Легковесный детектор для токсичного текстового контента

Михаил Мартинсон

29 сентября 2019 г.

Задача

В этом задании требовалось обучить лекгий и быстрый классификатор для короткий текстов. Датасет состоит из кротких сообщений людей

Условие

Данные Kaggle. Toxic Comment Classification Challenge

Репозиторий решения github.com/MartinsonMichael

Основной файл решения ссылка

1 Моя модель

Эксперименты

Так как в задании есть бонусная часть про интропретируемость модели, вначале я решил использовать в качестве основной модель основанную на деревьях. Немного эксперементов с Random Forest лежат в juputer-notebook Trees.ipynb. В эксперементах видно увеличение точности по мере роста глубины деревьев. Но модель так и не достигнув точности второй, нейронной модели, стала весить критично много (906BM для 1000 деревьев глубины 80).

Следующий и финальный эксепемент был с простой нейроной архетикторуй.

Финальная модель

Модель состоит из byte pair encoder, который переводит строку текста в последовательность токенов. Выбор этого токенизатора обусловлен с одной стороны возможностью держать в словаре целые слова и строить эмюеддинги для них, а с другой стороны способностью токенизировать любое слово.

Далее для каждого токена строется эмбеддинг маленького размера, к которому применяется Dence слой для попышения размерности. Это позволяет не создавать много весов для эмбеддингов, но в тоже время делать вычисляемый эмбеддинг слова не сильно маленьким.

Следующий шаг это два слоя двунаправленного LSTM слоя. Во многих работах по языковым моделям говорится, что нижние слои таких последовательных RNN слоев имеют тенденцию улавливать более низкоуровневые, текстовые фичи, а более высокие - более сложные семантические фичи. Для данной задачи хочется как раз более высокоуровневых фичей, но в тоже время нельзя делать модель слишком громоздкой, поэтому слоев два.

Далее еще один общий Dence слой и GlobalMaxPooling, чтобы привести выходы LSTM, которые могут быть разной длинны для рвзных предложений, к вектору фиксированной размерности. И финально еще два Dence слоя выдающие матрицу $ANS = [7, 2]$ для одного текста. Где $ANS[i, 1], i \in \overline{0, 5}$ вероятность тексту получить тег

`['toxic', 'severe_toxic', 'obscene', 'threat', 'insult', 'identity_hate']`

А $ANS[6, i], i \in \{0, 1\}$ - дополнительно добавленный таргет, являющейся 1, если хотя бы один из набора основных таргетов 1.

Точность модели и Недостатки

Данные очень несбалансированные, $\sim 89\%$ данный вообще не содержат ни одной метки. Поэтому метрика accuracy, которую я считаю, почти ни о чем не говорит, хотя и является довольно высокой.

Лейбл	`toxic`	`severe_toxic`	`obscene`	`threat`	`insult`	`identity_hate`
accuracy	0.96	0.99	0.98	0.99	0.97	0.99
rocauc	0.84	0.74	0.86	0.5	0.77	0.5

Среднее rocauc 0.72

Как можно видеть, rocauc для `threat` и `identity_hate` составляет 0.5, что достигалось бы случайным классификатором. То есть модель так и не научилась выделять эти метки.

Улучшения классификатора

Прямое следствие предыдущего пункта, улчшение классификации `threat` и `identity_hate`. Для этого можно потретировать модель только на сбалансированной подвыборке этих двух классов. Или сделать дополнительные слои в 'головах' предсказывающих распределения этих двух классов.

Для коротких текстов и легких классификаторов кажется важным как можно лучшем образом токенизировать текст. Возможно эксперементы в этой области сильно помогли бы.

Также можно изменить архитектуру классификатора. Например механизм получения эмбеддинга, можно взять уже предобученные.

2 Баланс между качеством и скоростью

Так как в моделях работающих над последовательностями и имеющими внутри архитектуры RNN довольно долгой частью является работа рекурентных блоков, так как их можно вычислять только строго последовательно. Поэтому возможно хорошим решением будет использование архитектур типа трансформера. В ней нет RNN блоков и она может хорошо параллелиться на нескольно процессоров. Но для этого придется полностью изменить архитектуру модели.

Глобально при выборе баланса между качеством и скоростью кажется хорошим решением, понять, что важнее, зафиксировать этот параметр и улучшать оставшийся. Отсюда мне кажется, что при выборе между легкой моделью, или оптимизацией большой, лучше выбрать легкую (конечно с достаточной capacity для решения задачи) и максимально хорошо ее обучить. Ибо при ускорении большой модели придется применить не меньше трюков, чем при обучении маленькой, но нужно еще и обучить большую.