Tweet Sentiment Analysis

Extracting Phrases from Tweets Based on Desired Sentiment

Table of Contents

Project Background

Executive Summary

The goal of the Tweet Sentiment Extraction challenge is to, given the text of the tweet and a sentiment (positive, neutral, negative), extract the word/phrase within the tweet that exemplifies the sentiment provided. The data is made up solely of a set of: a unique ID column, a tweet of varying length, the text post-masking and the associated search sentiment. The "textID" column is assumed to be such that each tweet has a "textID" and no two tweets share the same "textID"; it appears that each ID is an alphanumeric string, 10 characters in length. It is known that the "selected_text" column is a subset of the "text" column; it is also clear based on the data that the "selected_text" is always a contiguous segment of the "text" column. This challenge was extremely difficult; most basic data analysis techniques (regression, clustering, etc.) were potentially achievably using NLP techniques such as TF/IDF or Embeddings/BERT, but did not manage to discern anything related to the goal, and were overall unhelpful. Additionally, while deep learning is better suited to this type of problem, the highest performing models only performed ~20% better then resubmitting the raw data, and most models performed almost no better than the base case. All in all, the best models were those that did not over-complicate the problem – generally RNNs (recurrent neural networks) and CNNs (Convolutional Neural Networks) performed worse than MLPs (Multi-Layer Perceptrons). The best success I achieved was with a combination MLP and GAN (Generative Adversarial Network).

Benchmarking of Other Solutions

Benchmarking Table

Notebook Name	Feature Approach	Model Approach	Train/Test Perf
Twitter sentiment Extraction-Analysis, EDA and Model	Convert words to NLP Embedding	Named Entity Recognition	N/A
roberta interface 5 folds	Convert words to BERT Embedding	roBERTa Interface	71%
0.573 LB score in 10 lines of code	Split sentaces into Python lists	Select unique words only and return as submission	57%
Sentiment analysis + DistilBERT + SQuAD + Q/A tech	BERT Embeddings	BERT For Question Answering	69%
TSE – Full Sentances Only [by me, included for context]	None	None	59%

Figure 1 - Benchmarking Table

Benchmarking Analysis

All except one of the solutions presented above used NLP embeddings; specifically, BERT was used because it is the most state-of-the-art and accurate method for converting words to numerical representation. The one method that did not use BERT was extremely simplistic and did not model the words as anything except unique items.

The first method presented used Named Entity Recognition (NER) to complete the task. While the majority of the code was offloaded to a preexisting library, the basic work consisted of separating words, converting them to numerical representations using BERT embeddings, and then using deep learning to create entities consisting of separate words and phrases from the tweet, then use deep learning to correlate each entity with a name (either negative, neutral, or

positive). Finally based on the highest probability name-entity correlation for which the name matches the requested label, the answer is selected.

The second method uses roBERTa which is based on BERT by Google. roBERTa is model/method used to generalize the BERT embeddings to a wider array of actual problems. The majority of the heavy lifting in this approach was again offloaded to an external library. This method provided the best results of the batch with a success rate of 71% on both the small and large sized test sets.

The third method was the simplest. The method was essentially to split each tweet based on all space characters, and then to convert the list to a set, thereby removing duplicate words. The set was then joined with space characters and submitted as the solution. As a base case this is an extremely good result, and so can be used as a lower-bound to determine whether a model is performing correctly.

The final method was again mostly offloaded to an external library, but again used BERT, this time in conjunction with the Stanford Question and Answer Dataset (SQuAD). Using BERT and SQuAD, the model created answers with 69% accuracy, which is significantly better than the lower-bound case and very close to the roBERTa case.

Data Description and Initial Processing

	textID	text	selected_text	sentiment
0	cb774db0d1	I`d have responded, if I were going	I`d have responded, if I were going	neutral
1	549e992a42	Sooo SAD I will miss you here in San Diego!!!	Sooo SAD	negative
2	088c60f138	my boss is bullying me...	bullying me	negative
3	9642c003ef	what interview! leave me alone	leave me alone	negative
4	358bd9e861	Sons of ****, why couldn`t they put them on t...	Sons of ****,	negative

Figure 2 - Sample Training Data

	textID	text	sentiment
0	f87dea47db	Last session of the day http://twitpic.com/67ezh	neutral
1	96d74cb729	Shanghai is also really exciting (precisely -...	positive
2	eee518ae67	Recession hit Veronique Branquinho, she has to...	negative
3	01082688c6	happy bday!	positive
4	33987a8ee5	http://twitpic.com/4w75p - I like it!!	positive

Figure 3 - Example Testing Data

As can be seen above, the structure of the training data and the test data are the same, except that the test data is missing the "selected_text" column.

	selected_text	Sentiment
count	27480	27481
unique	22463	3
top	good	neutral
freq	199	11118

	Count
neutral	11117
positive	8582
negative	7781

Figure 4 - Test Data Relevant Summary & Sentiment Counts

The tables above describe the content of the train data. The train data has 27,481 unique IDs and sentiments and 27,480 unique tweets and selected text rows. The extra sentiment/id pair is due to a row where text and selected text are both NaN.

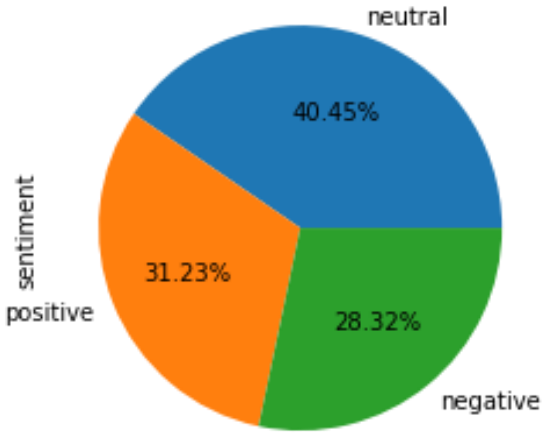

Figure 5 - Sentiment Distribution

The most common sentiment is neutral with ~11000 entries or roughly 41% of the data. Positive is the second most common sentiment with roughly 8600 entries followed closely by Negative with roughly 7800 entries. Overall, the three are occur roughly the same amount, with about a 4:3:3 frequency ratio respectively.

Figure 6 - Top "neutral" Selected Words

Figure 7 - Top "negative" Selected Words

Figure 8 - Top "positive" Selected Words

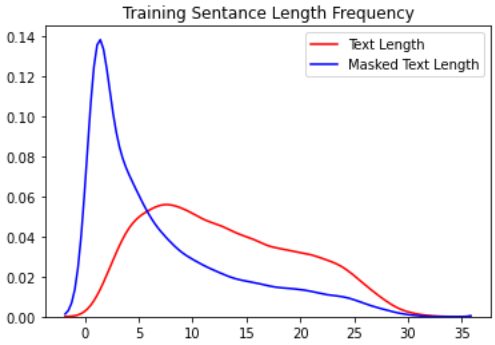

Figure 9 - Training Sentence Length Frequency

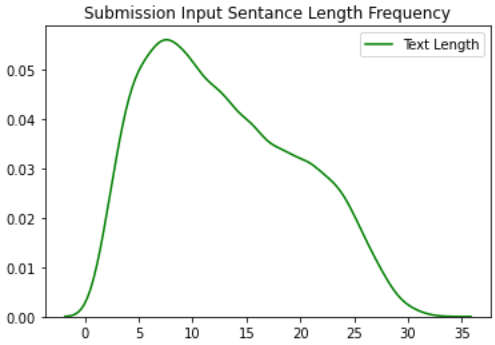

Figure 10 - Testing Sentence Length Frequency

The above figures show required makeup of the input and outputs of the model. Specifically, t figures highlight the most common words in the selected text, the distribution of the sentence lengths before and after having been masked for the training data, and the distribution of sentence lengths for the testing inputs. The word clouds show that negative phrases most often use the words “miss”, “sad” and “sorry”; the positive phrases most often use the words “good”, “love”, “thank”, “Happy” etc.; finally, the negative phrases most often use the word “now”, “lol” and “day” while the rest of the words are a conglomerate of positive and negative words”. The sentence/phrase length data indicates that the model should expect to receive sentences up to 35 words with most of the full sentences being around 5-10 words; the model should return phrases mostly between 0-5 words but should be prepared to return phrases up to or equal to the length of the input sentence. It is shown above that the training data over-represents, with respect to the submission input data, sentences with lengths between 0 and 5 words, and under-represents sentences with lengths between 15 and 35 words.

Modelling and Analysis

The competition has two independent variables, and one dependent variable. The two independent variables are the input tweet, and the and the input sentiment. The dependent variable was the output text mask. The challenge relied heavily on feature creating since the raw text data is not inherently numerically representable, and so needed to be transformer before being used in prediction tasks. For feature creation I elected to use BERT to transform the raw text data into mathematical representations. Specifically, all three major models were based on the huggingface DistilBERT embeddings model, and used a variety of different post-embedding networks to attempt to predict the outcome. DistilBERT is an "evolution" of the BERT model, created by refining the training process with the explicit goal of decreasing the overhead of using the model. DistilBERT runs 60% faster, using 40% fewer parameters, while retaining 95% of BERT's performance (measured by huggingface on the GLUE language understanding benchmark). DistilBERT is an excellent choice for this challenge because the lighter overhead and fewer parameters allows for transfer learning with smaller scale training data (since there are fewer gradients to optimize). The faster speed also allows the model to run efficiently on the relatively weak hardware which Kaggle allocates; this is especially important given the runtime limit of 180min for the challenge.For the modelling of this challenge, I tried three extant versions of a deep neural network.

DistilBERT RNN

The first model I attempted was a Recurrent Neural Network. RNNs (specticially Long-Short Term Memory Networks) are very popular in Natural Language Processing, and so the use

was a natural step. The model was composed of a BERT embedding fed directly into an LSTM, followed by a fully-connected linear (dense) layer to create the base embedding. The base embedding was concatenated to a numerical representation of the sentiment (1313 for 'positive', 2430 for 'negative', and 7974 for 'neutral'). The concatenated rectified using ReLU, then fed through a sparse linear layer with and output length of two. The corresponding shape of the output was the maximum sentence length by 2. Each M-long vector represented the unrectified probability that that specific location was the starting or ending location for the relevant sub-tweet (respectively). For the loss function cross entropy loss was used, and for prediction, log SoftMax was used to rectify the probability, then the maximum probability was chosen. The model performed badly, overall, only getting about 50% accuracy.

DistillBERT GAN

The second model I explored was an extension of the previous model. Understanding that this problem could be framed as a generative problem, I attempted to implement a Generative Adversarial Network. The GAN is architected such that there are actually two separate models which co-evolve by working against each other. The first model is the generator. The generator essentially works to generate an output based on the input. The second model is the discriminator. The discriminator works to, given an input, decide whether that input is a real solution to the problem at hand, or one created by the generator. The generator starts off by generating random data, which is fed directly into the discriminator, the discriminator is then rewarded of it successfully guesses that the data is from the generator, and penalized when it guesses wrong, the generator is inversely, rewarded when the discriminator guesses that the generator is true data, and penalized when it is found out. The discriminator is then fed true data, and penalized when it believes it is fake data, and rewarded

when it correctly discerns that it is true data. The models begin to co-evolve as the discriminator learns to successfully identify the true data, and that data is passed on to the generator as the generator learns how best to trick the discriminator. This model was not perfect for this application since it is easier in this case to assess the success of the generator as opposed to the success of the discriminator, but the GAN is highly successful in these cases in reducing (if not eliminating) over-fitting. Three different architectures of generators were tested.

Discriminator

The discriminator was composed of a DistilBERT embeddings layer, followed by a 1D batch normalization step, followed by a sparse linear layer that reduced the dimensionality from 769 rows to 1. A dropout layer was applied, then the data was rectified using ReLU and normalized again. The output of this step was a Mx1 vector where M is the maximum length of any tweet (in words). The generated mask was then concatenated to the vector to make it's dimensionality 2Mx1. The output was then run through another sparse layer to reduce the dimensionality to 1x1. The resultant value was then rectified using Sigmoid. The output represents the probability that the predicted mask matches the predicted input. The discriminator architecture constant across all generator architecture trials.

RNN Generator

The RNN model from the first model was copied as the generator to try and attempt the same model without overfitting. The model was not very successful only guessing the correct response 53% of the time (compared to a base result of 59%).

CNN Generator

The second model tested was a Convolutional Neural Network. The CNN was intended to retain the same special awareness of the RNN but with less overhead. A convolutional layer and a dense layer were swapped in place for the LSTM. The model was even less successful, only guessing the correct response 48% of the time (compared to a base result of 59%).

MLP Generator

The third model tested was a basic MLP. The MLP was the most basic of the three models tested and didn't incorporate any of the spatial awareness of the previous two models. This makes sense since BERT is intended to contain some special awareness inherently, and the inclusion of a word is not based on the words around it necessarily, but primarily based on its own sentiment. This model had the most success of the three, guessing the correct response 59% of the time. This model essentially replicated the base case by including every word, I believe that this is because the momentum in the Adam optimizer was not enough, and that there was a local minimum where all words were included and so the model settled in this valley without finding the absolute min (which is the goal of the optimizer). I tried this model without the discriminator afterwards and it was successful 58% of the time, and so it was clear that the discriminator had a positive effect (if minimal).

DistillBert For Question Answering

DistillBert for question answering is a pretrained model from huggingface. The model exists to find within a context sentence the answer to a question. The input to the model is the context and the question, and the output is the start index and the end index of the answer. This model, while not pre-trained for this specific task, should have been fairly well suited for use with transfer learning to learn to find the tweet using the sentiment as the question. That said,

after 50 epochs, the model was still very unsuccessful achieving only 50.1% accuracy on the private dataset.

Appendix

GitHub Link

https://github.com/ztipnis/kaggle-competition-tweet-sentiment-extraction
https://github.com/fall2020-intro-ml-apps/final-project-ztipnis