

Fall 2021 ACM Coding Challenge

Logan Jackson

September 6, 2021

1 Original Approach

In the Python solution the approach was originally going to be just to take the entire input.txt file and feed it into a pre-trained transformer. But unfortunately that didn't work. So on the second try I decided to split up the contents of input.txt and pass parts of the file to the model one at a time. This is problematic due to the way Transformers work, Transformers are highly reliant on context, and due to this process of splitting up the file I am denying that context to the transformer thus having to process each string individually giving less accurate results.

The next problem we face with this approach is how to interpret the outputs in a way that is valid, the output of distilbert is the probability that the input matches a label this is basically the model's confidence in its prediction i.e. an output of 0.99 is very confident and an output of 0.5 is the model being unsure. Attached to the confidence is a label, or the prediction of whether the sentiment is positive or negative this is why the lower bound is 0.5 because after a probability of 0.5 it will just swap to predicting the other option. Throughout the input we see that the model fluctuates between positive and negative predictions. The way I get the overall sentiment of the input is by adding all positive predictions to all negative predictions, where positive predictions are positive and negative predictions are negative, the result will give me which predictions the model has more of.

2 Numbers

Using the approach I previously discussed we obtain the number 0.022 meaning that the model is tending slightly towards its positive predictions. Since each prediction will be positive or negative the sum could be very positive or very negative, actually there is no bound to how positive or negative a result can be therefore we need to squish the result, I do this using tanh if we have a input that is super, super positive say every sentence is 0.99 positive and there's 100 sentences that means score is 99 by squishing this value with a tanh function we are able to find that this same value is just extremely close to 1.0 this process of squishing is called normalization and is commonly used to make unbounded outputs make sense. Passing the sum of output probabilities, to the tanh function we get 0.022 this normalized score is the exact same as our previous score, well it's actually really close it's not exactly 0.022

it's more like 0.02199 but the score is very close to 0.022 so for convenience and rounding I will say that our normalized score is 0.022 based on this we can conclude that the sentiment of the input is slightly more positive than negative.

3 Potential Fixes

I think this approach is potentially problematic for highly context dependent inputs, but if you can embed the output of the previous batch with the input of the next this would solve the problem.

```
previous_context = None
for i in range(len(strs)):
    # second arg optional passing None doesn't do anything
    res = clf(strs[i], previous_context)
    previous_context = res[0]['score']
return res
```

```
$ res = {'POSITIVE', 0.899999}
```

4 VADER Approach and Numbers

The pretrained model thinks {'neg': 0.065, 'neu': 0.748, 'pos': 0.187, 'compound': 0.9982} this means 6.5% of the text has a negative valence score 74% has a neutral valence score and 18% has a positive valence score. the model seems to believe that the text is mostly neutral now the issue here lies in "compound score" on the vaderSentiment github page it says:

```
positive sentiment: compound score >= 0.05
neutral sentiment: (0.05 < compound score > -0.05)
negative sentiment: compound score <= -0.05
```

This is not representative of what compound score actually is while looking at the source code [1] you can find how the compound score is actually calculated, based on our output above it implies that the input is very positive. But based on the way that compound score is actually calculated. We see that this is not the case.

$$\frac{sum_s}{\sqrt{sum_s + \alpha}}, sum_s = \sum p_i n_i$$

the problem lies in the way that VADER works and how it handles neutral sentiment. Due to the way valence score works VADER will have the tendency to produce true neutral results, a word will have +4, then another word -4 [1] this is also another issue with VADER it's based on individual words predetermined sentiment score meaning that VADER doesn't adapt sentiment scores based on context this is problematic when dealing with larger inputs where sentiment may be highly varied and context reliant. So back to the issue of our compound score. Since the score is 0.99 you would expect the overall sentiment of the paper to be extremely positive but when looking at the break down of the percent of the paper that is positive negative and neutral, we can see that the paper is mostly labeled neutral and having more positive than negative so if anything the overall sentiment of the paper should be slightly positive but not extremely positive.

So you may be wondering how do I know that a score of 0.99 is extremely positive? this is due to the normalization function presented above what the normalization does is effectively "squish" the score to be between -1 and 1, where -1 is very negative and 1 being very positive. In theory this works by having each paragraph be labeled either positive or negative and if you take the sum of all positives and negatives then squish it to a scale based on the

hyper parameter α you should get the overall sentiment of the paper. The problem lies in the way that it handles neutral scores, because it's a sum neutral scores are entirely ignored so even if a paper is mostly neutral say 99% neutral and 1% positive the compound score will come out to be 0.99 an extremely positive score.

So what does this number actually represent if it doesn't represent the overall sentiment of the paper due to the fact that it doesn't take into account neutral scores, and how do we fix this? The first question of what it represents is pretty easy it represents the normalized sum of positive and negative sentiments across an input, once again this is invalid because it doesn't take into account neutral sentiments by virtue of it being a sum. Second how do we fix this? Well you can actually weight the neutral scores by averaging the valence scores rather than adding them together.

$$score_{avg} = \frac{sentiment}{numInputs}, compoundScore \frac{score_{avg}}{\sqrt{score_{avg} + \alpha}}$$

The reason we normalize this sum is because if a paper is extremely positive or extremely negative we will get an average score of > 1 or < -1 , thus why we have to normalize the score. This is the main reason why the nltk solution contains a fork of the vader source code because I had to modify the model in order to get a number that was representative of the overall sentiment of the input. After all of this the sentiment score is 0.025 or within ± 0.05 of 0.0 giving us a slightly positive but overall neutral score.

5 Transformer from Scratch Approach

Yeah so the model took to long to train.

6 Naive Bayes Approach

This is the solution that I was able to do in julia as implementing a transformer then training it was taking too long I decided to use Naive Bayes instead, this is built into a convenient library in julia called TextAnalysis. Naive Bayes uses statistical inference in order to determine the probability of an event given another event. There's an entire paper on why Naive Bayes seems to work so well for sentiment analysis but I'm not going to go too much into it here basically know that Naive Bayes is really really junk. Luckily our input seemed to have worked fairly well with the model I trained, although there are known issues with this model.

6.1 Data and Cleaning

For the training I'm using the SST2 dataset [] I produce a csv file for training with a python script called data.py that just appends the words to the sentiment scores. After that I read in the csv file to julia and do some magic.

```
function preprocess_sst2(df)
    df = df[:, ["body_text", "sentiment values"]]
    rename!(df, ["text", "sentiment"])
    sentiments = map((x) -> x >= 0.5 ? "negative" : "positive",
                    df.sentiment)
    return df.text, sentiments, unique(sentiments)
end
```

I used explore.jl to figure out what I needed to do to clean the data then used this function to get the things I needed for training the Naive Bayes model. There's another function in the file for cleaning a twitter dataset but the model was overwhelmingly negative so I decided to use SST2 instead. What's going on here is that I select the columns I want, the text and sentiment columns, then I rename the columns for easy access. Next I map over the sentiment values and label each as either positive or negative, we'll come back to this later. Lastly the function returns the text, sentiments (array of labels), and unique(sentiments). We can think of unique(sentiments) as being the A and B in, $P(A)$ and $P(B)$.

6.2 Training the Model

For training the model we have a simple function.

```
model = let
  nbc = NaiveBayesClassifier(uniques)
  for (text, label) in zip(texts, labels)
    sd = prepare_string_doc(text)
    fit!(nbc, text, label)
  end
  nbc
end
```

All the function is doing is taking in the possible events (A, B as mentioned before) then iterating over all the words and all the labels in our dataset.

6.3 Numbers

The output of the model is 0.995 what does this mean? This means the model believes there is a 99.5% probability the input is positive. How I got here here is that I summed all of the probabilities of positive and negative events similar to how VADER works except instead of a valence score it's a probability in a particular classification. So you may be wondering how I dealt with the neutrality problem what if a score is neutral say 0.5 which would be true neutral for this model well the answer is jank you just ignore it, instead of saying 0.5 was true neutral it's actually positive, and less than 0.5 is negative this allows us to sum positive and negative probabilities this model also relies on the idea that in any given input space there is $< \alpha$ true neutral sentiments, which is often true α in this case is representative of arbitrary statistical confidence in a result. After we sum the probabilities we get some crazy unbounded value that is representative of the probability that the sentiment is either positive or negative in an unbounded distribution. We normalize the value using tanh -1 being most negative and +1 being most positive. Our output is then the probability of either positive or negative in a normalized space. * I HAVE NO IDEA IF THIS IS TRUE

$$score = \tanh(\sum A_i B_i)$$

6.4 Naive Bayes vs VADER

You may be thinking what makes our score different from valence score in VADER and why it's okay for this model to have an output like 0.995 positive but it's not okay for VADER to have that same output. Do the problems with VADER simply not apply to this model? So yes the problems with VADER do not apply here the reason is our model is outputting a probability where as VADER's valence score is outputting the strength of a particular sentiment. Let's say a word is +4 another word is -3 then the valence score is 1 a max valence score is ± 15 this score normalized is 0.25 what this means is that the sentiment is decently positive. But with our model we take those two words as inputs and output the probability that it's positive and the probability that it's negative in this case we may get something like "positive" \rightarrow 0.99, "negative" \rightarrow 0.01, meaning that our model predicts that the sentence has a 99% chance of being positive.

7 Conclusion

The best solution was probably the transformer, but unfortunately I had no time. Based on the other results of Naive Bayes, VADER, and dilbert I can conclude that input.txt file is mostly neutral with a tendency towards positive sentiment. After reading the input myself I agree with the models the beginning of the file is neutral/positive then there's one part that could be negative near the beginning middle then after that the rest seems to be neutral/positive.

References

- [1] E.E. Hutto, C.J. & Gilbert. Vader source code.