BTNLP Exercise 5

Eva Gavaller, Diana Davidson and David Matkovic

February 19, 2024

1 Pre-processing Reflection

In Exercise 4, we first pre-processed the Twitter data by removing emojis and emoticons with Regex. Emojis are important for sentiment, especially those with facial expressions. They can be paired with a sentence in a sincere or sarcastic way (the latter is harder for sentiment analysis tools like VADER to parse). Emojis have been shown to intensify polarity for both positively and negatively rated sentences. In the initial preprocessing phase, we also removed punctuation (periods, commas, exclamation marks, question marks etc). The removal of periods first still kept remnants of website URLs in the tweets, leaving elements that are 1) not useful for sentiment analysis and 2) annoying to remove with regex since URLs have many different configurations. The use of exclamation marks (and the number of them) make sentences more intense and impact the polarity to be more positive or negative, depending on the rest of the sentence [4]. We also made all the texts lowercase, removing any potential sentences or phrases that were written in all caps – another intensifier affecting polarity scores [4].

Another factor that potentially impacted the results of our sentiment analysis was using NLTK's built-in stopword list. A few categories of words in NLTK's built-in stopword list include words that indicate negation (eg. not, no, none), intensifiers (eg. very), quantifiers (eg. many, some, all, few), words that help with expression opinions (eg. for, against), and modal verbs — whether combined with negation or not (eg. must, should). If we had made our own stopword list that would keep intensifiers, quantifiers, and negation, then we could perhaps have better results (as in, more true positives and true negatives). Hutto & Gilbert (2014) found that VADER catches 90% of instances where negation flips the polarity of a text (eg. "The food here isn't really all that great.") [4]. Because we used NLTK's built-in stopword list however, words like isn't, hasn't, couldn't, and shouldn't were removed, affecting the polarity scores (and potentially making them the opposite of intended by the text). Slang words or phrases sometimes incorporate prepositions,

which are included in NLTK's built-in stopword list. For example, "go off sis" once put through the stopword filter would become "go sis". Inspecting words for their lexical properties (ie. parts-of-speech) could improve sentiment analysis by providing more context in sentences [4].

Since Twitter data is from a social media site, this data is especially prone to spelling errors, making some parts of sentences harder to parse for polarity scores and more difficult to deal with in the pre-processing stage. Twitter's character limit doesn't leave much room for incorporating broader context into sentences. Sentiment analysis tools do better on inspected polarity as more sentences and more contextual information are included in the dataset. Further preprocessing we did in this exercise included removing all the "data found" and "unk" entries in the dataframe — of which there were about 2000 in total.

2 Explanation of Libraries

2.1 NLTK

NLTK has a SentimentAnalyzer and a subjectivity module inside the nltk.sentiment library. With the built-in subjectivity corpus, one can calculate subjectivity and objectivity for that specific dataset, but Diana couldn't find a way to do this using the Twitter data. To calculate the subjectivity of the built-in dataset, one would have to assign variables to the subjective and objective documents: *subj_docs = [(sent, 'subj') for sent in subjectivity.sents(categories='subj') [:n_instances])* and *obj_docs = [(sent, 'obj') for sent in subjectivity.sents(categories='obj') [2] [:n_instances])*. Then the majority of the dataset will be training data, and the rest will be test data to see how well the sentiment analysis runs with NLTK's Sentiment Analyzer. One can use different trainers (eg. NaiveBayes) to compare the F-measures, precision, and recall values for the model [2]. NLTK also has a frequency distribution feature (easy to implement with this line: *lower_fd = nltk.FreqDist([w.lower() for w in fd])* once the data has been tokenized), which can be insightful for large datasets to see words are used most (or least) often [2].

2.2 VADER

VADER has a SentimentIntensityAnalyzer, which can give the polarity scores (compound, negative, neutral, and positive). The compound score has a range of -1 to 1, and the positive, negative, and neutral scores have a range of 0 to 1, because they are ratios for the proportions of text that fall into each category [1]. In the literature, the typical thresholds for positive, neutral, and negative sentiment are set at the following: compound scores greater than or equal to 0.5 are positive, less

than or equal to -0.5 are negative, and neutral is in between. The positive, neutral, and negative scores are useful for multidimensional measures of a sentiment in a given sentence, while the compound score is useful if you want a single measure of sentiment [1]. VADER is especially sensitive to both polarity and intensity of sentiments expressed in social media contexts. Hutto & Gilbert (2014) found that VADER outperformed human raters in correctly classifying sentiment of tweets into positive, neutral, or negative classes.

2.3 TextBlob

TextBlob sentiment analysis is a rule-based sentiment analysis [3]. Sentiment analysis using TextBlob is quite straightforward. According to Alexandros Pappas from Medium, one can simply install TextBlob [from textblob import TextBlob] and perform sentiment analysis by using the [.sentiment] function. Before this, however, the text or document that will be analyzed needs to be made into a TextBlob object using *[text = "This is a great tutorial on sentiment analysis!" / blob = TextBlob(text)]* for example [6]. After using [.sentiment] on the textBlob object, a tuple will be output, showing values of both polarity and subjectivity [6]. Polarity is the positivity or negativity of the text. The range of polarity using the TextBlob sentiment analysis is negative one to one [6]. As implied, if the score is closer to negative one, the sentiment is more negative, and if the score is closer to one, the sentiment is more positive. Depending on the documentation, 0 could be specified as a "neutral" as in the article by Pappas [6]. However, other articles do not specify the "neutral" score so it is uncertain if "neutral" is a real category or not. Subjectivity gives a score based on how subjective the document is. This score goes from 0 to 1 [6]. Additionally, to get just one of these scores (instead of both as a tuple), simply coding *[blob.sentiment.polarity]* or *[blob.sentiment.subjectivity]* will output them individually [6].

2.4 Spacy

In comparison to TextBlob, Spacy requires a bit more coding to do sentiment analysis, depending on the approach that is taken. One easy approach is using TextBlob through Spacy. After first importing Spacy and loading the Spacy model that will be used, one has to additionally import spacytextblob as such *[from spacytextblob.spacytextblob import SpacyTextBlob]* [7]. Then using *[nlp = spacy.load('en_core_web_sm')]*, the function "nlp" is made [7]. After, we need to use *[nlp.add_pipe('spacytextblob')]*, to add the newly imported spacytextblob to the pipeline [7]. Similarly to TextBlob, the text or document to be analyzed has to be turned into an object that Spacy can work with [7]. *[text = 'I had a really horrible*

day. It was the worst day ever! But every now and then I have a really good day that makes me happy.' / doc = nlp(text)] for example, will make this text into a Doc object (Spacytextblob, n.d.). After this, simply use [._.blob] to use the function that TextBlob provides, such as [doc._.blob.polarity] or [Doc._.blob.subjectivity] [7]. Unlike TextBlob, Spacy requires a bit more work to do sentiment analysis. "Spacy does not have built-in sentiment analysis. Instead, you have to customize its text classification module" [5]. Vishnu Nandakumar from Medium coded their own model as an example for customizing the text module classification of Spacy (see Figure 1) [5]. You can see that they added the labels "positive" and "negative" and wrote a code to train the model [5]. Nandakumar then packaged the model and now it can be easily implemented into the "nlp" function [5]. Instead of using *[nlp = spacy.load('en_core_web_sm')]*, if *[nlp = eng_spacysentiment.load()]* is used (after installing the eng_spacysentiment model), then the Doc object now has the function [.cats] that will output the positive and negative scores at the same time [5]. In comparison to the TextBlob sentiment analysis, some may like seeing both positive and negative scores at the same time. Depending on the goal of the analysis, however, it may be more useful to have one polarity score and then the subjectivity score. All in all, both the TextBlob and Spacy sentiment analyses are useful, but using TextBlob is a fair bit simpler and since it can be implemented in Spacy as well, seems like a good choice for beginners.

3 Application and Results

Comparing the results of both types of sentiment analysis, it is easy to determine that both VADER and TextBlob have their individual strengths and weaknesses. The processing time of VADER was a lot longer than that of TextBlob. However, VADER looks like it was more accurate with "True Negative" judgements while TextBlob had a higher percentage of "True Positive" judgements. VADER also had a surprisingly bad performance identifying neutral statements. Additionally, there is a chance that these percentages may be quite off, as TextBlob does not have a neutral polarity category. We realize that we could have coded a sort of subjective neutral category based off of the compound score that TextBlob gives for sentiment, but for the sake of time, we did not. Moreover, it is important to notice that in the TextBlob sentiment analysis data frame had rows with no polarity score given. Consequently, in the visualization of the TextBlob sentiment in comparison with the sentiment of the original data frame, there is a category for "One Factor w/o Polarity." In this bar, the "True" values represent times when TextBlob did not give a polarity score, while the "False" values indicate the equivalent of a "False Neutral," as there is no way there can be a "True Neutral" in TextBlob if there is no category

coded into it. Seeing the differences between VADER and TextBlob, it is hard to accurately compare the scores without either removing the neutral category from VADER or coding in the neutral category in TextBlob.

As we found a piece of documentation that stated that Textblob's "neutral" polarity score is 0, we changed our function to use 0 and "neutral" and both of the VADER and TextBlob results started to look more similar, though there were still some differences. VADER was still better at calculating negative scores, as it had a higher amount of "True Negative" scores, while TextBlob had a higher amount of "True Neutral" scores. (It is worth it to note that the "False" sections of each of these categories mean the the score was not the same as the category e.g. the actual score was "positive" and the predicted score was not "positive".)

To give one additional point about this section of coding, due to inconsistencies with module names and our computers, we were not able to use ConfusionMatrix from pandas_ml and instead, coded a function to compare the sentiments of two data frames and output "True Positive", "False Negative", "True Negative", etc. This is the reason we used a stacked bar chart instead of a matrix, as we felt like this would be an easier visualization for us to understand, as well as to code, considering we are beginners.

4 Afterthoughts

All in all, these past assignments were quite hard for us as beginners. During many sections in the exercises, we were not sure how to code them at all. But, while doing these assignments, we learned a lot by having to work out everything on our own. We especially enjoyed the Regex portion, as it was very helpful to learn and there wasn't a steep learning curve. It was also very helpful to work on the assignments in teams, as we could split up and individually do different parts of each exercise, making the assignment go much faster. Additionally, we could discuss problems and help each other with our code. It would have been fun to go into more details in using each library, because learning about eight in one assignment was overwhelming and felt relatively surface-level.

References

[1] *About the Scoring*. (2021). VaderSentiment. Retrieved February 16, 2024, from `https://vadersentiment.readthedocs.io/en/latest/pages/about_the_scoring.html`.

[2] Bird, S., Klein, E., and Loper, E. (2009). Natural Language Processing with Python. O'Reilly Media Inc. https://www.nltk.org/book/

[3] Es, S. (2023, August 30). *Sentiment Analysis in Python: TextBlob vs Vader Sentiment vs Flair vs Building It From Scratch*. neptune.ai. https://neptune.ai/blog/sentiment-analysis-python-textblob-vs-vader-vs-flair

[4] Hutto, C., & Gilbert, E. (2014, May). Vader: A parsimonious rule-based model for sentiment analysis of social media text. In *Proceedings of the international AAAI conference on web and social media* (Vol. 8, No. 1, pp. 216-225).

[5] Nandakumar, V. (2022, February 9). *AI in the real world —2. Sentiment Analysis using spaCy pipelines*. Medium. https://medium.com/mlearning-ai/ai-in-the-real-world-2-sentiment-analysis-using-spacy-pipelines-b39a2618d7c1

[6] Pappas, A. (2022, December 2). How to perform sentiment analysis in Python using TextBlob. Medium. https://medium.com/@apappascs/how-to-perform-sentiment-analysis-in-python-using-textblob-dc64f262e9ea

[7] spacytextblob. (n.d.). https://spacytextblob.netlify.app/

A Appendix

See next page.

```python
# data processing

train_texts = df['text'].values
train_labels = [{'cats': {'positive': label == 'positive',
                          'negative': label == 'negative'}}
                for label in df['sentiment']]

# training the model
nlp = spacy.blank("en")
config = Config().from_str(single_label_bow_config)
text_cat = nlp.add_pipe('textcat', config=config, last=True)
text_cat.add_label("positive")
text_cat.add_label("negative")


losses = {}
for epoch in range(25):
    random.shuffle(train_data)
    # Create the batch generator with batch size = 8
    batches = minibatch(train_data, size=8)
    # Iterate through minibatches
    for batch in batches:
        texts, annotations = zip(*batch)

        example = []
        # Update the model with iterating each text
        for i in range(len(texts)):
            doc = nlp.make_doc(texts[i])
            example.append(Example.from_dict(doc, annotations[i]))

            # Update the model
        nlp.update(example, drop=0.5, losses=losses)
    print(losses)
```

spacy-train.py hosted with ❤ by GitHub view raw

Figure 1: Nandakumar's Code