться и получать помощь по любым проблемам, с которыми они сталкиваются.

Scrapy — это мощная среда веб-сканирования, обеспечивающая большую гибкость и функциональность для извлечения данных с веб-сайтов. Однако для эффективного использования требуются некоторые знания Python и веб-разработки.

Пример использования Scrapy для извлечения цитат с сайта http://quotes.toscrape.com/:

Сначала установите Scrapy, запустив **pip install scrapy** в командной строке или терминале.

Затем создайте новый проект Scrapy, запустив **scrapy startproject quotes_scraper** в командной строке или терминале. Это создаст новый каталог под названием **quotes_scraper**.

Затем перейдите в каталог **spiders** внутри каталога **quotes_scraper** и создайте новый файл с именем **quotes_spider.py**. Добавьте следующий код в этот файл:

```
import scrapy

class QuotesSpider(scrapy.Spider):
    name = "quotes"

    start_urls = [
        'http://quotes.toscrape.com/page/1/',
    ]
```

```
    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'text':
quote.css('span.text::text').get(),
                'author': quote.css('span
small::text').get(),
                'tags': quote.css('div.tags
a.tag::text').getall(),
            }

        next_page = response.css('li.next
a::attr(href)').get()
        if next_page is not None:

            yield response.follow(next_page,
self.parse)
```

Этот спайдер определяет наименование спайдера, начальный URL-адрес для парсинга и метод анализа, который отвечает за извлечение цитат с каждой страницы и переход по ссылкам на следующую страницу, если они существуют.

Чтобы запустить спайдера, перейдите в каталог **quotes_scraper** в командной строке или терминале и запустите сканирование цитат **scrapy crawl quotes**. Это запустит спайдера и выведет очищенные цитаты на вашу консоль.

Вот пример того, как может выглядеть результат:

```
{'text': '“The world as we have created it is a
process of our thinking. It cannot be changed
without changing our thinking.”', 'author':
'Albert Einstein', 'tags': ['change', 'deep-
thoughts', 'thinking', 'world']}
{'text': '“It is our choices, Harry, that show
what we truly are, far more than our
abilities.”', 'author': 'J.K. Rowling', 'tags':
['abilities', 'choices']}
{'text': '“There are only two ways to live your
life. One is as though nothing is a miracle.
The other is as though everything is a
miracle.”', 'author': 'Albert Einstein',
'tags': ['inspirational', 'life', 'live',
'miracle', 'miracles']}
```

Каждая цитата представлена в виде словаря с ключами для текста цитаты, автора и тегов.

За и против

За

- Мощная платформа веб-парсинга, которая обрабатывает асинхронные запросы и поддерживает селекторы XPath и CSS
- Возможность извлекать данные из различных источников, таких как веб-сайты, API и даже базы данных
- Встроенная поддержка выполнения распространенных задач по парсингу веб-страниц, таких как предотвращение обнаружения ботов и управление сеансами пользователей
- Включает встроенную поддержку экспорта собранных данных в различные форматы, включая JSON, CSV и XML
- Поддерживает различные варианты настройки, включая промежуточное ПО, расширения и конвейеры

Против

- Крутой курс обучения, особенно для новичков, которые плохо знакомы с веб-парсингом
- Требуются некоторые знания селекторов XPath и CSS для извлечения данных с веб-страниц
- Не подходит для всех типов задач веб-парсинга, особенно тех, которые требуют более сложной логики парсинга или использования моделей машинного обучения

- Требует дополнительных настроек и установок по сравнению с другими более простыми библиотеками парсинга веб-страниц, что может занять много времени

SELENIUM

Selenium — это библиотека, которая обеспечивает веб-автоматизацию и тестирование, предоставляя возможность программного взаимодействия с веб-страницами. Она позволяет разработчикам автоматизировать веб-браузеры, моделировать взаимодействие пользователей с веб-сайтами и собирать веб-данные.

Selenium широко используется при тестировании и автоматизации веб-приложений. Она поддерживает различные языки программирования, включая Python, Java, C#, Ruby и JavaScript, и может работать с различными браузерами, такими как Chrome, Firefox, Safari и Internet Explorer.

С помощью Selenium вы можете создавать сценарии для автоматизации повторяющихся задач, таких как заполнение форм, нажатие кнопок, навигация по страницам и извлечение данных с веб-страниц.

В целом, Selenium — мощный инструмент для веб-автоматизации и тестирования, который может значительно упростить задачи, которые в противном случае были бы трудоемкими и затратными по времени.

Пример использования кода Selenium для парсинга веб-страниц:

```
from selenium import webdriver
from selenium.webdriver.common.by import By

# Set up the driver
driver =
```

```
webdriver.Chrome('path/to/chromedriver')

# Navigate to the website you want to scrape
driver.get('https://www.example.com'
)


# Find the element you want to interact with
and perform actions
element = driver.find_element(By.XPATH,
'//button[@id="button-id"]')
element.click()

# Extract the data you want from the website
data_element = driver.find_element(By.XPATH,
'//div[@class="data-class"]')
data = data_element.text

# Clean up and close the driver
driver.quit()
```

В этом примере мы используем драйвер Chrome и переходим на веб-сайт. Затем мы находим элемент кнопки и нажимаем на него, что вызывает загрузку некоторых данных на страницу. Затем мы находим элемент, содержащий данные, которые мы хотим очистить, и извлекаем его текст. Наконец, мы очищаем и закрываем драйвер.

Обратите внимание, что парсинг веб-страниц может быть «серой зоной» с юридической и этической точки зрения, а условия обслуживания некоторых веб-сайтов могут запрещать это. Обязательно ознакомьтесь с политикой веб-сайта и будьте этичны в своих действиях по парсингу данных.

За и против

За

- Можно взаимодействовать с веб-страницами, как если бы вы использовали веб-браузер, что позволяет выполнять более сложные задачи по сбору данных

- Поддерживает широкий спектр браузеров, включая Chrome, Firefox, Safari и Internet Explorer

- Может обрабатывать динамический контент, загружаемый с помощью JavaScript, AJAX и других технологий

- Поддерживает автономный просмотр, что позволяет запускать задачи парсинга без графического пользовательского интерфейса

- Поддерживает различные языки программирования, включая Python, Java, Ruby и C#

Против

- Может работать медленнее, чем другие библиотеки веб-парсинга, поскольку использует автоматизацию браузера

- Требует дополнительных настроек и установок по сравнению с другими библиотеками

- Может быть более ресурсоемкой, поскольку для запуска требуется экземпляр браузера

- Может не подходить для всех задач парсинга веб-страниц, особенно тех, которые требуют высокой скорости и масштабируемости