Web Science (M) Coursework

Name: Odysseas Polycarpou

GUID: 2210049p

Introduction

Tweets can be classified based on emotion. This assignment regards the development of a Twitter Crawler, to gather and classify/change tweets from Twitter for different sentiment classes based on the content of the tweet. The software is developed using Python Programming Language and several of its libraries and streams Twitter and fetches tweets using the Twitter API and Google Word2Vec model. We also use MongoDB Compass to store our data.

The source code of our software can be found in GitHub at:
https://github.com/podyssea/TwitterCrawler

Some functions are also used from stackoverflow.com for stripping emojis and setting regexes during encoding of some of the symbols that are possible to be found in a tweet string. Instructions for marker can be found in the GitHub repository page.

The executables that need to be ran to fetch, process, and analyse the data, in order, are:

1. **MainTweetCrawler.py** – Streams and saves tweets
2. **MainTweetProcessor.py** – Processes saved tweets
3. **createDataframe.py** – Creates statistics on the fetched data
4. **DatabaseData.py** – Creates data frame with the fetched data
5. **Analysis.py** – Produces classification report

Our software is capable of collecting tweets by streaming the publicly available tweets on Twitter in real-time when executed. When these tweets are saved, they are stored locally in a MongoDB compass database instance as a collection, they are then processed, where we parse the tweets saved and attach a sentiment emotion on each tweet based on its content. In some cases, and where necessary we change the emotion label already on the tweet, if our software evaluates that this is appropriate.

Once the tweets are processed, we use a database evaluation to find out how many tweets were processed in total and how many tweets correspond to each class. Also, the time in which this data is collected is also reported. Finally, our software writes a classification report to evaluate the performance of our model based on several metrics.

After running the MainTweetCrawler.py the streamer starts to fetch data. The time taken to fetch the data is:

12/03/2021 10:26:55 AM - 12/03/2021 10:37:02 AM

For all emotion classes. Therefore, we can see that the fetching of data takes approximately 7-8 minutes.

Data Crawl & Rules

Twitter API

For data streaming and collection, we have used TwitterAPI library. The streaming and collecting of tweets are developed from within the function to operate in real-time when executed. The reason this library was chosen is that it allows the collection of a significant volume of tweets without being limited to twitter's rate limits when fetching tweets. The relevance of the tweets is also an important factor for choosing the API on a real-time, which allow us to carry out a meaningful analysis.

The Twitter API also supports a parameter named track which when specified, works as a keyword match for the tweets fetched. For this implementation, we assign pre-set emotion label terms and encoded emoticons to allow us to filter the results for each emotion class. Also, when fetching tweets, we have noticed that some of them are duplicated. For this reason, we needed to introduce a function in the source code to remove duplications from the feed.

Collected data

The classes in which we classify the fetched tweets are 6:

- Excitement
- Happy
- Pleasant
- Surprise
- Fear
- Anger

For each sentiment class we specify a dictionary with relevant terms in order to specify which hashtag terms are related to each class. With these, we also declare and encode emojis that might be found in a tweeter string body. For example:

{ “excitement": ["#excited", "#excitement", "😁"],

"happy": ["#happy", "#joy", "#love", "😀", ":)"],

"pleasant": ["#pleasant", "#calm", "#positive", "🙂"],

"surprise": ["#sad", "#frustrated", "#negative", "😦", "😮", ":("],

"fear": ["#scared", "#afraid", "#disgusted", "#depressed", "😳", "😢", "😨"],

"anger": ["#angry", "#mad", "#raging", "😡", "😠", "☹"]}

Our software can report the statistics based on the fetch data. After our data is gathered and processed, our createStatistics.py module can generate a csv file with all the numbers for the classes specified. These are, the class name, the number of tweets per class and the times in which the tweets are collected. Below we present a table taken from one of our tests runs of the software.

Sentiment Class	Number of tweets per class	Time started	Time finished
anger	28	12/03/2021 10:27	12/03/2021 10:36
excitement	6	12/03/2021 10:29	12/03/2021 10:34
fear	64	12/03/2021 10:26	12/03/2021 10:37
happy	150	12/03/2021 10:26	12/03/2021 10:29
pleasant	11	12/03/2021 10:27	12/03/2021 10:35
surprise	10	12/03/2021 10:29	12/03/2021 10:36

What this table tells us, is that for some emotion classes, the criteria we have set have not been able to find as many as 150 tweets. For example, “excitement” and “pleasant” has very few processed tweets, either because of ambiguity or small number of fetched tweets classified during data collection. We have also truncated all classes to a limit of 150 tweets. This gives us a total of about 900 tweets before processing.

Processing of Tweets

It was very important to collect clean data. Due to the popularity and freedom of expression in Twitter there might be many tweets which are not easy to be classified or hard to understand. To tackle this, we introduced and defined several functions which parse, filter, transform and even tokenize the content of the tweets.

The first step the API goes through is identify the relevant terms/hashtags/emojis in the emotion class map mentioned above and allocate a class to the tweet fetched. This is what our TwitterAPI thinks is correct as to classification given the relevant term since they are identified as synonymous. After being saved the tweets are filtered again, by a function which parses the tweet body and looks for any presence of other emotion labels. If there is a match, the tweet is discarded, and this gives us the confidence that the tweet we have is clean. The tweets are then stored as a collection in MongoDB compass.

After storing the tweets in the database, they are processed by running the MainTweetProcessor.py module. When ran, this executable, refilters and relabels the tweets using the Word2Vec model. How this is done is by computing the cosine similarity of each relevant “emotion” word, with every word found in the tweet. Then it takes the average and compares the similarity scores for emotion assigned in the first step and reclassifies the tweet if it thinks it is necessary. Moreover, it is important to note that the processing removes all “@” and “…” symbols from the tweets. We have also stemmed and lemmatised the tweets, but this would not preserve the human-readable nature of the tweet which will be used later in crowdsourcing. We use the raw tweets streaming data for crowdsourcing and the lemmatised version for computational use.

Sentiment Use

Emoticons and emojis are very important in sentiment analysis. They are used in the stage of processing. We use this emojis as Unicode strings (found in stackoverflow) and can be passed in the track parameter which the TwitterAPI will recognize as a relevant emotion term. This is the only step where we use emojis mainly due to the fact that they cannot be recognized by the Word2Vec model in terms of similarity.

Crowdsourcing

For crowdsourcing we chose to work with a method on the data we gathered to evaluate whether our data is clean or not, and whether the emotion detecting, and relabelling was accurate. Each worker can parse and analyse the emotion labelling of a tweet. In essence, take the tweet with the initial labelling, then look at the emotion allocated after relabelling using the Word2Vec model and evaluate whether the new emotion label was allocated reasonably.

This can be executed by running the Analysis.py executable from our code folder. This module creates a data frame with the tweet string id, the initial emotion label, and the new human readable emotion label as columns. Based on the values in these we can **use classification report** from sklearn to evaluate our data based on several metrics. We could have chosen a more accurate classification by investing in priced crowdsourcing methods.

Results

Sklearn has a module classification report which we can use to evaluate our data between the human labels and process generated labels. Some of the metrics we have used to evaluate our data on are precision, recall, f1-score, and support. From these we can then find the accuracy, macro average and weighted average. A representation of our results is shown below.

	Precision	**Recall**	**F1-score**	**Support**
Anger	0.27	0.14	0.19	28
Excitement	0.00	0.00	0.00	6
Fear	0.47	0.25	0.33	64
Happy	0.85	0.53	0.65	150
Pleasant	0.13	0.18	0.15	11
Surprise	0.08	0.20	0.12	10
Accuracy			0.38	269
Macro Average	0.30	0.22	0.24	269
Weighted Average	0.62	0.38	0.47	269

Admittedly, from our results table above we can see that the performance is not very good. We can see this from the scores which are relatively low. Especially for “Excitement” and “Surprise”. “Excitement” class got a result of 0 in Precision, Recall and F1-score having a small value of support. On the other hand, we can see a better performance on “Anger” and “Pleasant” while a strong performance can be seen on classes “Happy” and “Fear”. This comes from the fact that some of the tweets fetched for some classes are very few in contract with other classes. The accuracy scores of our evaluation are also provided above. We can see a low performance of Macro average of 0.30, however a bigger score for the Weighted Average of 0.62 for precision values for classes. Recall scores are even lower with 0.22 and 0.38 for Macro Average and Weighted Average consequently. Looking also at the accuracy of the F1-score we can see a low score of 0.38. What the above table tells us is that the performance of our crawler is not the best with lots of low values and disagreement among classes. To create a better crowdsource we could have invested more. This would put a cost on the experiment however produce a more accurate model.