Data Mining Lab2 Report

112062595 林育丞

Overview

Preprocess steps

Model backbone

The other attempt (my focus)

Preprocess steps

- Convert JSON Entries to CSV rows.

The initial data is stored in a JSON list, both the testing and the training data are clumped together; while having extra files for their identification and emotions, this isn't ideal for later steps. So, I had performed some clean up and saved the result so that I don't need to ever worry about it again.

pairing ID and emotions (processed_data / pair_json_with_emotion.py)

JSON to csv (processed_data / emotion_json_to_csv.py)

```
tweet_id,text,hashtags
0x28b412,"Confident of your obedience, I write to you, knowing that you will do even mo
0x2de201,"""Trust is not the same as faith. A friend is someone you trust. Putting fait
0x218443,When do you have enough ? When are you satisfied ? Is you goal really all abou
0x2939d5,"God woke you up, now chase the day #GodsPlan #GodsWork <LH>",GodsPlan|GodsWor
0x26289a,"In these tough times, who do YOU turn to as your symbol of hope? <LH>",
0x31c6e0,Turns out you can recognise people by their undies. <LH>,
0x32edee,"I like how Hayvens mommy, daddy, and the keyboard warriors have to jump into
0x3714ee,I just love it when every single one of my songs just delete themselves..😡😒
```

▲ Keep only the important features and make them more compact.

In this process, I also **removed lots of unnecessary features**, such as _score, _index, _crawldate, _type (maybe the data crawled from Elastic Search?)

- Convert emojis to descriptive captions.

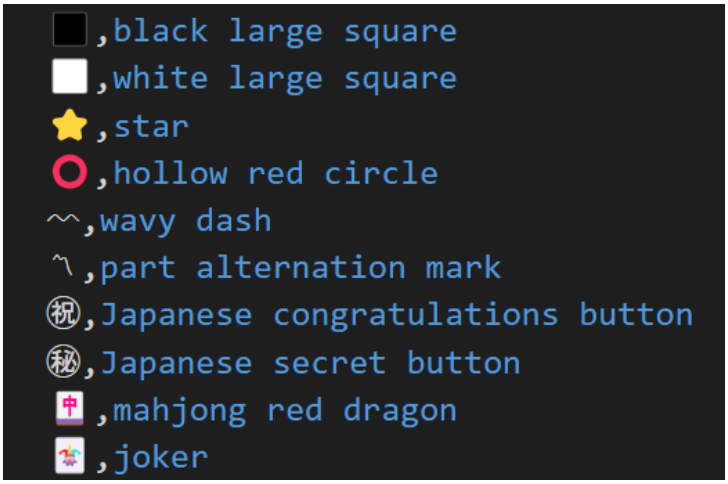
By utilizing spacyemoji package and regular expression, we can extract the emojis and pair them with the corresponding captions.

Extract unique emojis from the data (emoji_test / extract_emoji.py)

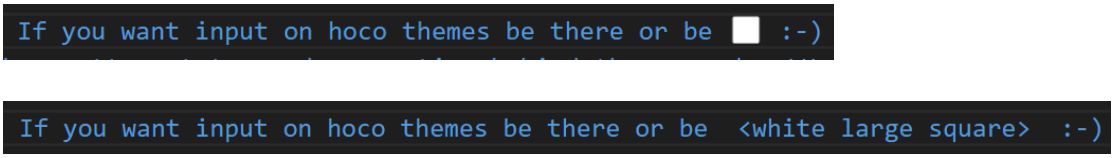
Pair emojis with description (emoji_test / spacy_test.py)

(Replacing the emojis in the csv files is trivial, I didn't keep the file for that.)

After these steps, the data is now in csv form and with emoji descriptions, which should increase the compatibility of tokenizers that don't recognize emojis.



▲ Intermediate file that contains the emoji-caption pair.



▲ Converted test data example.

Model backbone

For the model, I choose to use the BERT as the tokenizer and use the BertForSequenceClassification as the model backbone. I had some experience with it before the class, so it is not difficult for me to setup the environment. I didn't bother changing the hyper parameters and left everything as it is.

Note that the code was adapted from two publicly available notebooks, with some custom modifications (dm-lab2.ipynb). If those notebooks didn't exist, I will still use BERT or other transformers as the backbone, they just saved me a little bit of time.

The other attempt

- Motivation

I attempted to use GPT-4o-mini (from OpenAI) as backbone, by creating an automatic program that can regularly send patches of data to the API endpoints, then parse the return result.

The idea is that it should be incredibly easy for the GPT models to analyze the sentiment behind the tweets, and from the fact that there's no runtime limit in this Kaggle competition, **the most powerful emotion analyzing and reasoning engine we can possibly use is the GPT-4 series**.

- The feasibility (price, time, accuracy)

The API calls require money and have rate limits. By analyzing the tokens count per request (one of the functions in token-test.py), it turns out that **GPT-4o-mini can process all 400,000 test data, with under $10 USD, under 3 hours.**

Model	Pricing
gpt-4o-mini	$0.150 / 1M input tokens
	$0.075 / 1M cached** input tokens
	$0.600 / 1M output tokens

▲ Pricing for the GPT-4o-mini from the official website

Each message contains system input and user input, the former is used to tell the model how to interact with the input, and the latter is the actual input. The token count for a **(system + 100 tweets batch) is 4000**. Since the output price is quadrupled, but the output length is shorter (only tweet_id and emotion), we can assume it has the same price as the input.

400,000 data = 400 batches * (100 per batch) = 400 * (4000 tokens) = 16M tokens

16M tokens should be about $2.4 USD, so the **price won't be an issue**.

MODEL	RPM	RPD	TPM
gpt-4o	500	-	30,000
gpt-4o-mini	500	10,000	200,000

▲ Rate limits from the official website

OpenAI has different rate limits criteria and will stop responding when any of the limits is reached. The 200,000 token per minute is the most restrictive one for my case, which means **we can only send 50 batches per minute (maximum).**

16M token will require more than 80 minutes to process, which is very acceptable, even if my calculation is way off and somehow it needs to be doubled (160 minutes), it is still faster than the BERT training for 3 epochs (360 minutes). So, **the time won't be an issue**.

Accuracy wise, I had tested it with the ground truth data, surprisingly, the result is very bad. The correct percentage is only about 40%. **Accuracy is the issue**.

```
For example, the input will be :
tweet_id,text
0x376b20,"People who post ""add me on #Snapchat"" must be dehydrated. Cuz man.... t
0x2d5350,"@brianklaas As we see, Trump is dangerous to #freepress around the world.
0x1cd5b0,Now ISSA is stalking Tasha 😂😂😂 <LH>
0x1d755c,@RISKshow @TheKevinAllison Thx for the BEST TIME tonight. What stories! He
0x2c91a8,Still waiting on those supplies Liscus. <LH>
```

▲ Part of the system message for the GPT model.

- Improve performance

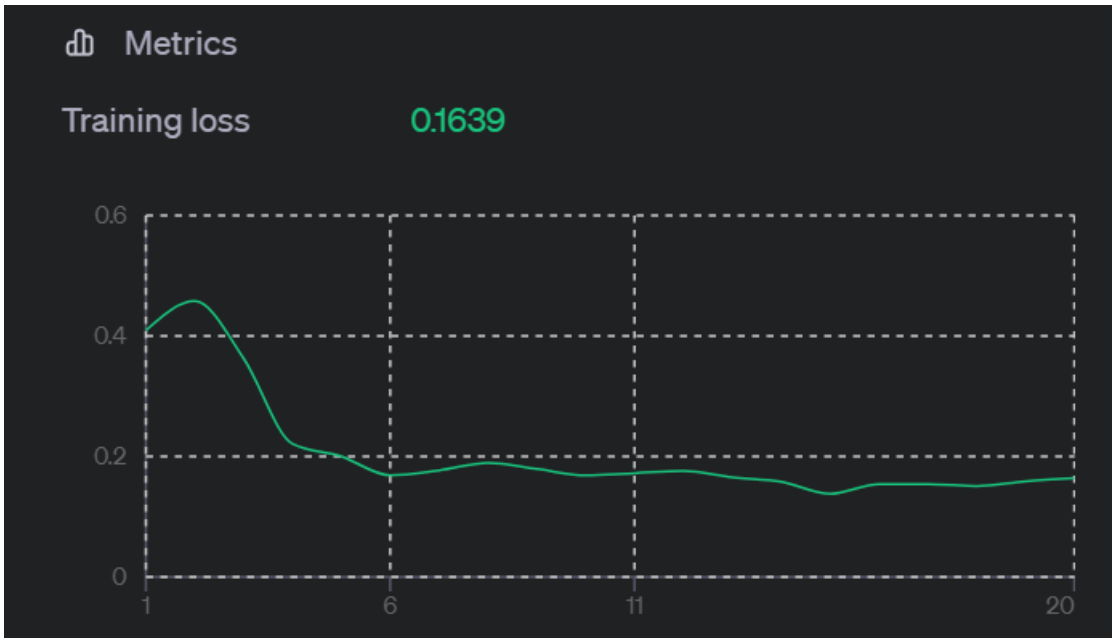
At first, I've tried to

1. Increase the number of samples in the system message
2. Clearer prompt (a little bit of prompt engineering)
3. Use the data with emojis
4. Reduce the number of tweets per batch (from 100 to 50 to 10)

They didn't impact much on the performance, except for the batch size, although it is still very limited in improvement.

Then I **tried to use the full model, GPT-4o**, which cost more than 15x of the mini one. But it performed even worse, by uncontrollably output emotions that aren't in our defined category. About 0.2 Accuracy.

Finally, I **tried to fine-tune the model** (create-fine-tuning.py, verify-jsonl.py). I had uploaded the fine-tune examples to OpenAI, and the model finished fine-tuning in about 10 minutes. By using the fine-tuned model, the outcome still isn't acceptable, with roughly the same performance as the un-modified model.



▲ Fine-tune progress, from the OpenAI project dashboard.

- Final thought

After some inspecting, **I've realized it is the quality of the data that makes GPT models have a hard time**. I 've copied some of the train data and asked ChatGPT to analyze if the emotion label is reasonable. **Turns out some of the labels are very far off from how human experience**.

For example,
0x23b037 "I love suffering 🙃 🙃 I love when valium does nothing to help 🙃 🙃 I love when my doctors say that they've done all they can 🙃 🙃 " is labeled as joy.

I tried to analyze the output from the GPT models and turns out that **some of the predictions it makes, actually make more sense than the ground truth**.

For example,
0x1fa303, "no wait okay so fuck people who think tattoos are bad but for some reason the idea of a tattoo that says "#regret" on it just tickles me",

The label in the data is sadness, while the GPT prediction is joy.

My humble guess is that the emotion labels came from a not-so-convincing classification model, so that the models with advanced reasoning will have a hard time delivering "accurate" predictions.

The fact that this method didn't work is discouraging, as if we were asked to train the model, just for the not-so-accurate dataset.