Ian Laird & Andrew Walker

Natural Language Processing

May 8, 2019

**Final Project Report**

Our program starts out by finding the potential aspects of the reviews. To do this we first create a corpus of the reviews using the MyCorpusReader class created by the first undergraduate group. Next, we use the CorpusReader\_TFIDF from Program 1, trained on the Brown corpus to extract the TF-IDF vectors for each review. Using these values, we create an “average” TF-IDF value which we use to rank the words in decreasing order to extract the most important words. We consider the top 1% of the words although this is a value that could be adjusted to maximize performance.

These potential aspects are further filtered down by finding the most common bigrams in the review corpus. To do this we use the BigramCollocationFinder from NLTK. Then we use the bigrams to condense down our unigram potential aspects. For each bigram (w1,w2) if both w1 and w2 are unigram potential aspects we no longer consider w2 as a potential aspect. An example of why we do this is if the bigram (“battery”, “life”) is common and the unigrams “battery” and “life” are both potential aspects then we really only care about “battery”. We are making the assumption that if two aspects appear together frequently, they are probably both part of the same aspect.

All potential aspects that do not appear most commonly as nouns are eliminated. We do this by tagging every sentence in the review corpus using NLTK’s pos\_tag() method. We keep track of the count for each POS tag that the potential aspects appear as. We use these counts to create a probability that an aspect is a certain POS. We remove any aspect that isn’t most probable to be a noun. We chose to use the Universal Tagset for this problem to not have to manage the various types of nouns in the full tagset. The next step is to remove any aspects with a count greater than a certain threshold. This can be adjusted as needed but for this project we chose 3. We did this step since we found aspects to be things not discussed infrequently but also not discusses frequently. We found 3 to be a good threshold to capture the “middle ground” of the potential aspects which contained the aspects we care about.

Lastly, we remove any aspects that contain punctuation or numeric digits are removed. This step is mostly to remove potential aspects that are solely numeric or punctuation. We found “2” and “.” appearing commonly before this step was added. Once we return these aspects to the runner program from our AspectDetector, we consider only the top 20% of the potential aspects. This is another value that could be adjusted to maximize performance.

At this point the Wu-Palmer distance between the potential aspects and each of the other potential aspects is calculated using the method created by the second undergraduate group. All potential aspects with an average Wu Palmer distance below the cutoff point are eliminated. For this project we used 0.35 as the cutoff point, which was found using trial and error. We found that words with a high similarity are usually aspects we care about. For example, our highest similarity was between “keyboard” and “buttons”, two aspects we care about. This is a very naïve sense of clustering to find the most consistent aspects.

We then used the Stanford Dependency Parser from CoreNLP to parse the review corpus. We found every possible parse tree using the parser and we looked to see if the potential aspect ever appeared as the subject. If a potential aspect appears as the subject, we consider it to be a true aspect. We ran into issues with the parser being nondeterministic so through trial and error we found other situations in which we would want to keep the aspect. This made our final list of aspects more verbose but ensured we had many more of the aspects we care about.

The potential aspects remaining at this point are considered to be the aspects of the reviews. We search every sentence to see if it contains the aspect and if it does then we split the sentence by comma delimitation into “sentence chunks”. This is because we found that typically a sentence divided by comma had different aspects being discussed on either side of the comma or one section wasn’t relevant to the aspect in the other section. Each sentence chunk that contains an aspect is then analyzed for its sentiment. Sentiment detection is done using our SentimentAnalyzer class. This class uses a LinerSVC model from sk-learn to predict the sentiment of the sentence we pass it. The LinearSVC model is trained using corpra from IMBD designed for sentiment analysis training. We transform the sentence using a CountVectorizer that’s binary and considers ngrams of length 1 and 2. We keep a count of each aspect’s positive, negative and neutral sentiment. Last step in the runner program is to of course output each aspect with its sentiment.

Detailed instructions for how to get the project running are in the README.md file in the project. In order to get the project running you will need to install the sklearn and scipy libraries as well as install the Standford’s CoreNLP library. Instructions on how to do this are copied from our README.md below.

* Download CoreNLP at <https://stanfordnlp.github.io/CoreNLP/download.html>
* Navigate to where this is unzipped and run the following command
  + java -mx4g -cp "\*" edu.stanford.nlp.pipeline.StanfordCoreNLPServer -preload tokenize,ssplit,pos,lemma,ner,parse,depparse -status\_port 9000 -port 9000 -timeout 15000

Additionally, for subsequent runs of the same data you can create a data folder in the root directory of the project which will cache the LinearSVC model, the CountVectorizer, the initial list of potential aspects and a set of words to detect if the data changes.

When run against the sample data we correctly identify 7/9 aspects and of those 5/7 have the correct sentiment attached. The two aspects that we’re missing are “call quality” and “texting”. This is because these don’t explicitly appear in the text however these aspects do appear spread out throughout our list of aspects. For example, “call quality” can be broken into [“reception”, “call”, “signal”], all three of which we do find. Furthermore, if you combine the sentiments of these three aspects it comes out to [4,0,0] which is the correct sentiment for “call quality”. Our program was very good at extracting aspects directly from the text however inferring aspects, like what is needed for the aspects we miss, would require more work.

Our full project, including the training corpra and more verbose installation instructions can be found at our GitHub (<https://github.com/walker76/sentiment-analysis>)