LT2326 HT21 Machine Learning for Statistical NLP

Project Report

Calvin Kullvén Liao

Upvote / Downvote Prediction Using Machine Learning

Introduction

The aim of this machine learning project is to build a neural network model that can predict a comment's reception on social media platforms and/or online forums. When given a string of text input, the model predicts whether it will be positively or negatively received by other netizens.

The motivation behind the idea is that nowadays, a lot of communication happens on online platforms, whether between individuals or from enterprises and public figures. If one can predict a social media post that would be popular, it would be useful for public relations, influencers, and advertising. On the other hand, one can also refrain from posting a content that the model predicts to be negatively received and avoid potential controversy and outcry.

The codes for data-loading and model-training are written in Python 3 in a Jupyter notebook. A trained and saved model can be loaded. The datasets used in the model training are available on Kaggle. PyTorch and torchtext v 0.9~0.10 (the latest version torchtext has a bug that prevents the vocab class from running properly on MLTGPU server) modules are also required.

This report discusses the selection of datasets, the pre-processing of the dataset, the neural network architecture, the evaluation of a trained model, and possible improvements in the future.

Background

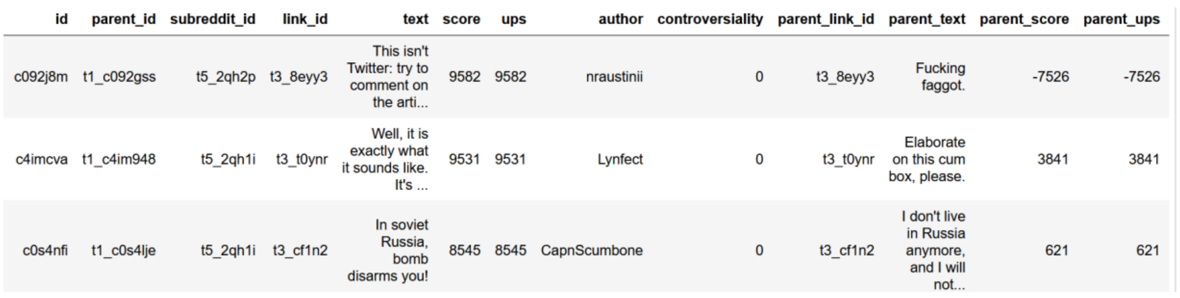
A challenge when it comes to analysing the contents on social media and their reactions is that the content formats are diverse (it can be text, images, videos, or a combination) and that the reaction systems are not uniform across different social media platforms. For example, Instagram allows only a linear number of 'likes' from 0 to infinite, but Facebook allows seven reactions, whereas most other platforms such as YouTube and Reddit has the dichotomous Like/Dislike, Thumbs up/down, or Upvote/Downvote system.

In this project, a collection of comments from Reddit, a web content rating-and-discussion platform, is chosen as the training and testing data, as the content on Reddit is mostly text-based (with occasional links to gifs or images, which can be excluded during data pre-processing), and the reaction to a comment is either an Upvote or Downvote, which when added together indicates a comments overall score.

A problem worth noting with using ‘subreddit’ (one of the many forums on Reddit that dedicates to a particular topic) comments is that most of them are the reply to a ‘parent comment’, meaning the content is, to some extent, contextually related to said parent comment, and that there is a vast number of subreddits dedicated to various topics and hence varied audiences. The same comment may get different reactions in different subreddits or when replying to different parent comments.

Dataset

The datasets used in this project is download from Kaggle (https://www.kaggle.com/ehallmar/reddit-comment-score-prediction) which consists of two csv files, each with 2,000,000 comments. The comments are already classified as either positive or negative based on the number of their up/down votes. A comment also has metadata such as its author, id, controversiality (which is a Boolean automatically marked if the comment has unusually high number up/down votes), as well as the text and these metadata of the comment’s parent comment.



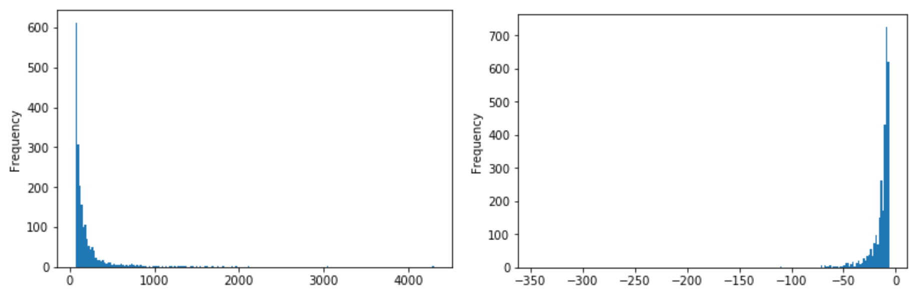
id	parent_id	subreddit_id	link_id	text	score	ups	author	controversiality	parent_link_id	parent_text	parent_score	parent_ups
c092j8m	t1_c092gss	t5_2qh2p	t3_8eyy3	This isn't Twitter: try to comment on the arti...	9582	9582	nraustinii	0	t3_8eyy3	Fucking faggot.	-7526	-7526
c4imcva	t1_c4im948	t5_2qh1i	t3_t0ynr	Well, it is exactly what it sounds like. It's ...	9531	9531	Lynfect	0	t3_t0ynr	Elaborate on this cum box, please.	3841	3841
c0s4nfi	t1_c0s4lje	t5_2qh1i	t3_cf1n2	In soviet Russia, bomb disarms you!	8545	8545	CapnScumbone	0	t3_cf1n2	I don't live in Russia anymore, and I will not...	621	621

An example of the data (not showing all columns).

Methodology

Data pre-processing:

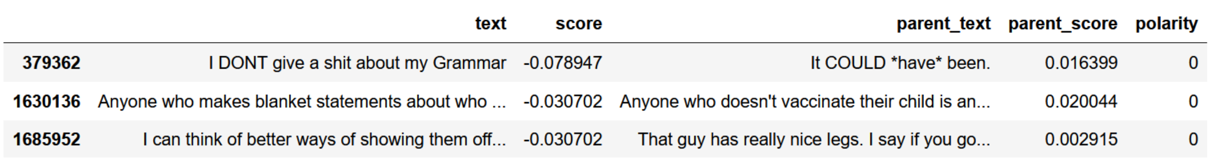
The original dataset files is nearly 2 GB with 4,000,000 entries, for the sake of efficiency, only a smaller portion, ie, 2000 each from positive and negative comments, were sampled. The original aim was to predict the actual score, ie, the total votes that a comment has, but after observing the distribution of scores, it appears that the up/down votes are concentrated, with extremely high/low score being a rarity. But upvotes have a wider range (with scores under 500 being more common) than downvotes (scores below -100 are rare). One way to pre-process this may be excluding the very high/low scores and only keep the ones within the common score range. Or, seeing that most votes fall within a small range, treat it as a positive vs negative binary classification. Since the csv files are already classified as positive and negative, it was easy to add a new column with a binary value: 1 for positive and 0 for negative.



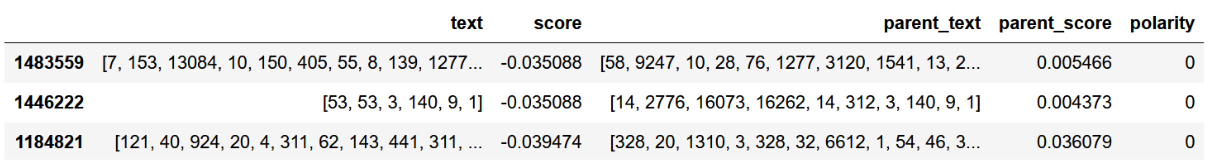
The distribution of upvotes and downvotes.

Some of the comments are non-string or only contain a URL to an image or a video, these are excluded by filtering those that contain only "https//:…/" and no other text content.

The text is turned into lowercase and processed into a list of tokens by using NLTK's TweetTokenizer, which automatically removes delimitators like '\n' or '\r' and is suitable for the internet comments in this case. The punctuations are preserved as they contribute to the tone of the text. It may be interesting to preserve the original latter cases, as something in all capital case may attract more attention, but these comments are not the norm. Lemmatizing the token was also attempted but it took too long to process, and the lemmatizer is prone to make mistake unless it also analyses the part-of-speech of the token. Hence the original tokens were kept. The tokens were fed to torchtext's vocab class to build the vocabulary and its index mapping, which in turn can be used to convert a list of tokens into a list of integers, and to a numpy 1D array.



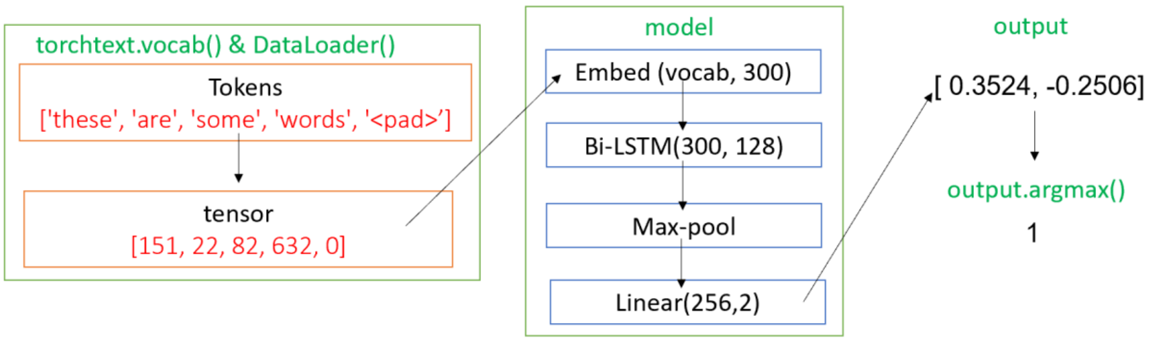
	text	score	parent_text	parent_score	polarity
379362	I DONT give a shit about my Grammar	-0.078947	It COULD *have* been.	0.016399	0
1630136	Anyone who makes blanket statements about who ...	-0.030702	Anyone who doesn't vaccinate their child is an...	0.020044	0
1685952	I can think of better ways of showing them off...	-0.030702	That guy has really nice legs. I say if you go...	0.002915	0



	text	score	parent_text	parent_score	polarity
1483559	[7, 153, 13084, 10, 150, 405, 55, 8, 139, 1277...	-0.035088	[58, 9247, 10, 28, 76, 1277, 3120, 1541, 13, 2...	0.005466	0
1446222	[53, 53, 3, 140, 9, 1]	-0.035088	[14, 2776, 16073, 16262, 14, 312, 3, 140, 9, 1]	0.004373	0
1184821	[121, 40, 924, 20, 4, 311, 62, 143, 441, 311, ...	-0.039474	[328, 20, 1310, 3, 328, 32, 6612, 1, 54, 46, 3...	0.036079	0

The rows of the data is turned into a list of tuples so it can be batch-loaded by using PyTorch's DataLoader class, which loads the tuples and convert the scalars and arrays into PyTorch tensors. The text tensors are of various lengths, so the pad_sequence function is used to pad each tensor with zeros to make every tensor in a batch the same length (as long as the longest one in the batch).

The neural network model is an RNN Bi-LSTM network that takes the padded text tensors as input and outputs a binary prediction.

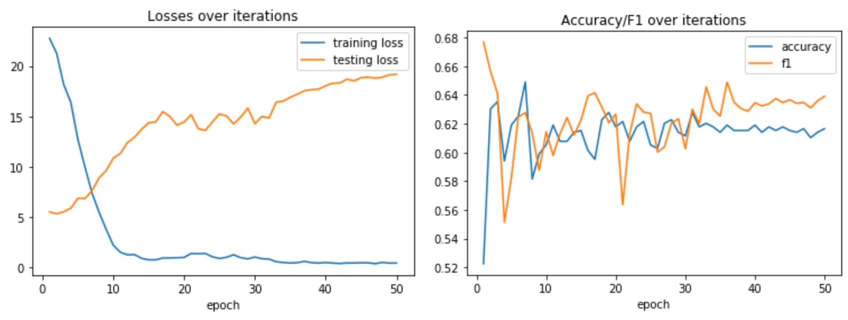


In the model, the text token tensors go through an embedding layer first, then through a bi-LSTM. The RNN-output from the LSTM is then max-pooled and go through a final linear layer for the binary classification. To prevent overfitting, dropout layers are applied between the layers.

For simplicity, I only took the comment text itself as features and the binary pos/neg labels as targets. In other words, it is based on the native assumption that the text itself is indicative of its popularity and ignoring the contextual information such as its parent comment, parent score, and/or authors.

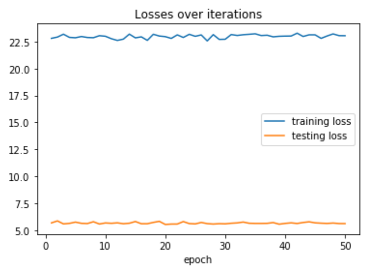
The model used the abovesaid text/binary label as X and Y, and iterated over 50 epochs, and in each epoch, the trained model was evaluated with the test data to collect the model accuracy and F1 of that epoch. The cross entropy losses from each epoch's training and evaluating loops were also collected.

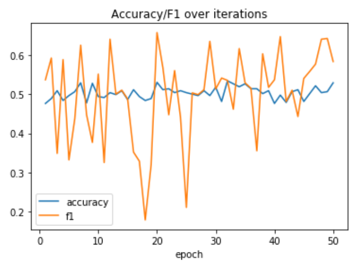
Results



Performance of RNN -> maxpool -> linear

The evaluation results collected from the epochs show that the training loss stabilised after 10 epochs but the testing loss was steadily increasing. In terms of the performance on test data in each epoch, the accuracy and F1 fluctuated a lot in the first 30 epochs, before starting to stabilise at around 62%, it is not a sufficiently satisfactory accuracy if it were to be used to make predictions in real life applications though.





Performance of RNN -> flatten -> linear

I tried skipping maxpooling and put a flattened RNN output through the fully-connected layer, and the result was worse: no decline of losses, and a consistently lower accuracy across the epochs.

I also tried using texts that are not in the test data, converted them to tensors by using the torchtext.vocab class, but the prediction was not meaningful, as the word tokens would by default be indexed 0 for <unk>.

```
get_model_inputs('how much wood would a woodchuck chuck if a woodchuck would chuck wood?')
```

```
(tensor([[    72,    117,   3081,     51,      5,      0, 12133,     29,      5,      0,
             51, 12133,   3081,     15]], device='cuda:3'),
```

Future Work and Conclusion

The model in the project only used a build-from-the-scratch vocabulary mapping, which only contains the tokens in the dataset texts, and since they were not lemmatized, the vocabulary has the various forms of the same word. It would be worth utilising a pre-trained models and tokenizers such as BERT that have been trained with much larger-sized datasets and have superior natural language understanding performance.

As mentioned in the methodology, only a comment's text and pos/neg binary were used as the features and labels in the training process, but predicting the reactions to a social media post would require one to also consider the contextual metadata beyond the post's content itself, as described in the respective Instagram-likes-predicting project by Ding et al (2019) and Cantero Priego (2020) where an Instagram post's image, text caption, and contextual metadata were processed by different sub-models to extract laten information, before being combined and put through the main model's dense linear layers. The datasets in the project do not share the same type of contextual data as Instagram posts, but it would be worth it to consider these data beyond the comment text and extra their information with a sub-model in a similar manner and give the main predictor model more context to make predictions.

In conclusion, it is not enough to predict a human user's reaction to a social media content by looking at the content itself, instead, it is necessary to account for the context the content is in. I would be interested to explore which of the contextual data is more important when it come to predicting reaction and popularity, as well as the deep learning models that would make use of these data.

References

Ding, K., Ma, K. en Wang, S. (2019) Intrinsic Image Popularity Assessment. Proceedings of the 27th ACM International Conference on Multimedia. ACM. doi: 10.1145/3343031.3351007.

Cantero Priego, J. (2020) Predicting the number of likes on Instagram with TensorFlow. Master's Thesis. Universitat Politècnica de Catalunya.

Resources

Datasets
https://www.kaggle.com/ehallmar/reddit-comment-score-prediction

A similar work built on the same datasets
https://www.kaggle.com/danofer/reddit-comments-scores-nlp/notebook

Model and library
https://pytorch.org/tutorials/beginner/text_sentiment_ngrams_tutorial.html