УДК 004.021

СРАВНЕНИЕ ПРОИЗВОДИТЕЛЬНОСТИ РАЗЛИЧНЫХ ПОДХОДОВ В ЗАДАЧЕ КЛАССИФИКАЦИИ

А. О. Истягин, О. В. Рычка
Донецкий национальный технический университет
e-mail: ist_75@mail.ru, red_rose_2005@mail.ru

Аннотация
В статье рассмотрены различные подходы в решении задачи классификации, реализована нейронная сеть для классификации типов номеров отелей, проведен анализ производительности работы регулярных выражений и нейронных сетей, построены графики зависимости скорости классификации больших объемов данных от выбранного алгоритма, сделаны выводы о целесообразности и уместности применения различных подходов. Сделан вывод, что подход с применением нейронной сети обрабатывает данные значительно быстрее, однако требует дополнительных ресурсов для внедрения.

Введение

После завершения пандемии и периода самоизоляции вновь актуальным стал отельный бизнес. В мире существуют сотни тысяч различных отелей, гостиниц и курортных домов, в каждом – десятки номеров, комнат и тарифов для проживания. Для привлечения большего количества гостей отели заключают контракты с фирмами-агрегаторами, которые собирают в общую базу данных множество отелей и предлагают их клиентам, в том числе другим агрегаторам (далее: поставщикам), каждый из которых, разумеется, увеличивает исходные цены для извлечения собственной прибыли. Подобная цепочка наценок может проходить через множество фирм, прежде чем дойдет до конечного потребителя. Каждая такая фирма должна правильно систематизировать собственные категории номеров и спальных мест всех своих отелей и отелей других поставщиков, с которыми заключен контракт. Единого правила или алгоритма классификации не существует, и все поставщики делают это по-своему, что значительно усложняет классификацию типов номеров.

Постановка задачи

Целью данного исследования является определение наиболее оптимального механизма определения типов номеров отельного бизнеса.

Рассмотрим функционал подробнее: на вход механизма должна подаваться строка, содержащее название номера, на выходе алгоритм должен выдавать одну из записей в заранее сформированных справочниках (тип номера, тип кровати и подтип номера соответственно, в порядке убывания приоритета) [1, 2].

Для лучшего понимания приведем некоторый пример. Пусть в справочнике «тип номера» существуют следующие типы: «стандарт», «люкс», «эконом» и другие, в справочнике «тип кровати»: «одноместная», «две одноместные», «двухместная», «двухместная king size» и другие, в «подтипе номера»: «апартаменты», «вид на море», «вид на горы», «двухэтажный» и другие. Таким образом, номер «односпальный номер эконом с видом на море» должен быть классифицирован как «эконом», «одноместная» и «вид на море» соответственно. Также должна быть реализована мультиязычность и возможность сокращения слов в названии. В случае невозможности определить один из типов возвращать зарезервированный тип «не определено».

Анализ существующих подходов

В общем случае достаточно простую задачу классификации можно решить, используя конечный набор регулярных выражений: шаблонов ключевых слов, по которым можно определить необходимые типы номеров и кроватей. Множество современных информационных систем реализованы на базе трехзвенной архитектуры: клиентское приложение, слой бизнес-логики и слой доступа к данным с самой базой данных или любыми другими источниками.

В случае задачи классификации алгоритм нельзя размещать на стороне клиента, т.к. клиент по определению не должен ничего рассчитывать. Исходя из этого, алгоритм можно разместить в базе данных или на слое бизнес-логики приложения.

В случае с реляционными базами данных, которыми пользуются многие информационные системы, целесообразно разместить алгоритм

внутри хранимой функции или процедуры (в зависимости от СУБД и способов реализации). Это обеспечит наиболее быстрый доступ к данным.

В случае размещения алгоритма на слое бизнес-логики реализация может быть выполнена в любой форме (встраиваться в существующую систему, быть встраиваемым модулем или сторонним сервисом). Также следует изучить возможность применения нейронных сетей для классификации типов номеров и кроватей в отдельном модуле.

Предложенный механизм регулярных выражений уже реализован у одного из поставщиков, название которого не будет упомянуто в этой работе, как и детали реализации. Рассмотрим принцип работы в общих чертах. В ходе исследования будем придерживаться программно-инструментальных средств поставщика для наиболее корректного сравнения производительности алгоритмов. В данном случае будут рассмотрены база данных SQL [3] Server и язык программирования C# [4]. Т.к. решения с нейронными сетями у поставщика нет, оно будет реализовано в демонстрационном режиме самостоятельно с использованием языка Python и библиотеки PyTorch [5].

Регулярные выражения, выполняемые внутри базы данных SQL Server

Рассмотрим подробнее решение поставщика по классификации комнат через регулярные выражения. Сперва надо создать и наполнить необходимые таблицы реляционной базы данных.

Таблица RoomKeyword представляет собой справочник, содержащий регулярные выражения и их идентификаторы.

Таблица RoomKeywordRoomTypesRefs является таблицей пересечений и ссылается на таблицу RoomKeyword и RoomType, а также содержит числовое значение приоритета каждого регулярного выражения (например, в названии номера «Номер люкс бизнес» приоритетнее часть «бизнес», чем «люкс», т.к. люкс – более общее описание номера, а бизнес – более узкое).

Последняя из базовых таблиц – RoomType, содержащая тип номера и его идентификатор.

На основании этих таблиц можно реализовать простейший механизм классификации (и при необходимости дополнительно его оптимизировать созданием индексов, разделением на подзапросы и т.д.). Т.к. механизм предполагает выбор самого приоритетного совпадения, для множественного поиска реализуем циклический проход по названиям комнат, передаваемых как аргумент хранимой процедуры, с использованием курсора. Результаты разберем в следующих разделах.

Регулярные выражения, выполняемые внутри API на C#

Реализация механизма классификации в коде API[6] на C# позволит внедрять классификатор в любую часть существующей бизнес-логики как встроенный модуль или отдельный сервис. Для классификации комнат сперва необходимо загрузить в память названия комнат и все шаблоны.

Для связи C#-программ с базами данных применим библиотеку Dapper, она позволяет писать SQL-команды в сыром виде в коде, выполнять их и приводить ответ к экземплярам классов, что облегчит работу с данными в объектно-ориентированном языке.

Ожидаемо, что загрузка большого количества данных требует времени. Можно предположить, что это время компенсируется фактом того, что с загруженными в оперативную память работа будет проходить быстрее, чем с построчным считыванием из базы в случае работы внутри БД.

Также важно обратить внимание, что комнаты не классифицируются сразу большими блоками: в каждом отеле их обычно меньше сотни, добавление отелей в систему также не происходит большими группами. Для более корректного сравнения проанализируем время классификации большого количества комнат с учетом времени загрузки и без учета времени загрузки по отдельности.

Нейронная сеть, классифицирующая типы комнат

Так как принцип работы нейронной сети не настолько очевиден, как циклический проход по всем названиям комнат и сравнение каждого с шаблоном, целесообразно подробнее описать разработанный подход. Во многом он строится на применении базовых утилит языка Python и библиотек PyTorch [7], Transformers, SKLearn, в том числе предобученная модель rubert-tiny2.

Для обучения модели и синтеза результата сперва необходимо определить классы входных данных. Класс RoomDataset содержит информацию обо всех входных данных: типы комнат, названия, их количества и др.

Класс BalancedDataset имеет схожие атрибуты, но при инициализации разбивает все названия на группы по типу номера и балансирует их так, чтобы редкие типы номеров имели такое же влияние при обучении, как и более частые типы.

Класс RoomTypeElement содержит информацию об одной такой группе и хранит только идентификатор типа комнаты и список названий номеров, которые ему соответствуют.

Далее необходимо загрузить исходные данные в объекты вышеописанных классов. Загружать на обучение сразу все – не лучшая идея (особенно, когда речь идет о миллионах названий номеров, когда необходимо обучить сеть с нуля), поэтому уместно будет разделить все данные на блоки меньшего размера и загружать их постепенно. Этим займется класс DataLoader.

Также необходимо разделить все исходные данные на обучающую и проверяющую (тестовую) выборку. Усредненно-оптимальный процент разбиения – 85 к 15 соответственно. Самая ответственная часть реализации нейронной сети – обучение. Передавая данные небольшими блоками, механизм PyTorch[8] будет подстраивать веса нейросети.

После обучения блока необходимо проверить, насколько лучше нейросеть стала справляться с классификацией номеров. По результатам проверки строятся метрики: потери, точность, F1-меры (микро и макро).

Для предотвращения переобучения (когда модель идеально справляется с классификацией данных из обучающей выборки, но хуже справляется с тестовыми данными) модели реализована ранняя остановка обучения.

После завершения обучения (это достаточно длительный процесс) модель готова к классификации любых входных параметров с доверительной точностью (порядка 99.8%). Полученный модуль можно встроить в любую другую систему как самостоятельный микросервис и использовать изолированно для быстрого получения результатов.

О скорости обучения и работы подробнее в следующем разделе. Важно отметить, что ключевая особенность работы нейросетей – расчеты на видеопамяти, а не на центральном процессоре как в предыдущих методах. Расчеты на видеокартах значительно увеличивают скорость выполнения операций за счет автоматизированного распараллеливания процессов. Дальнейший анализ будет учитывать эту особенность подхода нейронных сетей.

Анализ производительности рассмотренных механизмов

Производительность будем сравнивать по скорости классификации. Все тесты будут проводиться на одном и том же ПК, в проверке будет задействовано переменное число названий номеров, по результатам времени обработки которых построим наглядный график. На текущий момент поставщик имеет количество комнат, кратное миллиону, однако классифицируются они зачастую маленькими группами.

Конфигурация тестового стенда:
CPU: Ryzen 5 2600,
RAM: 32 GB DDR4,
GPU: GeForce 1660.

В таблице 1 приведена сводная информация по результатам экспериментов, а на рисунке 1 представлены графики зависимости времени расчета от количества классифицируемых номеров и подхода. Примечание: анализ производительности нейронной сети проводится с учетом расчетов на видеокарте; все полученные результаты по итогам классификации должны быть сохранены для возможности дальнейшей работы с ними.

Рисунок 1 – График зависимости времени классификации от количества записей для различных подходов

Таблица 1 - Результаты классификации различными методами

	Время обработки 10 тыс. записей	Время обработки 50 тыс. записей	Время обработки 100 тыс. записей	Время обработки 150 тыс. записей
Регулярные выражения БД	17 сек	75 сек	165 сек	236 сек
Регулярные выражения C#	4 сек	33 сек	114 сек	234 сек
Регулярные выражения C# с учетом загрузки данных из БД	5 сек	36 сек	117 сек	245 сек
Нейронная сеть	6 сек	21 сек	41 сек	61 сек
Нейронная сеть с учетом загрузки данных из БД	7 сек	24 сек	42 сек	64 сек

На графике зависимости времени классификации от количества записей и типа алгоритма видим значительное преимущество подхода с нейронной сетью. Подходы с регулярными выражениями показывают сходный результат, однако классификация в C#-коде имеет явно выраженную экспоненциальную зависимость, в то время как обработка в базе данных – линейную.

Выводы

По результатам проделанной работы можно сделать следующие выводы:

- в пределах десяти тысяч записей разница во времени между тремя рассмотренными подходами несущественна;
- свыше десяти тысяч названий наилучший результат с большим отрывом показывает нейронная сеть.

До ста пятидесяти тысяч записей также можно использовать подход с реализацией классификации в C#-коде, однако из-за экспоненциального роста времени расчета классификация сверх ста пятидесяти тысяч в C#-коде не целесообразна.

Подход с применением нейронной сети обрабатывает данные значительно быстрее, однако требует дополнительных ресурсов для внедрения:

- предварительная классификация (в том числе ручная для новых типов комнат);
- обучение модели, место в памяти для хранения модели, дополнительная аппаратура (GPU);
- доработка подхода до состояния готового к внедрению сервиса.

Подходы с регулярными выражениями также требуют предварительной классификации в случае появления новых типов комнат или частных случаев, однако не требует дополнительных затрат на внедрение и оптимизацию. Реализация в коде позволит внедрять алгоритм как самостоятельный настраиваемый сервис, реализация в БД не дает существенных преимуществ.

Все три изученные подхода имеют как свои преимущества, так и недостатки, которые необходимо учитывать при достижении поставленных бизнес-целей.

Литература

1. Мартин, Р. С. Чистый код: создание, анализ и рефакторинг. Библиотека программиста / Р. С. Мартин. — СПб.: Питер, 2013. — 464 с.: ил.

2. Суркова, Н. Е. Методология структурного проектирования информационных систем: Монография / Н. Е. Суркова, А. В. Остроух. — Красноярск: Научно-инновационный центр, 2014. — 190 с.

3. Бейли, Л. Изучаем SQL / Л. Бейли. — СПб.: Питер, 2012. — 592 с.: ил.

4. Шилдт, Г. C# 4.0: полное руководство. — М.: Вильямс, 2011. — 1056 с.

5. Стивенс, Э. PyTorch. Освещая глубокое обучение / Э. Стивенс, Л. Антига, Т. Виман. — СПб.: Питер, 2022. – 576 с.: ил.

6. Общие сведения о минимальных API [Электронный ресурс] / Интернет-Ресурс. - Режим доступа: https://learn.microsoft.com/ru-ru/aspnet/core/fundamentals/minimal-apis?view=aspnetcore6.0&viewFallbackFrom=aspnetcore-2.0

7. Введение в PyTorch [Электронный ресурс] / Интернет-Ресурс. - Режим доступа: https:// pythonist.ru/vvedenie-v-pytorch/?ysclid=lv3j7of2o1668414893

8. Официальная документация PyTorch [Электронный ресурс] / Интернет-Ресурс. - Режим доступа: https://pytorch.org/docs/stable/index.html

9. Джонсон, Д. Учебное пособие по PyTorch: регрессия, пример классификации ихобраэений [Электронный ресурс] / Д. Джонсон. / Интернет-Ресурс. - Режим доступа: https://www.guru99.com/ru/pytorch-tutorial.html

10. Лонца, А. Л. Алгоритмы обучения с подкреплением на Python / А. Л. Лонца / пер. с англ. А. А. Слинкина. – М.: ДМК Пресс, 2020. – 286 с.: ил.

Истягин А.О., Рычка О.В. Сравнение производительности различных подходов в задаче классификации. *В статье рассмотрены различные подходы в решении задачи классификации, реализована нейронная сеть для классификации типов номеров отелей, проведен анализ производительности работы регулярных выражений и нейронных сетей, построены графики зависимости скорости классификации больших объемов данных от выбранного алгоритма, сделаны выводы о целесообразности и уместности применения различных подходов. Сделан вывод, что подход с применением нейронной сети обрабатывает данные значительно быстрее, однако требует дополнительных ресурсов для внедрения.*

Ключевые слова: *программная инженерия, задача классификации, Python, C#, анализ производительности*

Istyagin A., Rychka O. Comparing the performance of different approaches in a classification task*. The article considers various approaches to solving the classification problem, implements a neural network for classifying hotel room types, analyzes the performance of regular expressions and neural networks, plots the dependence of the classification rate of large amounts of data on the chosen algorithm, and concludes on the expediency and appropriateness of using various approaches. It is concluded that the neural network approach processes data much faster, but requires additional resources for implementation.*

Keywords: *software engineering, classification problem, Python, C#, performance analysis*

Статья поступила в редакцию 07.12.2023
Рекомендована к публикации профессором Федяевым О. И.