Проект ML

Главное:

Мы решаем задачу NLP по классификации текста на предмет наличия в нём информации о катастрофе. Впоследствие мы используем её и внедрим в телеграм бота.

Аннотация

Мы работаем с датасетом "Natural Language Processing with Disaster Tweets" с Kaggle. Наша задача состоит в том, чтобы проанализировать размеченный список публикаций с Twitter и обучить модель классифицировать, несёт ли публикация информацию о катастрофе. Датасет полностью на английском языке. Состоит из 4 столбцов, несущих информацию и одного столбца, отведённого под номер публикации:

- `id` - уникальный номер для каждой публикации
- `text` - текст публикации
- `location` - откуда опубликовано (может быть пусто)
- `keyword` - ключевое слово (может быть пусто)
- `target` - только в тренировочном датасете, обозначает, несёт ли публикация сообщения о катастрофе (1) или нет (0)

Данная модель потенциально может помочь экстренным службам собирать данные о катастрофах.

Введение

Для решения задачи мы использовали Google Collab, где обучили модель трансформер из huggingface с замороженным backbone

Сначала мы анализировали предложенный датасет, потом очистили его с помощью регулярных выражений, так, например, были удалены символы неанглийского алфавита, обращения к другим пользователям и ссылки (они, скорее всего, не несут полезной информации и могут уменьшить качество модели). Колонка "местоположение" и "номер" были удалены, а колонка "ключевое слово" была

объединена с колонкой "текст", ключевые слова добавлялись в конце текста публикации, если были соответственно.

Далее создали копию тренировочного датасета и первую токенизировали с помощью предобученного токенизатора модели *roberta-large-mnli*, а вторую - с *bert-base-uncased* . Теперь, подготовив набор данных, мы обучили модели и сравнинили их.

Для обучения были выбраны модели **трансформера**, состоящие из нескольких кодировщиков. Они, в отличие от рекуррентных нейронный сетей, обладают механизмом внимания, не забывают контекст с течением времени.

Обозначения, терминология

Kaggle — онлайн-платформа для соревнований по анализу данных и машинному обучению, а также социальная сеть специалистов в этой области.

Датасет - набор данных

Backbone - основная сеть, служащая для извлечения признаков из поступающего на вход изображения. Предобученная модель

Замороженный backbone - Backbone замораживается и не обновялется в процессе обучения

NLP (Natural Language Processing) - Обработка естественного языка

Трансформер (Transformer) - архитектура глубоких нейронных сетей, представленная впервые в 2017 году в статье Attention is all you need

Google Colab - бесплатная облачная платформа для создания и выполнения кода на Python с возможностью использования центральных и графических процессоров

Huggingface -американская компания, разрабатывающая инструменты для создания приложений с использованием машинного обучения

Токенизация - это процесс разделения текста на более мелкие части, такие как слова или предложения. Этот процесс позволяет преобразовать непрерывный текст в дискретные элементы, с которыми можно работать отдельно.

ф1 метрика (f1_score) - в отличии от точности учитывает ещё и дисбаланс классов гармоническое среднее между точностью (Precision) и полнотой (Recall) в машинном обучении

$$F1_{score} = 2 * \frac{(Precision * Recall)}{(Precision + Recall)}$$

Precision - метрика, которая показывает долю объектов, классифицированных как "верные" и при этом действительно являющихся положительными

Recall - метрика, которая показывает, какую долю объектов положительного класса из всех объектов положительного класса нашёл алгоритм

Батч (Batch) - подмножество тренировочных данных, используемое в одной итерации обновления параметров модели

Эпохи - полная итерация по набору данных в процессе обучения модели

nn.Model - огромный класс, используемый для создания большинства нейросетей

Обзор литературы

Данные статьи помогут читателю разобраться в архитектуре трансформеров (нажмите на выделенное для ознакомления)

Архитектура BERT - статья для понимания архитектуры Bert

RoBERTa - статья на английском, в ней рассказывается об отличиях от модели BERT

Я же советую читателю посмотреть курс лекций от DLS, в которых подробно объясняется принцип работы трансформера: **Вот их YouTube канал**

Работа

В ходе работы было проведено несколько экспериментов, чтобы добиться максимального качества у модели. Мы сравниваем именно эти две модели, потому что они довольно похожи, но сильно отличаются своими размерами (RoBERTa больше).

На обучение модели выделялось 90% датасета, отведённого на обучение. На валидацию - 10 %.

1 эксперимент

Самым удачным оказалось обучение модели *bert-base-uncased* с замороженным backbone

Она обучалась 15 эпох 49 минут и показала хороший результат в 82 % точности и 81 % ф1 метрики. Именно результаты работы этой модели я использовал на Kaggle, где получил $F1_{score} = 0.8112$ (81 %) и занял в таблице лидеров 339 место.

[375/375 49:31, Epoch 15/15]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.545600	0.808743	0.784370	0.800995	0.784370	0.771676
2	1.119500	1.223617	0.730825	0.790264	0.730825	0.693374
3	0.617000	0.772295	0.696093	0.758237	0.696093	0.695451
4	0.949600	1.159930	0.703329	0.774946	0.703329	0.652064
5	0.528400	0.499053	0.817656	0.816602	0.817656	0.816095
6	0.564600	0.504178	0.811867	0.810901	0.811867	0.809770
7	0.969100	1.426244	0.568741	0.751769	0.568741	0.530959
8	1.232100	0.608037	0.768452	0.783738	0.768452	0.770703
9	0.706100	0.508716	0.807525	0.809095	0.807525	0.808103
10	0.726400	0.452463	0.810420	0.811646	0.810420	0.810893
11	0.504000	0.516380	0.765557	0.783283	0.765557	0.767877
12	0.500700	0.457056	0.811867	0.812102	0.811867	0.811977
13	0.449800	0.450179	0.810420	0.812323	0.810420	0.811081
14	0.533700	0.428337	0.819103	0.818540	0.819103	0.816840
15	0.463500	0.431114	0.820550	0.823601	0.820550	0.816144

2 эксперимент

Далее я разморозил backbone и попытался обучить модель с теми же параметрами.

Но модель стала занимать слишком много памяти графического процессора. Для её обучения пришлось уменьшать размер батча с 256 до 8. Качество модели значительно упало, а время её обучения сильно увеличилось. В итоге данная модель не вошла в итоговую работу, но заслуживает упоминания.

3 эксперимент

Но почему я выбрал именно *bert-base-uncased*? Потому что это относительно простая модель, время обучения которой должно быть небольшим. Первоначально планировалось обучить только модель *roberta-large-mnli* с замороженным и размороженным backbone. Roberta превосходит стандартный bert как минимум по тому, что была обучена на количестве данных в 10 раз больших, чем bert. С неё и начались эксперименты. Обучение Roberta с размороженным backbone приведено в первоначальной версии прикреплённого ноутбука. Эта модель обучалась очень долго. Я пробовал её обучить успешно два раза. На 10 эпох и на 20 эпох.

10 эпох:

[250/250 1:47:40, Epoch 10/10]

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	9.932000	11.460972	0.599132	0.358959	0.599132	0.448942
2	13.091500	10.624026	0.599132	0.358959	0.599132	0.448942
3	1.567700	5.492534	0.400868	0.460146	0.400868	0.231948
4	1.406600	2.125702	0.622287	0.768336	0.622287	0.499319
5	5.466400	1.190585	0.573082	0.671804	0.573082	0.557895
6	0.778000	1.030077	0.662808	0.674330	0.662808	0.665784
7	1.846500	1.357109	0.494935	0.620542	0.494935	0.450190
8	4.180400	3.959067	0.597685	0.358611	0.597685	0.448263
9	0.740800	0.653962	0.712012	0.757586	0.712012	0.672369
10	0.782100	0.516926	0.768452	0.766931	0.768452	0.764061

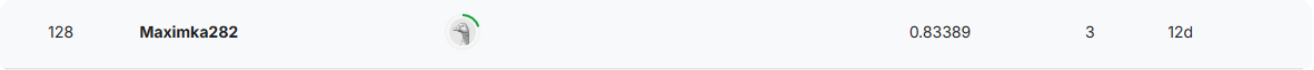
20 эпох:

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	10.305300	11.951322	0.599132	0.358959	0.599132	0.448942
2	12.592400	9.992604	0.597685	0.358611	0.597685	0.448263
3	1.116600	4.659231	0.599132	0.358959	0.599132	0.448942
4	3.453400	4.669896	0.599132	0.358959	0.599132	0.448942
5	4.580100	2.594554	0.567294	0.534905	0.567294	0.532632
6	1.395000	1.193070	0.638205	0.774423	0.638205	0.531393
7	1.090600	1.218599	0.622287	0.683664	0.622287	0.511768
8	0.947700	0.778547	0.645441	0.687812	0.645441	0.647033
9	5.324200	2.181677	0.461650	0.634588	0.461650	0.377971
10	4.149100	1.446784	0.617945	0.691145	0.617945	0.498954
11	3.923300	4.556591	0.599132	0.358959	0.599132	0.448942
12	2.251700	1.343438	0.636758	0.747581	0.636758	0.532364
13	0.718900	0.570683	0.764110	0.762805	0.764110	0.758995
14	3.109300	1.528565	0.664255	0.725856	0.664255	0.594413
15	0.873000	0.760654	0.623734	0.696693	0.623734	0.619303
16	0.930400	0.663370	0.735166	0.770106	0.735166	0.706645
17	0.885000	0.565336	0.690304	0.728593	0.690304	0.692339
18	0.784900	0.619988	0.707670	0.764794	0.707670	0.662498
19	0.734900	0.498731	0.756874	0.754939	0.756874	0.755378
20	0.848300	0.528310	0.752533	0.754402	0.752533	0.753276

Сложности также были вызваны с нестабильным интернет подключением, из-за которого Google Collab постоянно обновлял локальную среду, стирая все наработки

Финальный вариант

Спустя 3 месяца после первой работающей версии были сделаны ключевые изменения. Нужно было подготовить модель к экспорту, чтобы вставить её в телеграм бота. Я немного изменил код, а именно класс трансформера стал наследоваться от другого. Первоначально он наследовался от огромного класса nn.Model, но сохранить его было тяжело, потому-что в классе nn.Model нет метода сохранения (мне удалось сохранить в файл zip, но файл .json не нёс необходимых конфигураций модели). Поэтому класс трансформера стал наследоваться от PreTrainedModel, с помощью которого я смог всё удачно сохранить. Также я поменял размер батча, уменьшив его с 256 до 32 и понизил learning rate, чтобы модель лучше сходилась. Она стала требовать меньше памяти и хорошо обучилась. Результат повысился. Так, на платформе kaggle решение смогло набрать такой результат:



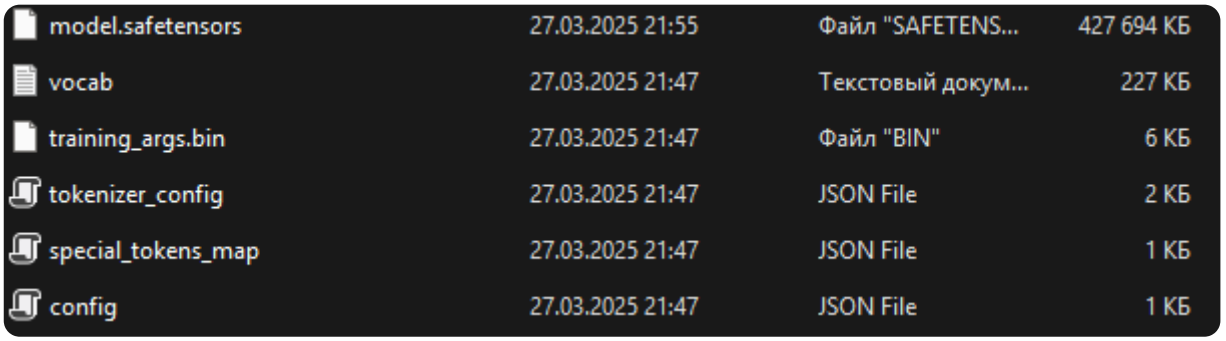
Вот, какие метрики получались на валидационном датасете:

Epoch	Training Loss	Validation Loss	Accuracy	Precision	Recall	F1
1	0.490300	0.405906	0.832370	0.832557	0.832370	0.832457
2	0.535700	0.374934	0.842486	0.843239	0.842486	0.840580
3	0.135300	0.464807	0.832370	0.833824	0.832370	0.832844
4	0.029700	0.528716	0.838150	0.837900	0.838150	0.836787
5	0.038400	0.678318	0.786127	0.791591	0.786127	0.787382
6	0.065100	0.812002	0.822254	0.823143	0.822254	0.819628
7	0.026300	0.889621	0.813584	0.812735	0.813584	0.812226
8	0.000800	0.999794	0.815029	0.814139	0.815029	0.814014
9	0.000700	1.076225	0.813584	0.812706	0.813584	0.812751
10	0.000500	1.200043	0.812139	0.811269	0.812139	0.811362
11	0.000900	1.194839	0.823699	0.822943	0.823699	0.822608
12	0.000200	1.279109	0.823699	0.824234	0.823699	0.821335
13	0.000200	1.285382	0.822254	0.821738	0.822254	0.820688
14	0.000500	1.307117	0.822254	0.821634	0.822254	0.820826
15	0.000200	1.317642	0.822254	0.821738	0.822254	0.820688

Лучшего результата она смогла достичь на первых эпохах, а потом начала переобучаться (это видно по столбцу "Training loss", который стал уже с 8 эпохи стремиться к нулю очень быстро)

Telegram bot

Сохраним модель со всеми конфигурациями:



model.safetensors	27.03.2025 21:55	Файл "SAFETENS...	427 694 КБ
vocab	27.03.2025 21:47	Текстовый докум...	227 КБ
training_args.bin	27.03.2025 21:47	Файл "BIN"	6 КБ
tokenizer_config	27.03.2025 21:47	JSON File	2 КБ
special_tokens_map	27.03.2025 21:47	JSON File	1 КБ
config	27.03.2025 21:47	JSON File	1 КБ

В файле model.safetensors содержатся веса модели, в config архитектура:


```
"architectures": [
  "BertForSequenceClassification"
],
"attention_probs_dropout_prob": 0.1,
"classifier_dropout": null,
"gradient_checkpointing": false,
"hidden_act": "gelu",
"hidden_dropout_prob": 0.1,
"hidden_size": 768,
"initializer_range": 0.02,
"intermediate_size": 3072,
"layer_norm_eps": 1e-12,
"max_position_embeddings": 512,
"model_type": "bert",
"num_attention_heads": 12,
"num_hidden_layers": 12,
"pad_token_id": 0,
"position_embedding_type": "absolute",
"problem_type": "single_label_classification",
"torch_dtype": "float32",
"transformers_version": "4.50.0",
"type_vocab_size": 2,
"use_cache": true,
"vocab_size": 30522
```


Телеграм бот нужен для демонстрации возможностей модели. Через тг бота @BotFather мы получаем уникальный токен.

Представленный бот написан без каких-либо крутых фишек. Запускается он локально. Чтобы он работал, нужно в терминале установить библиотеки, которые были в Google colab и библиотеку telegram для работы с API телеграма. Обязательно используем функцию, использованную в модели для очистки твитов.

Добавлена функция логгирования. Можно собирать данные о запросах, переданных боту и о пользователях, которые их отправляли.

Бот написан асинхронно для обработки несколько запросов параллельно, но у него всего одна команда, которая расскажет пользователю о том, что нужно делать.

Возможные усовершенствования

Можно модифицировать телеграм бота и добавить ему несколько команд, которые облегчат пользователю работу с большими объемами данных:

1. Добавить возможность разделить текст на предложения, проверить каждое из них и вывести, какие предложения могут нести информацию о катастрофе
2. Добавить возможность проверять много предложений за один запрос к боту (например файл подать внутрь)

Выводы

Мы провели серию экспериментов, в ходе которых обучали разные модели и сравнивали их точность. Самой точной моделью оказалась *bert-base-uncased* с замороженным backbone, которая в сравнении с другими рассмотренным показала наименьшее время работы и лучшее качество с первых эпох. *Roberta-large-mnli* по нашим предположениям должна была показать большее качество в связи с тем, что обучена она лучше, но на практике она не смогла превзойти *bert-base-uncased*. Изменив класс наследования, мы смогли сохранить модель и внедрить её в тг бота, что служит отличной демонстрацией возможностей модели, потому что теперь любой пользователь может с ними ознакомиться.

Мы сделали важный вывод, что заморозка backbone эффективна для малых датасетов

Заключение

С помощью модели *bert-base-uncased* мы смогли добиться хорошего качества в 83 % Мы убедились в том, что не всегда модели, обученные лучше показывают качество лучше, а также смогли продемонстрировать, как работает модель с помощью тг бота. Данная модель может помочь экстренным службам для защиты, к примеру, детей от разрушительного контента.