NLP Twitter Analysis

Hamed Feizabadi

July 12, 2023

Contents

1 Introduction

This is the final report for the NLP course project. In this project we have collected tweets from Twitter and labeled them using ChatGPT model. Then we have augmented the data using GPT-3.5-turbo model. After that we have trained a classifier using the augmented data and evaluated the results.

Word2vec, tokenizer and language model and other stuffs also have been trained on the collected data, which we will discuss in the following sections.

2 Repository

You can access the source code of this project at https://github.com/hamedhf/nlp_twitter_analysis

3 HuggingFace Dataset

The dataset is available at https://huggingface.co/datasets/hamedhf/nlp_twitter_analysis/tree/main

4 Requirements

If you have a GPU(especially Nvidia) then you are good to go and can run project locally, as we did. Otherwise you can use Google Colab to run the project. For that purpose you need to clone the github repo inside colab and run the commands provided with main.py file. These commands are also provided in the README.md file of the repository.

5 Installation

You should create a file named "users.csv" inside src folder which contains the Twitter username, University name an Actual name of the users you wish to analyze.

Furthermore installation instructions are provided in the README.md file of the repository.

6 Project Structure

```
rootfolder
├── data
│   ├── clean
│   ├── ...
│   ├── augment
│   ├── split.............................parsbert finetune data
│   └── language_model....................gpt2-fa finetune data
├── logs..................................all project logs
├── models
│   ├── gpt2..............................gpt2 farsi model
│   ├── parsbert..........................parsbert model
│   ├── word2vec..........................word2vec model
│   └── spm_tokenizer.....................sentencepiece tokenizer
│                                         model
└── src
    ├── latex.............................latex source files
    ├── main.py...........................main script for running
    │                                     commands
    └── utils
        ├── augment.py....................augment data using
        │                                 gpt-3.5-turbo
        ├── clean.py......................clean data
        ├── crawl.py......................crawl data from twitter
        ├── label.py......................label data using
        │                                 chatgpt
        ├── split.py......................split data into train,
        │                                 test and validation
        ├── constants.py..................project constants and
        │                                 configurations
        ├── word2vec.py...................train word2vec model
        ├── sentencepiece.py..............training tokenizer
        ├── gpt2.py.......................train gpt2 model
        ├── parsbert.py...................train parsbert model
        └── ...
```

Figure 1: Project Tree

7 Data Collection

We used selenium >= 4.6.0 for collecting data from Twitter. This tool helps us to bring up an actual browser and navigate through the pages. Note that you should have Chrome installed on your system for this to work. Then you can simply install other dependencies form pyproject.toml file using poetry or other package managers. The crawler script reads the users.csv file and for each user, it navigates to the user's profile and collects the tweets. The tweets are stored in a file named unlabeled.db inside data/raw folder. Then labeling script uses this and with the help of ChatGPT model, it generates the labels for each tweet and stores them in data/raw/labeled-run-date.csv file.

8 Data Format

The data is stored in a csv file with the following format:
tweet_time, tweet_owner, tweet_text, owner_university, owner_name, label. We use tweet_time, tweet_owner as unique identifiers for each tweet. The tweet_owner is Twitter username of the owner. The tweet_text is the actual text of the tweet. owner_university and owner_name are the university and actual name of the tweet owner. The label is the generated label for the tweet.

9 Data Preprocessing

We have splitted data with three criteria: split by sentence with hazm sentence tokenizer, split by word with hazm word tokenizer, split by word with hazm lemmatizer.

For cleaning the data, we used the following steps: remove emojis, remove urls, remove hashtags, remove mentions, remove numbers, remove punctuations. We used the hazm, cleantext and nltk libraries for this purpose.

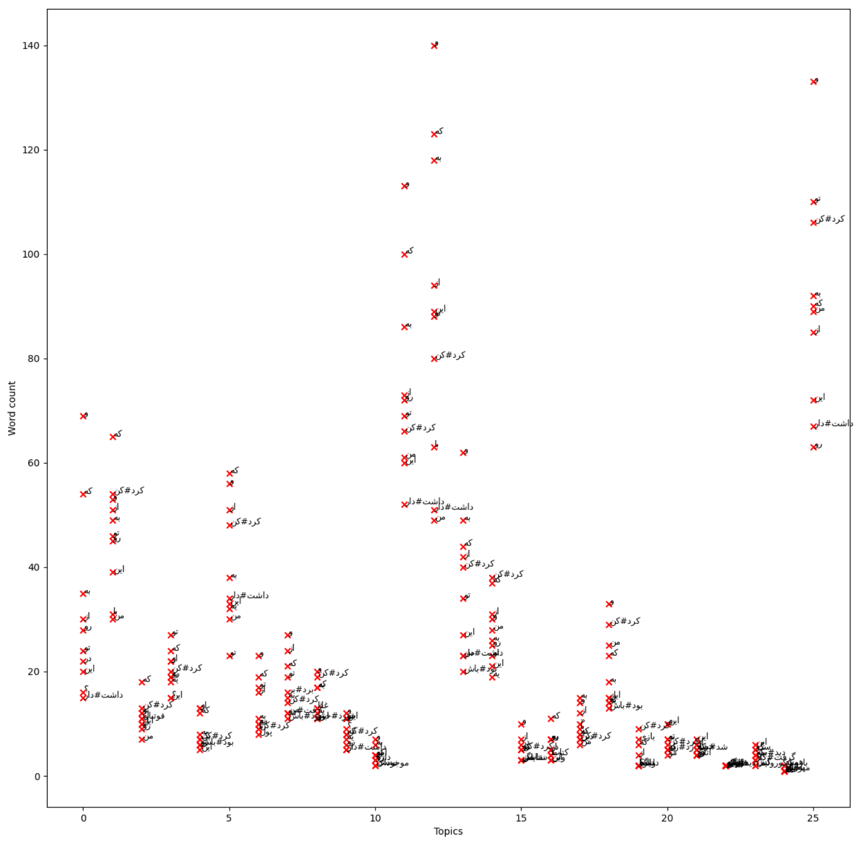
10 Labeling

We give label to the whole tweet using ChatGPT. For more info about labeling see the src/utils/label.py file. You can also see the labels in src/utils/

constants.py file.

11 Statistics

tweet-count	word-count	sentence-count	unique-word-count
2079	31987	2944	6725



12 Augmenting Data

For this part to work you need to sign up for an account at https://platform.openai.com and provide your openai api key in .env file. Then you can run the augment-data command.

The augmentation script takes a cleaned csv file as input and counts how many tweet we have for each label. Then it will fix the imbalance of the data by generating new tweets for the labels with less tweets. The generated tweets are stored in data/augment folder.

You can see the implementation detail of the augmentation script in src/utils/augment.py file. We have used gpt-3.5-turbo model for this purpose and the given prompt is like this:

```python
import openai
label = "home_and_garden"
temperature = 0.6
system_message = "Generate an informal Persian tweet about the given topic without any hashtags, mentions, links, or emojis."  # noqa
messages = [
    {
        "role": "system",
        "content": system_message
    },
    {"role": "user", "content": f"topic: {label}"}
]
response = openai.ChatCompletion.create(
    model="gpt-3.5-turbo",
    messages=messages,
    temperature=temperature,
    timeout=120
)

```

Temperature is a parameter that controls the randomness of the generated text. The higher the temperature, the more random the text. The lower the temperature, the more predictable the text. We have used 0.6 for this parameter and it is random enough for our purpose and if we increase it will take much more time to generate the text and this is not practical for our purpose.

Using this approach we have doubled our total data size and each label has at least 200 tweets. It is worth mentioning that because of openai api rate limit, it took us about **2 and a half days** to generate the data.

12.1 Generated Tweets

Some of the generated tweets:

هنر و طراحی چیزیه که زندگی رو جذاب و زیبا می‌کنه با نگاه به اثرهای هنرمندان و طراحان خودمون رو در دنیایی خلاقان…	openai	openai	art_and_design
دین و معنویت یه چیزه که هر کسی بهش نیاز داره حتی اگه نمیخواد به هیچ دینی تعلق داشته باشه همه ما به دنبال یه چی…	openai	openai	religion_and_spirituality
جهان ما تنها محلی است که برای زندگی ما وجود دارد بیایید برای حفظ محیطزیست و پایداری آن تلاش کنیم بچه‌ها را آموز…	openai	openai	environment_and_sustainabili
ورزش و ورزشکاری یکی از بهترین راه‌ها برای سلامتی و خوشحالیه همیشه باید به خودمون وقت بدیم و ورزش کنیم برید بیرو…	openai	openai	sports_and_athletics
کیا می‌دونین گربه‌ها می‌تونن بهترین دوستای انسان باشن؟ منم یه گربه دارم که همیشه باهامه و خیلی دلم براش تنگ میشه …	openai	openai	pets_and_animals
دین و معنویت دوستانه میگن که برای روحیه و آرامش خودمون به این دوچیز نیاز داریم هرکسی طبق اعتقادات و معتقدات خو…	openai	openai	religion_and_spirituality
اونقدر از کرمای صورت استفاده می‌کنم که توی صبح کارمون رو شروع نمی‌کنیم تا من همه‌ی مراحل آرایشمو تموم کنم	openai	openai	beauty_and_cosmetics
ورزش و ورزشکاری یکی از بهترین راه‌ها برای سلامتی و شادابی بدنه همیشه باید فعالیت و ورزش رو تو روزمره‌ات جا داد و…	openai	openai	sports_and_athletics

12.2 Augmented Data Statistics

	label	tweet count
1	politics_and_current_affairs	200
2	entertainment_and_pop_culture	200
3	sports_and_athletics	200
4	technology_and_innovation	200
5	science_and_discovery	200
6	health_and_wellness	200
7	business_and_finance	200
8	travel_and_adventure	200
9	food_and_cooking	200
10	fashion_and_style	200
11	environment_and_sustainability	200
12	education_and_learning	216
13	social_issues_and_activism	217
14	inspirational_and_motivational	200
15	funny_and_humorous	200
16	art_and_design	200
17	books_and_literature	200
18	religion_and_spirituality	200
19	family_and_parenting	200
20	gaming	200
21	beauty_and_cosmetics	200
22	home_and_garden	200
23	automotive	200
24	pets_and_animals	200
25	weather_and_seasons	200
26	other	413

13 Word2Vec

We used gensim library for training skipgram word2vec model because it is easy to use and fast. The implementation is in src/utils/word2vec.py file.

All ot the available commands are listed in **ReadME.md** file. Here we explain some of them.

13.1 Training

This command trains word2vec for a specific label.

```
 python src/main.py train-word2vec-label path-to-augmented-csv home_and_garden

```

This command trains word2vec for some preselected labels.

```
 python src/main.py train-word2vec-preselected path-to-augmented-csv

```

This command trains word2vec for all labels.

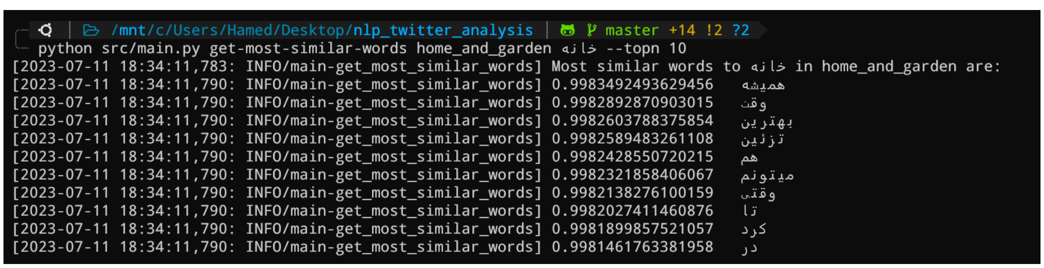
```
 python src/main.py train-word2vec-all path-to-augmented-csv

```

Each model is saved in a models/word2vec/label.npy file.

13.2 Evaluation

Let's find that in topic of home_and_ garden, which words are similar to the Persian word for home (khaneh).



Some of the similarity results are shown in the following image. We use cosine similarity for measuring the similarity between two words. The higher the similarity, the more similar the words are.

کلمات مشابه برای کلمه ایران:

[('تعداد', 0.9875343441963196),
('خوابگاه', 0.9871758818626404),
('حکم', 0.9862680435180664),
('ددلاین', 0.9860983490943909),
('اونور', 0.9860574007034302),
('اولش', 0.9858595728874207),
('انقلاب', 0.9857739210128784),
('تنها\u200cتری', 0.9856471419334412),
('بی\u200cگناه', 0.9855296611785889),
('تومن', 0.9853056073188782)]

کلمات مشابه برای کلمه سیاست:

[('جاری', 0.984733521938324),
('امور', 0.9736297726631165),
('پیچیده', 0.9675320982933044),
('روزها', 0.9606905579566956),
('سیاسی', 0.9490455389022827),
('بازار', 0.9372583627700806),
('اخبار', 0.9359972476959229),
('شده', 0.9271780252456665),
('تجارت', 0.925977885723114),
('گرون', 0.9195088744163513)]

کلمات مشابه برای کلمه ماشین:

[('بخرم', 0.9738640189170837),
('روزی', 0.9613807797431946),
('یکیشونو', 0.9528212547302246),
('قدرتمند', 0.948646605014801),
('بتونم', 0.9442844390869141),
('خفن', 0.9425691366195679),
('خیابونا', 0.9352948069572449),
('میخوام', 0.9346776604652405),
('بشم', 0.9335148334503174),
('خرید', 0.9271440505981445)]

کلمات مشابه برای کلمه گل:

[('باغچه\u200cام', 0.9820027351379395),
('گل\u200cها', 0.9808597564697266),
('دکوراسیون', 0.9766149520874023),
('آروم', 0.970939040184021),
('می\u200cگذرونم', 0.9690070748329163),
('گیاهانم', 0.9681393504142761),
('باغچه\u200cی', 0.9678846001625061),
('باغ', 0.9676481485366821),
('گیاه', 0.9649395942687988),
('تمیز', 0.9636048674583435)]

14 Tokenizer with byte pair encoding

With the help of sentencepiece library we trained a tokenizer with byte pair encoding. The implementation is in src/utils/sentencepiece.py file. We used the augmented data for training the tokenizer. The tokenizer is saved in models/spm_tokenizer/tokenizer(vocab_size).model file.

We have trained the tokenizer with different vocab_size and tested them with this metric that how many of the words in the test set will be mapped to UNK token. The results are shown in the following table.

vocab_size	UNK count
100	316
1000	57

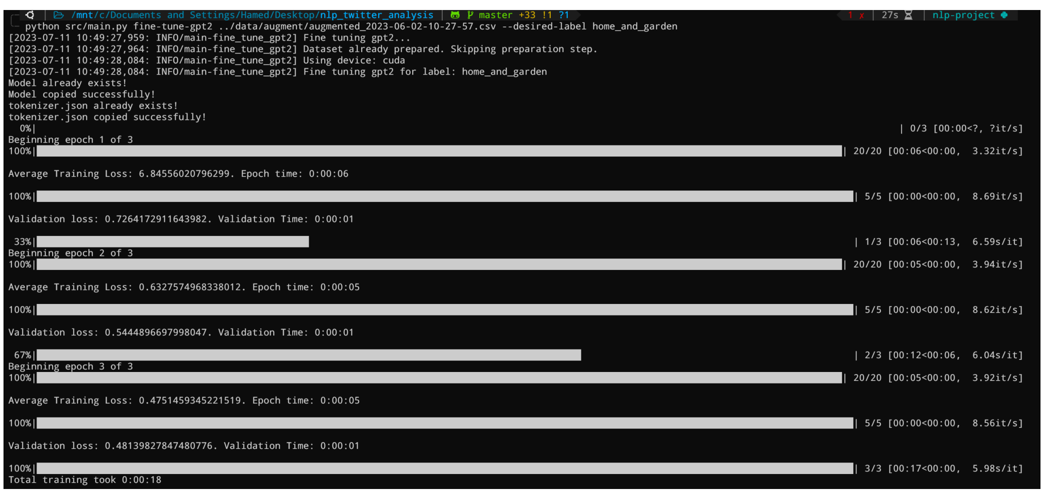
As you can see, larger vocab_size leads to less UNK count on the test set.

15 Language Model

We used huggingface transformers library for training the language model. The implementation is in src/utils/gpt2.py file. We used HooshvareLab/gpt2-fa model for this purpose. The model is trained on huge Persian corpus and it is available at https://huggingface.co/HooshvareLab/gpt2-fa. Use of pretained model helps us gain better results with less data.

15.1 Fine Tuning

The below command shows how to fine tune the model for a specific label. The script will check for dataset in data/languagemodel and if it is not available, it will create it using the augmented data. Then it will fine tune the model for the given label and save the model in models/gpt2/label folder.



15.1.1 Training and Validation Loss

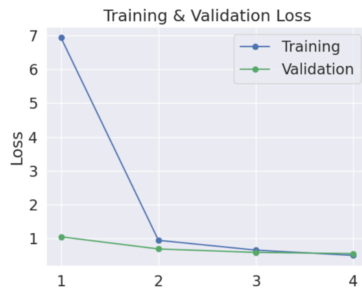
Following image shows the training and validation loss for some of the preselected labels, which you can see the list of them in src/utils/constants.py file. Also you can fine tune the model for them too. If you provide a label then it will be fine tuned for that label, otherwise it will be fine tuned for all of the preselected labels.

(a) education_and_learning

(b) environment_and_sustainability

(c) home_and_garden

(d) politics_and_current_affairs



(e) weather_and_seasons

All implementation details are in src/utils/gpt2.py file. Each model is saved in models/gpt2/label folder.

15.2 Generating Tweets

If we fine tune gpt2-fa on politics label amd we give it prompt about politics, we expect it to generate a tweet about politics. The following image shows the result of this experiment.

It is important to note that we should set max_seq length according to the common length of the tweets in the dataset. If we set it a large number, then model will be overfitted on PAD token and it will generate a lot of PAD tokens.

16 Classification

We have utilized the HooshvareLab/bert-fa-base-uncased model(parsbert v2), a variant of BERT specifically designed for Persian language processing, to extract features from the tweets for our text classification task. The tweets were tokenized using the ParseBERT tokenizer, which splits the text into individual tokens, including special tokens like [CLS] denoting the classification task. The ParseBERT model, consisting of multiple transformer layers, was then employed to process the tokenized input and learn contextual dependencies among the tokens. By leveraging the pre-trained contextualized representations of the HooshvareLab/bert-fa-base-uncased model, we could effectively capture the semantic and syntactic information within the Persian tweets. Finally, the contextualized representation of the [CLS] token was passed through a classification head, enabling the model to map the features to the appropriate number of output labels. This approach empowered us to perform accurate and efficient text classification on Persian tweets.

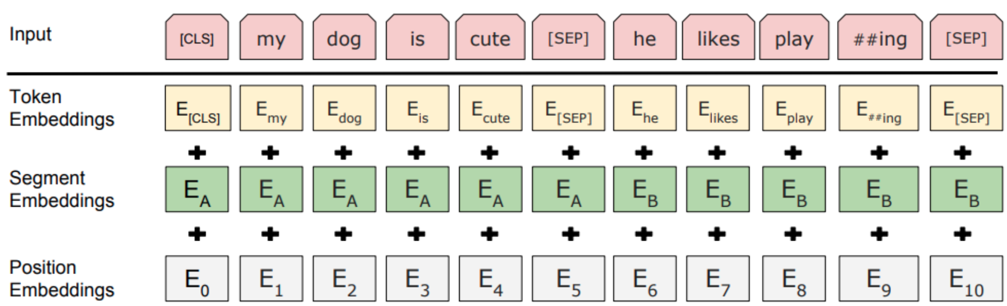


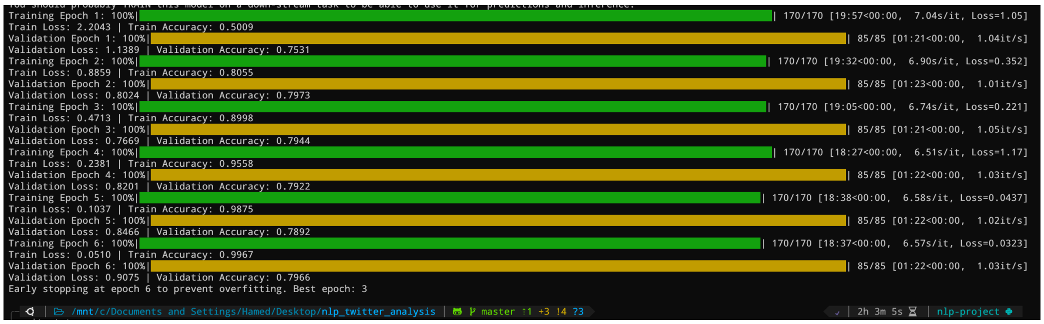
Figure 2: [CLS] token representation is used for classification.

The model is available at https://huggingface.co/HooshvareLab/bert-fa-base-uncased. Also you can find the implementation details in src/utils/parsbert.py file.

16.1 Fine Tuning

```
python src/main.py fine-tune-parsbert path-to-augmented-csv
```

The below image shows the process of fine tuning parsbert model. The script will check for dataset in data/split and if it is not available, it will create it using the augmented data.



We have used early stopping tequnique to prevent overfitting. The accuracy on validation set is roughly 80%.

The model is saved in models/parsbert/best and models/parsbert/final folder. The best model is the model with the highest accuracy on validation set. The last model is the model at the end of the training process.

16.2 Testing

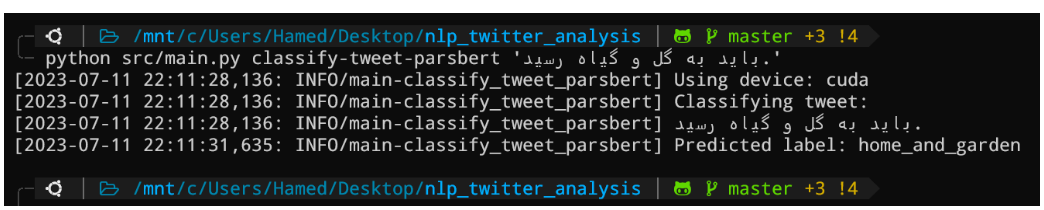
```
    python src/main.py test-parsbert

```

test-loss	test-accuracy
0.7573	0.7985

16.3 Inference

Following image shows how to use our main model for inference. The model will be loaded from models/parsbert/best folder.



17 OpenAI API for classification

We used OpenAI API to classify the tweets. The following code shows what prompt was given to the API.

Probably the most important parameter is the temperature parameter. It controls the randomness of the generated text. The higher the temperature, the more random the generated text is. The lower the temperature, the more predictable the generated text is. The default value is 0.7. We have used 0.2 so that the model be more deterministic.

```python
def get_tweet_label(
        api_key: str,
        api_base: str,
        tweet: str,
        sleep_seconds: int = 10
) -> str:
    openai.api_key = api_key
    openai.api_base = api_base
    topics = list(TOPICS.values())
    messages = [
        {
            "role": "system",
            "content": f"Classify the topic of the future tweet into only one of the following categories: {', '.join(topics)}. some of these tweets are in slang persian language. please try to understand them. Just type the topic and nothing else."
        },
        {"role": "user", "content": f"Tweet: {tweet}"},

    ]

    while True:
        try:
            print("*" * 100)
            print(messages)
            response = openai.ChatCompletion.create(
                model="gpt-3.5-turbo",
                messages=messages,
                temperature=0.2,  # we want the model to be more deterministic
            )
            print(response)
            print("*" * 100)
```

```python
            label = str(response['choices'][0]['message']['content']).strip()
            label = get_clean_label(label)
            break
        except (ServiceUnavailableError, APIError, Timeout, RateLimitError):
            print(f"Some error occurred. Sleeping for {sleep_seconds} seconds and trying again")
            time.sleep(sleep_seconds)
        except KeyError:
            print("KeyError occurred. Setting label to 'unknown'.")
            label = 'unknown'
            break
    return label

```

17.1 Accuracy

Because we are doing classification task, accuracy is the most important metric and we don't need to calculate other metrics like precision, recall and f1 score.

The following table shows the accuracy of the model on small test dataset.

total-tweets	total-correct	accuracy
78	63	0.8077

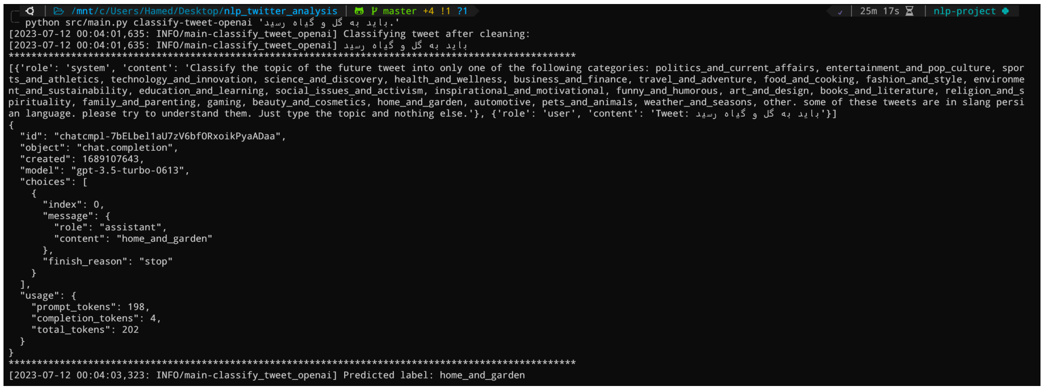
The accuracy is a little bit better than the parsbert model. It was expected because the model is trained on a huge dataset and it is a state of the art model.

There are couple of problems with this approach:

- Very strict API rate limits. To overcome this we had to shrink the test dataset massively.
- The model is not deterministic. We can lower the randomness by lowering the temperature parameter but at end of the day it is still a random process.
- The model is not open source. We have to use the API which is not free.

17.2 Inference

The following image shows how to use the OpenAI API for inference.



18 Report Generation

The report geration process is completely automated. You can find related commands about generating phase 1 and phase 2 reports in ReadME.md file.

19 Resources

- https://github.com/hooshvare/parsgpt
- https://huggingface.co/HooshvareLab/bert-fa-base-uncased
- https://platform.openai.com/docs/introduction
- https://www.geeksforgeeks.org/python-word-embedding-using-word2vec
- https://www.kaggle.com/code/akshat0007/bert-for-sequence-classification
- https://www.kaggle.com/code/nulldata/fine-tuning-gpt-2-to-generate-netlfix-descriptions
- https://colab.research.google.com/github/hooshvare/parsgpt/blob/master/notebooks/Persian_Poetry_FineTuning.ipynb

- https://medium.com/geekculture/easy-sentencepiece-for-subword-tokenization-in-python-and-tensorflow-4361a1ed8e39