Tweet Sentiment Extraction: Extract Support Phrases for Sentiment Labels

William Lucca (wblucca@wpi.edu)
Pradnya Mahurkar (pmahurkar@wpi.edu)
Surya Vadivazhagu (svadivazhagu@wpi.edu)

Introduction

Goal of competition

The goal of this competition is capturing the sentiment in the language of the tweets by picking out the part of the tweet (word or phrase) that reflects the sentiment.

What are we predicting?

The competition provided us with a training dataset of various tweets, tweet IDs, their desired predictions, and their respective emotions as labels (positive, negative, and neutral).

	textID	text	selected_text	sentiment
	27481 unique values	27480 unique values	22463 unique values	neutral 40% positive 31% Other (1) 28%
1	cb774db0d1	I`d have responded, if I were going	I`d have responded, if I were going	neutral
2	549e992a42	Sooo SAD I will miss you here in San Diego!!!	Sooo SAD	negative
3	088c60f138	my boss is bullying me...	bullying me	negative
4	9642c003ef	what interview! leave me alone	leave me alone	negative

The dataset provided to us in the Kaggle competition

Based on the sentiment labels, we have to predict which part of the tweet (word or phrase) reflects the emotion. The part of the tweet should be predicted in the following manner:

- Positive tweet: Pick out the positive words in the tweet. For example:
  Tweet: *I really really like the song Love Story by Taylor Swift*
  Prediction: *like*

- Negative tweet: Pick out the negative words in the tweet. For example:
  Tweet: *what interview! leave me alone*
  Prediction: *leave me alone*

- Neutral: Pick out the neutral words in the tweet (alternatively the entire tweet)
  Tweet: *Both of you*
  Prediction: *Both of you*

Why is the problem important/interesting?

The problem is interesting because it involves Natural Language Processing (NLP). The course gave us a lot of exposure to creating machine learning models for image classification and weight training. Our group wanted to take a step further and apply our knowledge of machine learning to NLP. Our group was also interested in solving this problem because we wanted to explore off-the-shelf softwares such as BERT, FastText, and Turi Create. It is important to solve this problem because with all the tweets going around on Twitter, it is hard to predict the impact of it on companies, brands, and people. Positive tweets may go viral whereas negative tweets may have a devastating effect. In such cases, capturing the sentiment of tweets is important.

Methods

Baseline Model

Our baseline model simply submitted the entire tweet as it appeared in the test set. In this way we could determine which of our future models are effective at predicting meaningful information, and which are going astray. Running this model using the entire training set for validation produced an average Jaccard score of *0.58878*, and running it on Kaggle's public leaderboard test set gave the score *0.594*.

Counting Words Approach

Our next model for creating predictions after the baseline model was a simple approach which worked by counting words. This method was inspired by Kaggle user Nick Koprowicz[1]. For preprocessing, we simply converted all the tweets to lowercase.

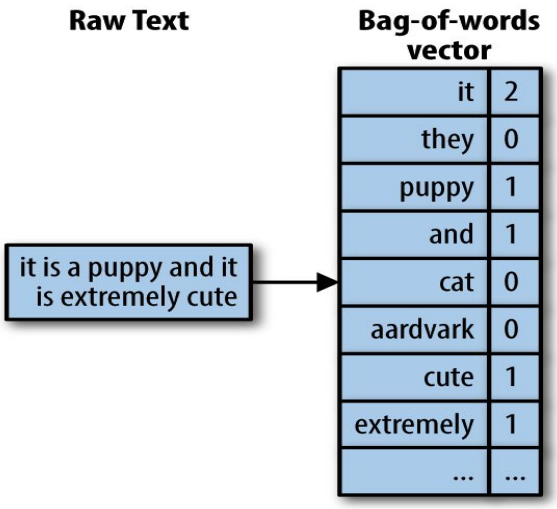
In the model, we found the words and weights for the words which were used in different kinds of tweets (positive, negative, and neutral). We searched all the subsets of a tweet and calculated the score based on the weights. For the positive and negative tweets, the output was the most highly weighted subset. For neutral tweets, we could simply return the neutral tweets.

We split the training set into the training set and validation set in a 1:1 ratio. The program took a couple of minutes to run and gave us the accuracy of *0.65249*. The accuracy of the testing set was after the Kaggle submission was *0.648*.

The experiment allowed us to explore and understand different ways to solve this problem. The accuracy was better than our baseline model, however, not the best that we could achieve. We decided to try another shallow approach which we call the "Bag of Words Approach."

Bag of Words Approach

Our next model for creating predictions was a little deeper than the counting words approach and featured machine learning practices we learned earlier. We began this method by cleaning up the training tweets with some preprocessing like making all the letters lowercase. We also removed punctuation like single quotes and substituted whitespace in for punctuation that separates words like periods, question marks, and exclamation points. After this preprocessing was complete, we read in every word in the training set's positive and negative tweets and gave them each an ID number. This way the selected text for a training example could be represented as a vector of the counts of each word in the text. This is known as a "bag of words" representation, because it considers only the words and not their ordering or context.



Bag of words representation of a string. Multiplicity of words is maintained, but their ordering is lost[2].

Once we had a way of representing tweets as feature vectors, we created a design matrix for all the examples in the training set to perform linear regression using stochastic gradient descent. For *y* labels we used the positive or negative sentiment of the tweet (*+1* and *-1* respectively), and we used mean squared error for the loss function. This way the weights vector that the algorithm trained corresponded to how positive or negative any given word is.

Now the challenge was again selecting the support phrase for a given tweet/sentiment pair. Since we knew the positive and negative values of each single word (or 0 if we have not seen the word), we just needed to select a substring of the cleaned tweet that maximizes this value for positive tweets or minimizes this value for negative tweets. We used Kadane's algorithm for finding the subarray with the largest sum from an array of numbers[3]. This gave us a selection of words from the un-cleaned tweet to use for our submission.

Overall, the results were not astounding, and it even performed worse than the baseline. We created a validation set from 20% of the training data and created a systematic method for testing 72 different sets of hyperparameters for the linear regression training. Ultimately, no matter what hyperparameters were used, the model consistently achieved an average Jaccard index score of about *0.53* to *0.54* (see Appendix A). This indicated to us that this linear regressor model does not properly represent the sentiment of individual words, and a deeper, context-sensitive approach would be necessary.

Bidirectional Encoder Representations and Transformations (BERT) Approach

Devlin et al. through Google AI published a paper in 2015 about the BERT model, and it made waves in the NLP community. BERT made such a splash because of its bidirectionality, among other features. Bidirectional in the NLP community means a model is able to consider the context of a message in two ways[4]. For example, prior to BERT, traditional models would consider phrases like "She is right" and "Turn right" as the same phrase. Under the hood, they would store each word as in the vocabulary once, lacking any context. The word "right" used in the previous example would be stored as a single word, thus lacking any context whatsoever. As humans, we can understand the varying meanings in both these phrases, but prior models failed to do so[5]. However, because BERT is able to represent multiple contexts of the word, it enhances its accuracies across use cases. Finally, Google pre-trained its model across massive amounts of text (for example, the entire text content of Wikipedia)[4], and democratized its model to the general public. Suddenly machine learning enthusiasts could work with this open-source model that would have otherwise been impossible to train with their own hardware and time.

Taking a deep dive into BERT's architecture helps with understanding just how superior it is. Since this is a paper regarding our implementation of BERT, we will focus on what components were helpful in the sentiment analysis task. BERT uses the Masked Language model, which was adapted for our project[4]. The model randomly puts masks on words in a tweet and then later tries to guess the words based upon the context of other words used previously. This helps the model gain higher accuracy when predicting what words in a tweet give the sentiment[6].

The BERT architecture relies upon Transformer[7] to learn about context. Transformer is another Google-developed model that has an encoder and decoder to "transform" text and create a prediction from it. Transformer stands out and is a premier use case for this Kaggle challenge because it parses the entire tweet *at once*, rather than in a directional fashion. Directionality can cause a penalty for the context of the tweet. In this challenge, identifying sentiment in such a convoluted dataset as ours is only possible when the traditional left-to-right direction is abandoned.

We chose the HuggingFace BERT transformer because it satisfied all of our criteria for a NLP model for this task, and had strong evidence of scoring highly for other text-extraction tasks. Its ability to understand context stood out to us as selecting what words would be useful for sentiment extraction is highly dependent upon the context of that word, since tweets are so small (128 characters max)[8]. When implementing our BERT model and fine-tuning it for our use case, we implemented various techniques to process our data and ensure its integrity. We used text padding to ensure that the length of a tweet wouldn't impact its extracted selection being wrongly selected due to the BERT transformer, for example. We used the binary

cross-entropy loss function after analysis of the training data and seeing that selected text for "neutral" class tweets are almost always the entire tweet itself, due to the tweets being extremely vague. Thus, binary seemed the best choice. It also had the lowest cross entropy across varying iterations of the project. For our activation function, we decided to use the sigmoid function as it paired well with BERT for this project. Identifying which words would be most impactful leads to a value between (0,1) for each word in how relevant it is to the labeled sentiment, so thus we felt sigmoid would be the best. Our model performed marginally better than our shallow implementations, scoring a Jaccard similarity of *0.663* on the Kaggle competition, placing us at 794th position in the competition. We also conducted validation testing on 20% of the dataset to ensure our mode was reporting accurate values and was not biased by hyperparameters, which scored *0.64* in Jaccard similarity.

Results

To verify that our models were performing some sort of learning, we compared their scores with our "brainless" baseline submission. The models which performed better than this baseline predicted something meaningful, and as such they are marked in green. *Bag of Words* is marked in red because it performed worse than our baseline.

Model	Type	Validation Score (Jaccard)	Kaggle Score (Jaccard)
Submit Whole Tweet	Baseline	0.58878	0.594
Counting Words	Shallow	0.65249	0.648
Bag of Words	Shallow	0.54093	0.547
BERT w/preprocessing	Deep	0.64122	0.663

Comparison of our three models and a baseline score

Conclusions

While our multiple approaches produced many results different from the baseline score, they did not all perform better. The *Counting Words* and *Bag of Words* models were both "shallow" in that they did not use additional operations in hidden layers of abstraction, but they did approach the data in different ways. *Counting Words* essentially created separate word classifiers for positive and negative scenarios, using a different value for each word in its vocabulary depending on the sentiment. *Bag of Words* used linear regression to determine a single weight

value for each word, more positive values therefore being more likely to be selected in positive sentiment predictions and more negative for negative sentiment predictions. This strategy of separate values depending on positive and negative sentiments likely allowed the *Counting Words* model to derive some form of context for words with multiple meanings.

However, our deep model (*BERT*) approach performed best because it was able to consider how the meaning of words changes with the words around them. Since this model uses the relation of surrounding words in the text to create more features, it was able to produce predictions based on which specific context each word exists in. Since a single run of training and testing the *BERT* model took multiple hours to complete, we did not have the capacity to conduct a thorough hyperparameter search as we did for *Bag of Words*. This likely could have improved scores further, especially considering the complexity in training a multi-layer model. For future work, if we had a more powerful GPU unit for training, we could implement hyperparameter optimization. We exceeded the amount of hours allotted on Kaggle, so more hyperparameter optimization could lead to much higher accuracies in the future.

References

[1] Nick Koprowicz. 2020. A simple solution using only word counts. Retrieved May 11, 2020 from https://www.kaggle.com/nkoprowicz/a-simple-solution-using-only-word-counts

[2] Amanda Casari and Alice Zheng. 2018. *Feature Engineering for Machine Learning*. O'Reilly Media, Inc. Retrieved May 11, 2020 from https://uc-r.github.io/public/images/analytics/feature-engineering/bow-image.png

[3] Rohit Singal. 2018. *Kadane's Algorithm — (Dynamic Programming) — How and Why does it Work?* Medium. Retrieved May 5, 2020 from https://medium.com/@rsinghal757/kadanes-algorithm-dynamic-programming-how-and-why-does-it-work-3fd8849ed73d

[4] Devlin et al., 2019. *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding*. Google Research. Retrieved May 11, 2020 from https://arxiv.org/abs/1810.04805

[5] Cai 2019. *Sentiment Analysis of Tweets using Deep Neural Architectures*. Stanford University. Retrieved May 11, 2020 from https://web.stanford.edu/class/archive/cs/cs224n/cs224n.1194/reports/custom/15786247.pdf

[6] Lin et al., 2019. *A BERT-based Universal Model for Both Within- and Cross-sentence Clinical Temporal Relation Extraction*. Proceedings of the 2nd Clinical Natural Language Processing Workshop, ACM 2019. Retrieved May 11, 2020 from https://www.aclweb.org/anthology/W19-1908.pdf

[7] Song et al., 2020. *Utilizing BERT Intermediate Layers for Aspect Based Sentiment Analysis and Natural Language Inference*. Sun Yat-sen University. Retrieved May 11, 2020 from https://arxiv.org/abs/2002.04815

[8] Wof et al., 2020. *HuggingFace's Transformers: State-of-the-art Natural Language Processing.* Cornell University Publishing. Retrieved May 11, 2020 from https://arxiv.org/abs/1910.03771 and https://github.com/huggingface/transformers

[9] Vaswani et al., 2017. *Attention Is All You Need*. Google Research. Retrieved May 11, 2020 from https://research.google/pubs/pub46201/

Appendix A: Bag-of-words Data

Bag-of-words hyperparameter testing results on validation set

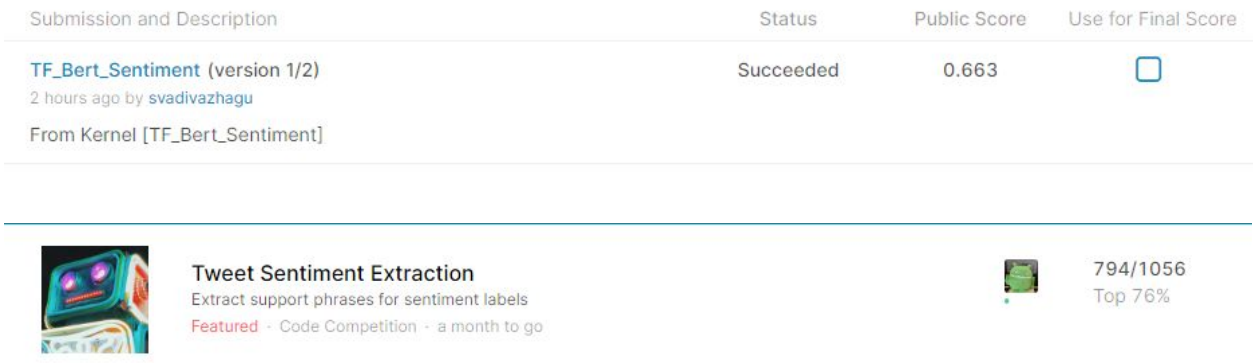
Epsilon	Batch Size	Epochs	Alpha	Jaccard
0.5	64	30	0.00005	0.5405
0.5	64	30	0.0001	0.5386
0.5	64	30	0.0005	0.5433
0.5	64	40	0.00005	0.54
0.5	64	40	0.0001	0.5392
0.5	64	40	0.0005	0.5397
0.5	64	50	0.00005	0.5371
0.5	64	50	0.0001	0.5387
0.5	64	50	0.0005	0.537
0.5	128	30	0.00005	0.5348
0.5	128	30	0.0001	0.5353
0.5	128	30	0.0005	0.5393
0.5	128	40	0.00005	0.5426
0.5	128	40	0.0001	0.5393
0.5	128	40	0.0005	0.5375
0.5	128	50	0.00005	0.5381
0.5	128	50	0.0001	0.5353
0.5	128	50	0.0005	0.5324
0.5	256	30	0.00005	0.5356
0.5	256	30	0.0001	0.5394
0.5	256	30	0.0005	0.5375
0.5	256	40	0.00005	0.5394
0.5	256	40	0.0001	0.5329
0.5	256	40	0.0005	0.5416
0.5	256	50	0.00005	0.5375
0.5	256	50	0.0001	0.5328
0.5	256	50	0.0005	0.5369
0.5	512	30	0.00005	0.5333
0.5	512	30	0.0001	0.5337

0.5	512	30	0.0005	0.5374
0.5	512	40	0.00005	0.5331
0.5	512	40	0.0001	0.5317
0.5	512	40	0.0005	0.5364
0.5	512	50	0.00005	0.5362
0.5	512	50	0.0001	0.5384
0.5	512	50	0.0005	0.5344
1	64	30	0.00005	0.5387
1	64	30	0.0001	0.5367
1	64	30	0.0005	0.5364
1	64	40	0.00005	0.5333
1	64	40	0.0001	0.5401
1	64	40	0.0005	0.5375
1	64	50	0.00005	0.5339
1	64	50	0.0001	0.5388
1	64	50	0.0005	0.5386
1	128	30	0.00005	0.5346
1	128	30	0.0001	0.5397
1	128	30	0.0005	0.5342
1	128	40	0.00005	0.5339
1	128	40	0.0001	0.5364
1	128	40	0.0005	0.5386
1	128	50	0.00005	0.5367
1	128	50	0.0001	0.5372
1	128	50	0.0005	0.5364
1	256	30	0.00005	0.5374
1	256	30	0.0001	0.5315
1	256	30	0.0005	0.5362
1	256	40	0.00005	0.534
1	256	40	0.0001	0.5381
1	256	40	0.0005	0.5381
1	256	50	0.00005	0.5323
1	256	50	0.0001	0.5348

1	256	50	0.0005	0.5309
1	512	30	0.00005	0.5358
1	512	30	0.0001	0.5362
1	512	30	0.0005	0.5343
1	512	40	0.00005	0.5329
1	512	40	0.0001	0.5365
1	512	40	0.0005	0.5338
1	512	50	0.00005	0.5375
1	512	50	0.0001	0.5341
1	512	50	0.0005	0.5326

Appendix B: Kaggle Submission

Kaggle submission for our project, and the resulting score of 66.3% placing us at 794th.



Here's a URL to our submission:
https://www.kaggle.com/svadivazhagu/tf-bert-sentiment?scriptVersionId=33797926