**Дипломная работа.**

**Сравнение различных библиотек для машинного обучения: scikit-learn, TensorFlow и PyTorch: Реализовать задачи классификации и регрессии с использованием scikit-learn, TensorFlow и PyTorch, сравнить их производительность и удобство использования.**

**Бондарев Дмитрий Алексеевич**

**Содержание дипломной работы.**

1. Введение.
2. Где используется машинное обучение?
3. Что такое машинное обучение?
4. Что включают в себя библиотеки **scikit-learn, TensorFlow и PyTorch?**
5. Примеры классификации и регрессии с использованием этих библиотек.
6. Сравнение Библиотек.
7. Рекомендации по использованию.

1.Введение .

Мы живем в эпоху научно технического прогресса, люди желая облегчить свой труд используют машины, различные компьютерные программы. Также изобретён Искусственный Интеллект (ИИ) который помогает во многих областях жизни.

Для того чтобы машины выполняли функции людей их нужно обучить. Для этого нужны специалисты по машинному обучению. В каких областях может найти себя такой специалист?

2.Где используется Машинное обучение?

1. **Финтех: кредитный скоринг, защита клиентов, поиск мошенников**

В финтехе и банках очень много решений сейчас принимается с помощью машинного обучения. Алгоритм помогает, например, решать задачи кредитного скоринга. Обрабатывая данные о клиенте из анкеты и открытых источников, система делает вывод о том, стоит выдавать кредит клиенту или нет. Это помогает менеджерам принимать решения и снижать риски для банка.

Также ML-модели могут отслеживать платежи и находить среди них мошеннические, когда кто-то украл карту клиента и пытается вывести с нее деньги. Машинное обучение помогает банкам искать клиентов в открытых источниках, например в реестрах юрлиц, отслеживать их отток, понимать, что клиент собирается уйти, и дает рекомендации, чтобы его удержать.

2. **Медицина: диагностика болезней на ранней стадии**

Диагностика — одно из важных направлений машинного обучения. Сейчас законодательно все устроено так, что поставить пациенту диагноз и назначить лечение может только человек. Все, что разрабатывается в сфере машинного обучения, — это системы помощи врачу.

Чаще всего для анализа при помощи моделей используются медицинские изображения: МРТ, КТ, флюорография. В текстовом анализе, как правило, написано число по определенному параметру (например сахар в крови) и границы допустимых значений. Такой результат может прочитать любой врач, они понимают, что означает выход за границу допустимых значений. С медицинскими изображениями все сложнее. Безусловно, технически мы можем сделать идеальное изображение всего, что у человека есть внутри. Но для этого нужно хорошо просветить тело радиацией, и человеку от этого будет очень плохо. Если же мы делаем изображения, которые не вредят здоровью, то качественную картинку получить невозможно. Она чаще всего зашумленная, и на ней сложно что-либо различить. Чтобы работать с такими изображениями, врачи специально учатся диагностике. К сожалению, специалистов, которые могут читать медицинские изображения, не хватает. Многие больницы сейчас покупают качественное оборудование, но врачей, которые могут интерпретировать результаты, просто нет.

Чем здесь могут помочь системы помощи медицинским специалистам? При помощи алгоритмов компьютерного зрения они анализируют изображение и выделяют области интереса: есть ли на картинке что-то нетипичное, на что стоит обратить внимание. Проанализировав область интереса, модель может определить, например, что на изображении — с высокой вероятностью туберкулез. После этого анализа изображение смотрит врач и подтверждает или опровергает диагноз, отправляет пациента на дообследование. Актуальное направление разработки сейчас — обследования, которые помогают видеть заболевания на ранней стадии развития, когда врач может еще не замечать их на медицинском изображении.

3. **Промышленность: контроль качества**

В промышленности очень много данных и очень много задач для алгоритмов: безопасность на производстве, автоматизация, контроль качества продукции. Трубы должны быть качественными, потому что, если их прорвет, это будет экологическая катастрофа. Ущерб от такого происшествия несопоставим со стоимостью трубы, поэтому для ТМК очень важно делать продукт высокого качества.

Вопрос в том, как это качество контролировать. На одном из этапов трубу просвечивают рентгеновским излучением. Снимок позволяет увидеть внутренние дефекты. Чтобы проконтролировать качество, нужен специалист, который будет сидеть и постоянно смотреть на эти изображения. Понятно, что человек не может проводить такой контроль стабильно качественно. Нужен прототип системы компьютерного зрения, которая отслеживает дефекты и сообщает о них сотруднику.

4. **Автомобили без водителей**

На сегодня в России законодательно разрешен проезд автомобилей без водителя в Москве, Московской области и Татарстане. По Иннополису уже ездят Яндекс. Такси без водителя. Проблема этого направления в том, что цена ошибки тут очень высока. Когда разработчики научатся решать проблемы, беспилотники появятся на улицах всех городов.

КАМАЗ занимается разработкой грузовиков без водителя, Россельмаш — беспилотных комбайнов, а Уралмаш начал думать о том, чтобы производить экскаваторы без водителя. Это очень перспективное направление.

5. **Фармакология: проектирование лекарств**

Один из последних прорывов машинного обучения в науках о жизни — при помощи нейронных сетей научились определять трехмерную структуру белка. Это упрощает процесс разработки новых лекарств. Например, когда мы расшифровываем геном какого-либо вируса, получаем одномерную последовательность, в то время как в реальности белки трехмерные. Чтобы определить трехмерную структуру генома, требуются длительные, на несколько месяцев эксперименты. А для того чтобы получить одно лекарство, таких экспериментов нужно несколько. Это долго и дорого.

Выяснилось, что при помощи нейросетей можно получить трехмерную структуру белка и определить, будет ли он взаимодействовать с вирусом нужным образом. Такую модель создала компания DeepMind. Для разработки алгоритма AlphaFold создатели использовали информацию о структурах 170 тысяч белков.

Эта разработка позволит быстрее проектировать лекарства и вакцины и с высокой точностью прогнозировать их эффективность.

6. **Информационная безопасность**

Как сейчас работают системы обнаружения вторжений? По так называемой сигнатуре атаки: когда вторжение произошло, производители антивирусов изучают, какие действия были произведены. Их записывают в сценарий антивируса: если происходят вот такие действия, то это атака вот такого типа и нужно ее остановить. Но это означает, что нет возможности остановить новые атаки — такие, которые раньше не происходили. Машинное обучение в информационной безопасности используется как раз, чтобы обнаруживать атаки нового вида, которых раньше никто никогда не видел.

7.**Безопастность и охрана людей**

Система определения лиц и слежка за потенциальными преступниками. Очень хорошее подспорье для служб безопасности. Так же сюда можно добавить распознавание номеров автомобилей и моделей автомобилей.

Но для того, чтобы машины выполняли эти действия их нужно обучить.

Для этого необходимо машинное обучение. Что это такое?

3.Что такое машинное обучение?

Из источника Urban University

Машинное обучение (ML) — это область искусственного интеллекта, которая занимается разработкой алгоритмов и статистических моделей, позволяющих компьютерам выполнять конкретные задачи без явного программирования на выполнение этих задач. Основная идея состоит в том, чтобы создать модель, способную "учиться" на данных и делать прогнозы или принимать решения на основе этого обучения.

Основные принципы машинного обучения

1. Обучение на данных: Модели машинного обучения обучаются на наборе данных. Этот процесс включает в себя использование алгоритмов, которые анализируют данные и выявляют в них закономерности.
2. Обобщение: после обучения на данных модель должна быть способна обобщать свои знания на новые, невиданные ранее данные. Это означает, что она должна давать правильные ответы не только на обучающих данных, но и на любых других данных из той же области.

10. Итеративное улучшение: Модели машинного обучения часто улучшаются итеративно. Это включает в себя оценку модели, выявление её недостатков и последующую настройку параметров или алгоритма для повышения точности.

Алгоритмы машинного обучения

Машинное обучение делится на три основных типа, каждый из которых включает в себя множество алгоритмов:

1. Обучение с учителем (Supervised Learning): Модель обучается на размеченных данных, где каждой входной информации соответствует правильный выход. Примеры алгоритмов: линейная регрессия, логистическая регрессия, деревья решений, случайный лес (Random Forest), поддерживающие векторы (SVM), нейронные сети.
2. Обучение без учителя (Unsupervised Learning): Модель обучается на неразмеченных данных и пытается выявить скрытые структуры. Примеры алгоритмов: кластеризация (например, k-средних), ассоциативные правила, метод главных компонент (PCA).
3. Обучение с подкреплением (Reinforcement Learning): Модель обучается на основе взаимодействия с окружающей средой, получая вознаграждения или наказания за свои действия. Примеры алгоритмов: Q-learning, SARSA, глубокое обучение с подкреплением.

Этапы работы с данными

1. Сбор данных: Первый этап включает сбор данных из различных источников. Данные могут быть получены из баз данных, файлов, веб-скрапинга, API и других источников.
2. Очистка данных: на этом этапе данные очищаются от шума и ошибок. Это может включать удаление или исправление пропущенных значений, обработку выбросов, устранение дубликатов и исправление ошибок в данных.
3. Подготовка данных: включает в себя приведение данных к такому виду, который может быть использован моделью. Это может включать нормализацию или стандартизацию данных, кодирование категориальных переменных, разбиение данных на обучающий и тестовый наборы.
4. Анализ данных: Предварительный анализ данных для понимания их структуры и свойств. Этот этап может включать визуализацию данных, вычисление статистических характеристик и выявление корреляций между переменными.

4.Что включают в себя библиотеки scikit-learn, TensorFlow и PyTorch?

Scikit-learn — это мощная и широко используемая библиотека для традиционного машинного обучения на языке Python. Он предоставляет полный набор инструментов для предварительной обработки данных, обучения моделей, оценки и настройки гиперпараметров, что делает его незаменимым ресурсом как для начинающих, так и для опытных практиков.

Подготовка данных

Прежде чем приступить к обучению моделей, крайне важно эффективно подготовить данные. Scikit-learn предлагает для этого различные утилиты:

Разделение данных: используется для разделения набора данных на обучающий и тестовый наборы, гарантируя, что модель оценивается на основе невидимых данных.train\_test\_split

Масштабирование и нормализация: Функции любят и помогают стандартизировать ваши функции, что особенно важно для алгоритмов, чувствительных к масштабам функций.StandardScalerMinMaxScaler

Кодирование категориальных переменных: используйте или преобразуйте категориальные данные в числовой формат, который могут обрабатывать алгоритмы машинного обучения.OneHotEncoderLabelEncoder

Scikit-learn поддерживает множество алгоритмов, от линейных моделей до ансамблевых методов.

Оценка модели

Оценка модели является ключом к пониманию ее производительности. Scikit-learn предоставляет несколько метрик:

Точность: доля правильных прогнозов.

Матрица несоответствий: таблица, описывающая производительность модели классификации.

Перекрестная проверка: используется для оценки того, как результаты статистического анализа будут обобщены в независимый набор данных.cross\_val\_score

Настройка гиперпараметров

Оптимизация гиперпараметров модели может значительно повысить производительность. Scikit-learn предлагает такие инструменты, как и для автоматизации этого/

Заключение

Таким образом, Scikit-learn является бесценным инструментом для решения традиционных задач машинного обучения. Удобный API, обширная документация и надежная поддержка сообщества делают его лучшим выбором для разработчиков, желающих внедрить решения машинного обучения. При сравнении Scikit-learn с такими фреймворками, как TensorFlow и PyTorch, он выделяется своей простотой и эффективностью в обработке традиционных задач машинного обучения, что делает его идеальным для проектов, не требующих возможностей глубокого обучения.

TensorFlow — это комплексный фреймворк машинного обучения с открытым исходным кодом, разработанный командой Google Brain. Он предназначен для упрощения разработки моделей глубокого обучения, предоставляя обширную экосистему инструментов, библиотек и ресурсов, которые позволяют исследователям и разработчикам эффективно создавать и развертывать приложения машинного обучения.

Ключевые особенности TensorFlow

Автоматическая дифференцировка: TensorFlow включает в себя мощные возможности автоматической дифференцировки, которые необходимы для обучения моделей глубокого обучения.

Многоязычная поддержка: Он поддерживает различные языки программирования, включая Python, C++ и JavaScript, что делает его универсальным для различных сред разработки.

Предварительно обученные модели: TensorFlow предлагает коллекцию предварительно обученных моделей и наборов данных, что позволяет разработчикам использовать существующую работу и ускорять свои проекты.

Статистические распределения: Библиотека предоставляет широкий спектр статистических распределений, которые могут быть полезны для различных задач машинного обучения.

Преимущества использования TensorFlow

Интеграция с Keras: TensorFlow оптимизирован для Keras, что позволяет легко создавать модели и экспериментировать с высокоуровневыми API.

Инструменты визуализации: Он предлагает надежные инструменты визуализации, такие как TensorBoard, которые помогают в мониторинге и отладке процесса обучения.

Гибкое развертывание: TensorFlow поддерживает развертывание на нескольких платформах, включая веб-интерфейсы, мобильные устройства, периферийные устройства и серверы, что делает его подходящим для широкого спектра приложений.

Ограничения TensorFlow

Поддержка ОС Windows: TensorFlow имеет ограниченную поддержку для пользователей ОС Windows, что может быть недостатком для разработчиков, работающих в этой среде.

Поддержка графических процессоров: В первую очередь он поддерживает графические процессоры NVIDIA и графические процессоры Python, что может ограничивать возможности для некоторых пользователей.

Применение TensorFlow

TensorFlow — это мощный инструмент для решения различных задач машинного обучения, в том числе:

Классификация изображений: эффективная классификация изображений с помощью сверточных нейронных сетей (СНС).

Обработка естественного языка: создание моделей для таких задач, как анализ тональности и перевод с одного языка на другой.

Распознавание речи: Разработка систем, которые могут понимать и обрабатывать человеческую речь.

Рекомендательные системы: создавайте персонализированные рекомендации на основе поведения пользователей.

Робототехника: реализация алгоритмов машинного обучения для роботизированного управления и автоматизации.

Медицинская визуализация: анализ медицинских изображений для диагностики и исследований.

Финансовое прогнозирование: используйте машинное обучение для прогнозирования рыночных тенденций и финансовых результатов.

Скорость обучения в TensorFlow

Скорость обучения является важным гиперпараметром в TensorFlow, который определяет, как часто веса модели обновляются во время обучения. Оптимальная скорость обучения может варьироваться в зависимости от конкретной задачи, архитектуры модели и размера набора данных. Обычно используются небольшие положительные числа, такие как 0,001 или 0,01, но часто необходимо экспериментировать, чтобы найти наиболее подходящий для данной задачи вариант.

5.Реализация задач классификации и регрессии с использованием scikit-learn, TensorFlow и PyTorch, сравнить их производительность и удобство использования.

Для реализации задач классификации и регрессии с использованием библиотек scikit-learn, TensorFlow и PyTorch, мы можем создать примеры на основе простых наборов данных, таких как `Iris` для классификации и `[Real estate.csv](https://github.com/sin013/Diplom/blob/main/datasets/Real%20estate.csv)` для регрессии. После этого мы можем сравнить производительность и удобство использования каждой библиотеки.

Благодаря быстрому развитию машинного обучения (ML) PyTorch стал ведущей платформой для исследований и создания прототипов. Его динамический вычислительный граф обеспечивает гибкое построение моделей, что делает его особенно привлекательным для исследователей, которым необходимо экспериментировать с различными архитектурами и алгоритмами.

Основные возможности PyTorch

Динамические вычислительные графы: в отличие от статических графов, используемых в других средах, динамические графы PyTorch позволяют разработчикам изменять поведение сети на лету, что имеет решающее значение для отладки и экспериментов.

Богатая экосистема: PyTorch поддерживает множество библиотек и инструментов, таких как torchvision для компьютерного зрения и torchtext для обработки естественного языка, повышая удобство использования в различных областях.

Сильная поддержка сообщества: активное сообщество вносит свой вклад в множество учебных пособий, форумов и ресурсов, облегчая новичкам начало работы, а опытным пользователям — поиск решений сложных проблем.

Оптимизация производительности

PyTorch также легко интегрируется с различными библиотеками оптимизации, что позволяет эффективно обучать модели. Использование таких библиотек, как TorchScript, позволяет пользователям преобразовывать свои модели в формат, который можно оптимизировать и запускать в производственной среде, обеспечивая плавный переход от исследования к развертыванию.

Сравнение с другими фреймворками

При сравнении PyTorch с другими фреймворками, такими как TensorFlow и scikit-learn, становится очевидно, что у каждой из них есть свои сильные стороны. В то время как TensorFlow часто предпочитают использовать в производственных средах из-за его надежных возможностей развертывания, PyTorch отлично подходит для исследовательских условий, где гибкость и простота использования имеют первостепенное значение. Выбор между scikit-learn, TensorFlow и PyTorch в конечном итоге зависит от конкретных потребностей проекта и знакомства команды с каждой платформой.

В заключение следует отметить, что PyTorch выделяется как мощный инструмент для исследователей и разработчиков, которые хотят быстро создавать прототипы и выполнять итерации своих моделей машинного обучения. Его уникальные функции и сильная поддержка сообщества делают его отличным выбором для продвижения исследований в области машинного обучения.

### Задача 1: Классификация с использованием scikit-learn, TensorFlow и PyTorch

from sklearn.datasets import load\_iris

from sklearn.linear\_model import LogisticRegression

# Загрузить данные

iris = load\_iris()

# Разделить данные на тренировочный и тестовый наборы

X\_train, X\_test, y\_train, y\_test = train\_test\_split(iris.data, iris.target, test\_size=0.25)

# Создать модель

model = LogisticRegression()

# Обучить модель

model.fit(X\_train, y\_train)

# Оценить модель

score = model.score(X\_test, y\_test)

print("Точность:", score)

# Создать таблицу зависимостей

y\_pred = model.predict(X\_test)

df = pd.DataFrame({'Реальное значение': y\_test, 'Предсказанное значение': y\_pred})

df['Правильность'] = df['Реальное значение'] == df['Предсказанное значение']

# Рассчитать точность для каждого класса

accuracy = df.groupby('Реальное значение')['Правильность'].mean()

# Отобразить таблицу зависимостей

print(df)

# Отобразить точность для каждого класса

print("Точность для каждого класса:")

print(accuracy)

#### 2. Классификация с использованием TensorFlow

import tensorflow as tf

from sklearn.datasets import load\_iris

from sklearn.model\_selection import train\_test\_split

# Загрузка данных

iris = load\_iris()

X = iris.data

y = iris.target

# Разделение данных на обучающий и тестовый наборы

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, test\_size=0.2, random\_state=42)

# Кодирование целевых значений с помощью кодирования с одним горячим вектором

y\_train = tf.keras.utils.to\_categorical(y\_train, num\_classes=3)

y\_test = tf.keras.utils.to\_categorical(y\_test, num\_classes=3)

# Создание модели

model = tf.keras.Sequential([

tf.keras.layers.Dense(units=10, activation='relu', input\_shape=(4,)),

tf.keras.layers.Dense(units=10, activation='relu'),

tf.keras.layers.Dense(units=3, activation='softmax')

])

# Компиляция модели

model.compile(optimizer='adam', loss='categorical\_crossentropy', metrics=['accuracy'])

# Обучение модели

model.fit(X\_train, y\_train, epochs=100, batch\_size=10)

# Оценка модели

loss, accuracy = model.evaluate(X\_test, y\_test)

print(f"Потеря: {loss}")

print(f"Точность: {accuracy}")

# Сохранение модели

model.save('model.keras')

# Создание таблицы зависимостей

y\_pred = model.predict(X\_test)

df = pd.DataFrame({'Реальное значение': y\_test.argmax(axis=1), 'Предсказанное значение': y\_pred.argmax(axis=1)})

df['Правильность'] = df['Реальное значение'] == df['Предсказанное значение']

# Рассчитать точность для каждого класса

accuracy = df.groupby('Реальное значение')['Правильность'].mean()

# Отобразить таблицу зависимостей

print(df)

# Отобразить точность для каждого класса

print("Точность для каждого класса:")

print(accuracy)

#### 3. Классификация с использованием PyTorch

import torch

from torch.utils.data import DataLoader

from sklearn import datasets

# Загрузить и подготовить данные

iris\_data, iris\_target = datasets.load\_iris(return\_X\_y=True, as\_frame=True)

iris\_data\_tensor = iris\_data\_tensor.float()

iris\_target\_tensor = iris\_target\_tensor.long()

iris\_loader = DataLoader(torch.utils.data.TensorDataset(iris\_data\_tensor, iris\_target\_tensor), batch\_size=64)

# Создать модель

class IrisModel(torch.nn.Module):

def \_\_init\_\_(self):

super(IrisModel, self).\_\_init\_\_()

self.fc1 = torch.nn.Linear(4, 10)

self.fc2 = torch.nn.Linear(10, 3)

def forward(self, x):

x = torch.relu(self.fc1(x))

x = torch.softmax(self.fc2(x), dim=1)

return x

model = IrisModel()

# Оптимизатор и функция потерь

optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

loss\_fn = torch.nn.CrossEntropyLoss()

# Обучить модель

for epoch in range(10):

for batch in iris\_loader:

data, labels = batch

outputs = model(data)

loss = loss\_fn(outputs, labels)

optimizer.zero\_grad()

loss.backward()

optimizer.step()

# Оценить модель

with torch.no\_grad():

correct = 0

total = 0

for batch in iris\_loader:

data, labels = batch

outputs = model(data)

\_, predicted = torch.max(outputs, dim=1)

correct += (predicted == labels).sum().item()

total += labels.size(0)

score = correct / total

print("Точность:", score)

# Создание таблицы зависимостей

y\_pred = model(iris\_data\_tensor.to(torch.device("cpu")))

df = pd.DataFrame({'Реальное значение': iris\_target, 'Предсказанное значение': y\_pred.argmax(axis=1)})

df['Правильность'] = df['Реальное значение'] == df['Предсказанное значение']

# Рассчитать точность для каждого класса

accuracy = df.groupby('Реальное значение')['Правильность'].mean()

# Отобразить таблицу зависимостей

print(df)

# Отобразить точность для каждого класса

print("Точность для каждого класса:")

print(accuracy)

### Задача 2: Регрессия с использованием scikit-learn, TensorFlow и PyTorch

#### 1. Регрессия с использованием scikit-learn

import pandas as pd

from sklearn.linear\_model import LinearRegression

from sklearn.model\_selection import train\_test\_split

import matplotlib.pyplot as plt

# Загрузка датасета

data = pd.read\_csv('/Real estate.csv')

# Разделение на обучающий и тестовый наборы

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data[['X1 transaction date', 'X2 house age', 'X3 distance to the nearest MRT station', 'X4 number of convenience stores', 'X5 latitude', 'X5 latitude']], data['Y house price of unit area'], test\_size=0.2)

# Создание и обучение модели

model = LinearRegression()

model.fit(X\_train, y\_train)

# Оценка модели

score = model.score(X\_test, y\_test)

print(f"Точность модели scikit-learn: {score}")

# Отображение графика обучения

plt.plot(model.coef\_)

plt.xlabel('Признак')

plt.ylabel('Коэффициент')

plt.title('График обучения модели')

plt.show()

#### 2. Регрессия с использованием TensorFlow

import pandas as pd

import tensorflow as tf

from tensorflow.keras.models import Sequential

from tensorflow.keras.layers import Dense

import matplotlib.pyplot as plt

# Загрузка датасета

data = pd.read\_csv('/Real estate.csv')

# Разделение на обучающий и тестовый наборы

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data[['X1 transaction date', 'X2 house age', 'X3 distance to the nearest MRT station', 'X4 number of convenience stores', 'X5 latitude', 'X5 latitude']], data['Y house price of unit area'], test\_size=0.2)

# Создание модели

model = Sequential()

model.add(Dense(1, input\_shape=(6,)))

# Компиляция модели

model.compile(optimizer='adam', loss='mse')

# Обучение модели

history = model.fit(X\_train, y\_train, epochs=100, validation\_data=(X\_test, y\_test))

# Оценка модели

score = model.evaluate(X\_test, y\_test)

print(f"Точность модели TensorFlow: {score}")

# Визуализация обучения

plt.plot(history.history['loss'], label='Ошибка обучения')

plt.plot(history.history['val\_loss'], label='Ошибка валидации')

plt.xlabel('Эпоха')

plt.ylabel('Среднеквадратичная ошибка')

plt.legend()

plt.show()

#### 3. Регрессия с использованием PyTorch

import pandas as pd

import torch

import torch.nn as nn

import matplotlib.pyplot as plt

# Загрузка датасета

data = pd.read\_csv('/Real estate.csv')

# Разделение на обучающий и тестовый наборы

X\_train, X\_test, y\_train, y\_test = train\_test\_split(data[['X1 transaction date', 'X2 house age', 'X3 distance to the nearest MRT station', 'X4 number of convenience stores', 'X5 latitude', 'X5 latitude']], data['Y house price of unit area'], test\_size=0.2)

# Преобразование данных в тензоры

X\_train = torch.Tensor(X\_train.values).float()

y\_train = torch.Tensor(y\_train.values).float().view(-1, 1)

X\_test = torch.Tensor(X\_test.values).float()

y\_test = torch.Tensor(y\_test.values).float().view(-1, 1)

# Создание модели

model = nn.Linear(6, 1)

# Определение функции потерь и оптимизатора

loss\_fn = nn.MSELoss()

optimizer = torch.optim.Adam(model.parameters(), lr=0.01)

# Обучение модели

loss\_history = []

for epoch in range(100):

# Прямой проход

y\_pred = model(X\_train)

# Вычисление ошибки

loss = loss\_fn(y\_pred, y\_train)

loss\_history.append(loss.item()) # Сохранение истории ошибки

# Обратный проход

optimizer.zero\_grad()

loss.backward()

# Обновление параметров

optimizer.step()

# Оценка модели

with torch.no\_grad():

y\_pred = model(X\_test)

score = loss\_fn(y\_pred, y\_test)

print(f"Точность модели PyTorch: {score}")

# Отображение графика обучения

plt.plot(loss\_history)

plt.xlabel('Эпоха')

plt.ylabel('Ошибка')

plt.title('График обучения модели')

plt.show()

6. Сравнение их производительности и удобства использования.

1. Производительность:

- Все модели могут показать конкурентоспособные результаты на простых задачах, но более сложные модели и большие наборы данных могут требовать больше времени для обучения в TensorFlow и PyTorch по сравнению с scikit-learn.

- TensorFlow и PyTorch предоставляют более мощные инструменты для настройки архитектуры модели, что позволяет достигать лучшей производительности на сложных задачах.

2. Удобство использования:

- scikit-learn: отличается простым и интуитивно понятным интерфейсом. Идеально подходит для быстрого прототипирования и работы с классическими алгоритмами машинного обучения.

- TensorFlow: обеспечивает высокий уровень гибкости для создания сложных моделей глубокого обучения, но требует больше кода и времени для настройки.

- PyTorch: более удобен в использовании, особенно для исследования, благодаря динамической вычислительной графике и гибкости. Он также упрощает отладку.

Scikit-learn против TensorFlow

Scikit-learn — широко используемая библиотека машинного обучения с открытым исходным кодом для Python. Она построена на основе и интегрируется с такими часто используемыми библиотеками, как NumPy, SciPy, Matplotlib и pandas, что делает ее доступной и универсальной.

TensorFlow , также библиотека машинного обучения с открытым исходным кодом, специализируется на глубоком обучении и нейронных сетях. TensorFlow поддерживает несколько языков программирования, таких как Python, C/C++, Java и Javascript, и другие.

Рассмотрите scikit-learn, если вы новичок в машинном обучении или разрабатываете что-то с использованием алгоритмов, не основанных на нейронных сетях.

Рассмотрите TensorFlow, если вы хотите использовать подход глубокого обучения в сочетании с аппаратным ускорением с помощью графических процессоров и TPU или на кластере компьютеров (что scikit-learn изначально не поддерживает).

PyTorch против scikit-learn

PyTorch — это библиотека программного обеспечения глубокого обучения для Python, C++ и Julia. PyTorch в основном используется для сквозного построения и обучения глубоких нейронных сетей с возможностью создания пользовательских моделей и алгоритмов обучения.

Scikit-learn — это библиотека для традиционных алгоритмов машинного обучения, используемых для кластеризации, классификации, регрессии и т. д. Природа Scikit-learn как «черного ящика» делает ее более доступной для тех, кто относительно недавно начал заниматься машинным обучением.

Рассмотрите PyTorch, если вы разрабатываете приложения, в которых есть задачи, требующие больших вычислительных затрат, такие как обработка естественного языка, компьютерное зрение и т. д. С Pytorch вы также можете использовать ускорение графического процессора в своих интересах.

Рассмотрите scikit-learn, если вы разрабатываете небольшой исследовательский проект, не требующий значительного объема данных. С scikit-learn вы будете фокусироваться не на настройке, а на скорости и удобстве использования алгоритмов машинного обучения.

PyTorch против TensorFlow

PyTorch — это фреймворк глубокого обучения с питоническим и объектно-ориентированным подходом. PyTorch имеет больше возможностей отладки и тестирования, чем TensorFlow.

TensorFlow — это низкоуровневая библиотека глубокого обучения, которая обеспечивает рабочие процессы для высокоуровневых API, таких как Keras, хотя и с меньшей вычислительной мощностью. TensorFlow в настоящее время используется более широко, чем PyTorch.

Если Python играет центральную роль в вашей разработке, рассмотрите PyTorch с его многочисленными возможностями отладки.

Рассмотрите TensorFlow, если вам нужна библиотека, совместимая с различными языками программирования, такими как C/C++, Java, JavaScript, Go и т. д. TensorFlow также имеет обширную поддержку нескольких платформ.

Заключение

Сценарий выбора библиотеки зависит от задачи. Если вы ищете простоту и скорость, scikit-learn может быть лучшим выбором для традиционных методов. Если ваши задачи связаны с глубоким обучением, TensorFlow и PyTorch предпочтительнее, причём PyTorch может быть удобнее на этапе prototyping и исследований.