NLP. Отчёт по ДЗ-1: Text Suggestion

Петухов Андрей

4 октября 2024 г.

Аннотация

В этом отчёте представлены мои результаты разработки системы автодополнения текста с использованием N-граммной модели и пользовательского интерфейса на основе библиотеки `reflex`. Описаны используемые техники, полученные результаты и возможные направления для дальнейшего улучшения интерфейса и модели. Итоговая реализованная система представляет удобный локалхост веб-сайт с приятным пользовательским интерфейсом. Используемая под капотом модель поддерживает вариантов с мгновенным инференсом благодаря эффективной реализации составляющих модели.

Содержание

1 Введение

Домашнее задание предполагало реализацию модели автодополнения слов и предложений для заданных параметров количества предлагаемых вариантов и длины каждого из них в словах или токенах. Само задание реализовано как проект, под все составляющие отведены отдельные модули на python. Пользовательский интерфейс полнотью оформлен в папке autocomoplete, где ключевым является модуль `autocomplete.py`, в котором реализованы и бэкенд и фронтенд и который запускается командой `reflex run` в терминале, поскольку я инициализировал директорию с помощью `reflex` - для запуска у себя нужно будет создать директорию с произвольным названием и перенести в неё содержание моего файла, а также подгрузить сохранённые версии моделей.

Самая большая итоговая модель обучалась на корпусе из 75 тысяч очищенных сообщений, поскольку так удалось достичь хороших дополнений и выучить весь словарь. Пробовались также сетапы из 5k моделей и 15k моделей. Получить доступ к сохранённым версиям моделей можно, написав мне в телеграм @aapetukhov.

2 Предобработка и чистка предложений

Изначальные данные были непригодны для обучения моделей и требовали существенной предобработки, которая представлена в модуле `utils.py`. Предобработка включала в себя удаление всех артефактов, которые поставлялись вместе с текстом сообщения: Message-ID, Date, From, To, Subject, Mime-Version, Content-Type, Content-Transfer-Encoding, FileName и сточки с X (непонятные артефакты - но они и не важны). Помимо этих очевидных для очистки элементов, я также очистил сообщения от ссылок, имейлов, подряд идущих пробелов и знаков препинания.

Помимо этого, я заменил все 're, 've, n't и прочие английские сокращения с апострофами на их соответствующие письменные аналоги (are, have, not и тд). Удалены были излишние пустые сторки, лишние пробелы, а также повторяющиеся знаки препинания, чтобы избежать символов типа — –"или множественные восклицательные знаки. В итоговом тексте были оставлены лишь английский буквы, приведённые в нижний регистр, а также знаки препинания "? "!"и "."Эти знаки были осталены намеренно с целью дать модели умение предсказывать знаки препинания.

Я также экспериментировал с лемматизацией и стеммингом из `nltk`, однако намеренно отказался от этих попыток - они были существенно менее успешными чем последующие реализации.

3 Токенизация

В качетсве токенизаторов я пробовал два разных подхода: базовый вариант с токенизацией по словам `nltk.word_tokenize` и с токенизацией по частям

слов, как предлагалось на лекции - с помощью WordPiece токенизации, для которой я взял `BertWordPieceTokenizer` из библиотеки `tokenizers`.

- `nltk.word_tokenize`: токенизация по словам. Использовалась в базовой конфигурации в первой части - во второй и в пользовательском интерфейсе используется продвинутый способ. Показало, тем не менее, хорошие результаты с логичными предложениями, к тому же быстро применяется к корпусу. Из минусов этого метода можно выделить то, что он слишком прямолинейный и не учитывает структуру слова, из-за чего модель, работающая с токенами из такого токенизатора, будет немного туповатой.

- `BertWordPieceTokenizer`: токенизация по частям слова (WordPiece). Мне показалось очень логичным использовтаь разбиение по частям слова - так модель сможет предсказывать сразу слова с подходящими окончаниями и профичи морфемами. Такой способ более вычислительно затратный, однако на инференс этот никак не влияет - достаточно один раз обучить токенизатор на корпусе и дальше просто применять его - благо в применении он очень быстр. Модель предлагает действительно логичные варианты. Из минусов можно надумать только то, что для него пришлось переписывать основной класс модели, чтобы она могла нормально работать с токенами, что связано исполючительно с особенностями этого токенизатора.

4 N-граммная модель

Ключевой сложностью в данном проекте была быстрая реализация n-граммной модели. Модель лежит в модуле `ngram_lm.py`. Сперва написал её неэффективно, очень долго искал продолжения для префиксов. Однако позже придумал сделать словарь: `prefix -> dict: next_word -> freq`. Так я мгновенно находил префикс из всего корпуса и для него соответствующие n-граммы. Благодаря улучшению в этом модуле инференс получается мгновенный, что видно на видео с демонстрацией, даже несмотря на beam-search в основном модели.

В инференсе она взаимодействует с последними n словами из промта, последнее из которых дополняется word completor-ом, а если запрос по длине меньше чем N токенов, то применяется паддинг символом `<PAD>`. Такая техника тоже советовалась на лекции. Сама модель дополнения слов лежит в модуле `word_completor.py` возвращает сразу отсортированные по вероятностям продолжения и работат молниеносно благодаря предложенному в шаблоне префиксному дереву, поиск по которому выполняется рекурсивно (см модуль `prefix_tree.py`.

5 Итоговые модели

Помимо предложенной базовой модели `TextSuggestion` я написал наследующую от неё `WordPieceTextSuggestion`. Обе модели лежат в модуле `suggester.py` и реализуют полный пайплайн предсказания следующих токенов по заданному входу.

5.1 Кастомная `WordPieceTextSuggestion`

Эта модель реализует сложную логику - предлагает `n_texts` разных вариантов продолжения последовательности благодаря реализации beam-search при поиске вариантов. На вход она умеет принимать обычную строку, которая препроцессится с помощью `preprocess_msg` из модуля `utils.py`, затем препроцешенное сообщение передаётся умному токенизатору по морфемам, для удобства декодирования сразу же передаются и айдишники этих токенов, поскольку декодируются именно они. Далее запускается beam-search по вариантам продолжения по одному слову. Среди вариантов выбираются только `n_texts` самых привлекательных, их привлекательность (`total_proba`) вычисляется либо как сумма, либо как кумулятивное произведение соответствующих вероятностей. Что примечательно, оба сетапа дают разумные результаты, но предпочтительным по логике предсказываемх продолжений мне показался вариант с суммой!

5.2 Базовая `TextSuggestion`

Эта модель реализует базовую логику и применяется только в первой части. Она поддерживает один способ продолжения текста (`n_texts = 1`). Она дополняет с помощью `WordCompletor` последний токен и передает всю последовательность n-граммной модели. n-граммная модель, в свою очередь, предлагает продолжения, из которых берётся самое вероятное.

6 Объединение модулей в `model.py`

Описанные выше модули используются в объединяющей их модели - `SuggestionModel` из модуля `model.py`. Модель представляет из себя удобную обёртку над `WordPieceTextSuggestion` - в неё достаточно передать корпус в виде iterable и основные параметры подмоделей и всё будет работать через метод `predict_sentence`, в который можно передать сообщения просто строчкой, и он его обработает. Варианты предлагаются не в виде списка списков слов, а также в виде обычных строчек. В демонстрации на видео `n_texts=3` и `n_words=3`.

7 Реализация пользовательского интерфейса с помощью reflex

Всё применение рефлекса лежит в `autocomplete.py`, который содержит в себе и фронтэнд, и бэкенд часть. Основные параметры для текстов и боксов я взял из сгенерированного при инициализации шаблона, поскольку они мне визуально понравились. Долго не получалось разобраться с полученной после инициализации файловой структурой, потом завести это веб-приложение в принципе, потом был бесконечный дебаг и приведение этого всего к приличному виду. В итоге я решил по туториалу в ютубе использовать только один ключевой модуль и всё реализовать в нём.

Оцените обязательно маленькие примочки - в окошке для ввода текста сам текст, который пишет пользователь, виден (!) - это у меня на удивление долго тоже не получалось сделать из-за непривычности к рефлексу в принципе. Также вас вперва приветствует заблюренная надпись "Hi stranger... Enter some text которая пропадает как только вы начинаете печатать свой текст в окошко. Я также добавил ссылку на свой гитхаб на веб-страничку. Если вы вводите пустой промт, то выскакивает окошко "Dude, you need to enter some text". Кнопка "Suggest autocomplete"интерактивная - при наведении на неё она выступает вперёд, что сделано с помощью `_hover`. Предложенные варианты также интерактивны благодаря этому параметру.

7.1 Бэкенд

Бэкенд обеспечивает быструю загрузку сайта без необходимости заново обучать все модели на гигантском корпусе текстов. Эта часть подгружает предобученные модели токенизаторв, дополнения слов и n-граммной модели из сохранённых версий, так что их можно использовать сходу - время занимает только сама загрузка моделей. Бэк-часть также реализует часть примочек, в частности если предложений к тексту не находится, то пользователю высвечивается предложение ввести другой текст. Это всё реализовано в начале `autocomplete.py` и классе `State`.

7.2 Фронтенд

Все примочки, цвета, боксы и расположения элементов на экране реализованы в функции `index()`, которая запускается, когда стартует приложение. Все боковые цвета, надписи и переходы, ощущаемые вовремя пользовательского экспириенса, реализованы именно там. Специфичные значения тех параметров обсуждать не буду, весь фронт по сути видно на видео.

8 Заключение

В итоге получилась качественная работа, в которой я реализовал кастомные классы с поддержкой beam-search и приятного пользовательского ин-

терфейса. Всё реализовано удобно в виде проекта, так что юпитер ноутбук максимально чистый.

Из потенциальных улучшений могу предложить обучить модель на полном наборе данных, но мне было страшно сжечь свой процессор, а на Kaggle CPU тоже сильно вспотела от таких объёмов, + Kaggle неудобный, а арендовывать тачку с работы на личном маке я не могу. Помимо этого, можно было бы сделать пользовательский интерфейс таким, чтобы модель автодополнения генерировала подсказки прямо во время печатания. Благодаря скорости работы модели это возможно.