Niccole Riera

COE 379L - Software for Responsible Intelligent Systems

Project 4 Report

Introduction and Project Statement

X, formerly known as Twitter, is a popular platform where millions of people go to leave opinions and reviews of places, topics, and companies. When people send tweets, they often use abbreviations and slang in their messages. The slang used on X varies greatly, with some users using very radical slang that could be perceived as negative when in reality it is positive, such as “sick”, or abbreviations like “u” instead of “you”. “Sick” is just one example of many words that literally mean one thing, but socially mean another. Considering this, as well as the fact that language evolves rapidly over time, I wondered how well a model could predict whether a sentence contains slang or is written more informally than a sentence that is written with a literal, formal tone. To further investigate this, I decided to fine-tune an existing transformers model on an NLP task: classifying tweets as containing slang/informally written vs formally written.

Data Sources and Technologies Used

The data that I used for this task came from tweets sent to a variety of airlines. The original dataset that I pulled from is linked in my github repository for this project so it can be viewed as a reference. The original dataset contains 13,057 rows of tweets. Since I wanted to manually label each of the tweets, I decided to use about 300 tweets. The final dataset that I used contained 302 tweets. I wanted to categorize each of the tweets as having slang/written informally vs no slang/formal. I went through the first few hundred tweets and found 151 tweets

that I marked as “yes” because they contained slang/informal phrases. Then, I found 151 tweets that I marked as “no” because they didn’t contain slang and were written more formally. I did 151 for each category because I wanted to make sure the dataset was balanced and there was an equal amount of tweets for each category. The criteria to categorize the tweet was whether there were abbreviations (such as “u” instead of “you”), acronyms (such as LOL), or words/phrases that are slang or not meant to be taken literally. If any of those applied, I would categorize the tweet as containing slang/informal language and mark “yes”. I utilized the transformers library to fine-tune the distilBERT model on this specific task. The distilBERT model is a smaller version of the BERT model which is used on various NLP tasks, yet it retains most of the BERT model’s performance. Since this dataset is relatively small, I figured the distilBERT model would also be a good option since it is a smaller model.

Methods Employed

Once I had the data labeled, I exported the 302 rows of tweets to a separate excel sheet to make processing the data easier. I imported it as a Pandas DataFrame. I split the data into training and testing DataFrames. I then split the testing data into testing and validation DataFrames. Overall, I had 211 tweets for training data, 46 tweets for testing data, and 45 tweets for validation data. I converted each of these DataFrames into a dataset object. I then class-encoded the “slang/informal” column to make evaluating the model easier. When I first tried training the model, I ran into a problem here because I didn’t rename the column from “slang/informal” to “label”. According to ChatGPT, the trainer class requires that the name “label” be used and expects this format to represent the actual labels for the input data. Since it wasn’t originally labeled this, the trainer couldn’t identify the column that contained the labels,

so it couldn't train the model. After this, I wrote the preprocess function to take in a sample as an input and access the "text" key to pass to the tokenizer. I added each of the test, train, and validation DataFrames to a dictionary and converted the dictionary into a DatasetDict to make it easier to map the preprocess function to each dataset and to access each of the sets later on in the Trainer. I still hadn't dealt with padding which was needed to save space, so I instantiated a DataCollatorWithPathing object and passed the tokenizer. The metric I used for evaluating the model was accuracy. I tried other metrics such as Matthew's correlation, f1, and precision, but they all gave errors, so I stuck with accuracy. To use this metric, I created the compute_metrics function to take in the predictions as a tuple and return a dictionary that maps the string to floats. For the training arguments, I used 10 epochs. When I first successfully trained the model, I had used 5 epochs. However, since it was computing relatively quickly, I decided it would be okay to increase the epochs. Then, I defined the Trainer object using the model, the arguments, the tokenized test and validation datasets, the tokenizer, the data collator, and the function for computing metrics. Finally, I trained the model. When I originally trained the model, under the "training loss" column, all of the outputs said "no logs". Later, I realized that the default logging steps is 500, which means that no loss gets reported before 500 steps. I wanted to see if I could get some sort of output, so I tried lowering the logging steps to 100, and I still got no logs for the first 7 epochs, but I got values for the last 3. To help visualize the amount of true/false predictions, I created a confusion matrix for the validation data and the test data.

Results

The loss indicates the discrepancy between the model's predictions and the actual value of the training/validation data. I found that the model deviated by 33.9% from the actual value of the training data and by 47.6% away from the actual values of the validation. Lower values for

loss are better because it means that the predictions are closer to the actual value, which means better performance. The loss for the training data is much lower than the loss for the validation data, which is to be expected since the model is being trained on the training data. The loss for the validation is not as good but it's still relatively small. The accuracy of the model came out to be 0.844, meaning the model correctly classified 84.4% of the dataset. I created a confusion matrix to visualize how many tweets were correctly classified. I did one for the validation data and one for the test data. On the test data, there are 26 true negatives and 13 true positives. This means that 39/46, 84.7%, of the tweets were correctly classified. This somewhat matches what we see from the model's accuracy of 84.4%. On the validation data, there are 22 true negatives and 18 true positives. This means that 40/45, 88.8%, of the tweets were correctly classified. Given these results, I am fairly confident in the model's ability to classify tweets as slang/informal vs formal.

References

"Distilbert Model Guide." A Streamlined Transformer Model for NLP Tasks, www.modelbit.com/model-hub/distilbert-model-guide. Accessed 2 May 2024.

Fine-Tune a Pretrained Model, huggingface.co/docs/transformers/training. Accessed 2 May 2024.

Lokare, Ganesh. "Effortless Sentiment Analysis with Hugging Face Transformers: A Beginner's Guide." *Medium*, Medium, 4 Feb. 2023, medium.com/@lokaregns/effortless-sentiment-analysis-with-hugging-face-transformers-a-beginners-guide-359b0c8a1787.

Main Classes, huggingface.co/docs/datasets/package_reference/main_classes. Accessed 2 May 2024.

minksminks 2, et al. "How Can I Plot a Confusion Matrix?" *Stack Overflow*, 1 Nov. 1961, stackoverflow.com/questions/35572000/how-can-i-plot-a-confusion-matrix#:~:text=you%20can%20use%20plt.matshow%28%29%20instead%20of%20plt.imshow%28%29%20or,heatmap%20%28see%20documentation%29%20to%20plot%20the%20confusion%20matrix.

"Question Answering - Hugging Face NLP Course." *Question Answering - Hugging Face*, huggingface.co/learn/nlp-course/chapter7/7?fw=pt#question-answering. Accessed 2 May 2024.

user3668129user3668129 4, and Timbus CalinTimbus Calin 14.5k66 gold badges4444 silver badges6666 bronze badges. "Why There Are No Logs and Which Model Is Saved?" *Stack Overflow*, 1 Apr. 1968,

stackoverflow.com/questions/73182816/why-there-are-no-logs-and-which-model-is-saved.