Talha Mohammed
UHID: 2031877

In this paper, we will discuss how we built a sentiment search engine for an Amazon reviews dataset. We will discuss the background of this project and examine the work we are building upon in the field of Computer Science, particularly in the subfield of Natural Language Processing. We will also discuss what is novel, what problem we are trying to solve, and what our software engineering-based solution to it is. We will also analyze whether our solution to the problem is novel, whether it has been proposed or implemented before. Then, we will go into detail about the various methods we propose to build this sentiment-sensitive search engine. We will discuss its architecture, in a programmatic sense, and also the technologies we are using, such as external Python libraries and the like. Then we will discuss in detail our results, how we obtained these results, and we will discuss what tests we used to compare our baseline method to the methods we proposed for this software engineering problem. Furthermore, we will discuss what are methods actually entail, what they really mean. We will analyze whether our methods proved to be effective, or not, whether they were a tad bit better than the naive baseline or not. We will also discuss the underlying factors in the methods that led to these results. Then we will summarize our findings in the end, and give a conclusion on what we learned from this project, and what we could have improved.

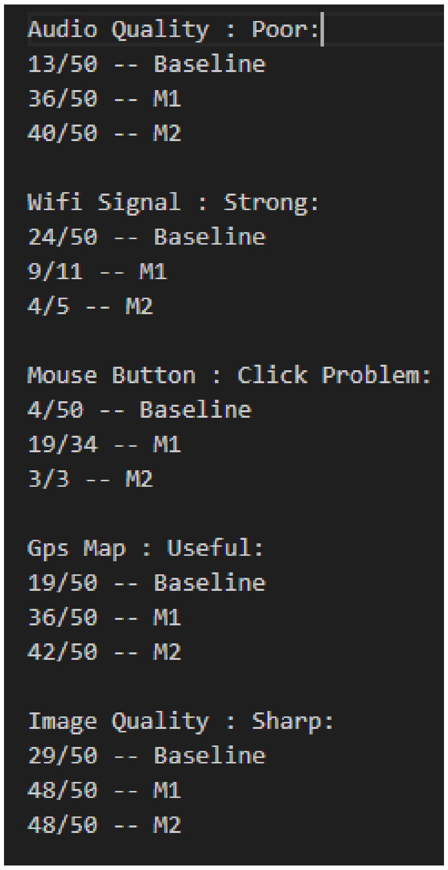
Firstly, as for the research we are building upon, I would say I am building upon the vast research and work that has been done in the field of Natural Language Processing. This is in general terms, but a few specific examples are indeed standout to me. One such example is how we have knowledge of a specific word being in its lemmatized form. For example, a word like “running”, in its lemmatized form, is simply “run”; this is especially useful for our case in processing reviews, so we can retrieve a review if it has a matching word, even if it doesn’t appear exactly like words in the query we search. Another thing we are building upon is how we also have knowledge of which words are stop words, which are basically words that are not semantically relevant, and removing them doesn’t really change the core meaning of the actual sentiment in the review. This is important because we want to decrease the number of words are model has to process. Going into more detail about my actual implementation, I use a baseline Boolean search and two other advanced methods. The Boolean baseline is the simplest form of review retrieval, as it just looks for matching words and then retrieves them. The next method I am using utilizes the actual sentiment of the words. So I basically utilized an opinion lexicon (thanks to [Hu and Liu, KDD, 2004]) that has all the positive words, and all the negative words, and checked if the opinion word describing the aspect is positive or negative, and filtered based on that. For my final method, I utilized the BERT LLM developed by Google, which analyzed the contextual meaning of the reviews and compared it to the query. Now, on to the problem we are trying to solve: we are trying to create a sentiment-sensitive search engine within an e-commerce review context. So not only are we paying attention to the words in the review and trying to match them with our query, but we are also looking at the sentiment expressed in the review. To the best of my knowledge, this type of search engine has not been implemented, not even by Amazon itself. So many search engines exist, but all of them just pay attention to matching words, rather than the sentiment itself in the review; that's the problem we are trying to solve. As for our approach, it is novel, this is because we are utilizing not only an opinion lexicon to check the sentiment expressed in the opinion words themselves, but we are also utilizing an LLM to analyze the contextual meaning expressed in the review and see if it's similar to the one expressed in our review. As stated, this hybrid approach of utilizing an LLM and an opinion lexicon is novel in this context.

As for the specific methods we are using, I will give a brief overview of how they work, and I will go into the specific architectural implementations in the next paragraph. The first method I implemented is the baseline Boolean search; all this does is pretty much check for matching words. As you can imagine, this is not very accurate, because it's not sentiment-sensitive. So our input will be, for example, “audio quality: poor”; this method will pretty much search for any reviews that satisfy the following boolean statement: “audio OR quality AND poor”. We can even just search for an aspect without any opinion, and it will retrieve all reviews containing that aspect. Furthermore, the broadest search we can do is by replacing the AND in the query we discussed above with an OR, so we will have reviews satisfying either or all of the Boolean statements. Our next method builds upon this, and this is an advanced, sentiment-sensitive method. As I alluded to earlier, it utilizes an opinion lexicon of positive and negative words, and it also checks the star rating the user gave to the product he/her is reviewing. So we pretty much perform a Boolean search like earlier to narrow our search down. The query will look like this: “audio AND quality AND poor”. So, as you can imagine, this is the narrowest Boolean query we can make, this is so we can only analyze the sentiment of the most probably relevant reviews. Then we check and see if the opinion is positive or negative. In this case, it's negative, so we only match reviews that have a star rating of below or equal to three. If it were a positive sentiment, we would only consider reviews that have a star rating of four or above equal to four. Our last and most advanced model is one in which we utilize a BERT LLM to calculate the cosine similarities between our query and the text of the review. Specifically, we are comparing the BERT embeddings of our query sentence and the review text. If it is above a certain threshold, let's say .70 in our case, we will add it to our return documents. Then we also apply the same filtering technique that we did in our previous advanced method, where we check the sentiment of the opinion term, then filter based on the star rating level. This makes our last model the most robust, as it combines the sentiment of the opinion with the BERT embedding similarity of the query with the review text.

So now we will speak on the specific architectural implementation of our methods, from the baseline to the most advanced model we have created. We will start with some of the data preprocessing that we do to ensure the data is in the correct format for our methods, then we will discuss specific ways we design our methods, from what is shared with all methods, to what makes them unique from each other. We will also discuss relevant technologies that we utilize in these methods. For starters, we designed our implementations in Python and used many important libraries to aid us in our data preprocessing, such as NLTK, which allows us to remove stop words from our raw review text, and then we utilize this library to also lemmatize the words into their root form. Another important library we use is Pandas. We use this for taking our raw data in the .pkl format to the Pandas Dataframe format, for ease of use in our methods. We then create a Python Set, in which we have a unique word in our dataset, which is our key, that links to the document IDs of the reviews that it is present in, so this is the value of our key. So like a unique word ⇒ to a set of document IDs. In total, we had about 153247 unique words! Just as a side note, since we are speaking about numbers, we have a total of about 2006 positive opinion words in our lexicon, and 4783 negative words in our lexicon. Now, on to the specific implementation of our baseline/advanced methods, for starters, our input will always be in the form of: “aspect1 aspect2”: “opinion1”, and only once we have a “opinion2”. So for all of our baseline/advanced methods, we take these as input, then we lemmatize them to ensure we can find them in our corpus word set. Then we have three operators, operator1, operator2, and operator3. This is relevant only to our baseline and first advanced method. Operator1 will always be OR, so we will have “aspect1 OR aspect2” in our query condition. Operator2 will always also be OR, if we have more than one opinion word, so for example: “opinion1 OR opinion2”. Then our third and final operator is pretty much tying up our aspect and opinion, so for example, we will have “aspect1 OR aspect2” AND “opinion1”. We will implement this query by using Python functions such as intersection(for AND) and union(for OR). Then we will return those document IDs that satisfy those conditions. That's our baseline. Now, for our first advanced method, we add another filtering technique. We check the sentiment of the opinion word. If we have two opinion words, we check the sentiment of the 2nd opinion word. So we call a function that checks if the opinion is in the positive or negative lexicon; if it's in neither, we give the word a neutral status. Now we run a for loop over all of the document IDS(the ones that satisfied our boolean checks), and we check if the sentiment of our opinion word is positive, and the rating of the review is greater than 3, then add it to our final return documents; if not, then skip it. In the reverse case, if our sentiment is negative, and the star rating is less than or equal to three, then we add it to our return documents. Lastly, if the sentiment word is neutral, we add it anyway to our return documents. This ensures that no irrelevant reviews are added. If my sentiment is negative, I only want negative reviews! We implement this filtering technique by reviewing sentiment in our 2nd advanced method as well, but only after our first initial filtering, which is our novel part. Thanks to Navid for providing the starter code and the embeddings of the entire sentence corpus. From our input query, we create a target sentence like: “The {aspect} is {opinion}”. We then calculate the BERT embedding of that sentence. As a side note, we are using the BERT model named “all-MiniLM-L6-v2”. Once we calculate the BERT embedding of our target sentence, we compare that embedding with all other sentence embeddings in the corpus to measure how similar the embeddings are. Our standard of measure is the cosine similarity measure. Then, if our similarity measure is above a certain threshold, let's say 0.7, in our case, we add it to our return documents. This is why our 2nd advanced method performs the best, incorporating an LLM to analyze the sentences. Let's get into the performance of our methods.

Query	Baseline (Boolean)			Method 1 (M1)			Method 2 (M2)		
	# Ret.	# Rel.	Prec.	# Ret.	# Rel.	Prec.	# Ret.	# Rel.	Prec.
audio quality:poor	1586	412	.2598	61	44	0.7213	122	98	0.8032
wifi signal:strong	214	103	.4813	11	9	.8181	5	4	0.8
mouse button:click problem	3363	270	.0803	34	19	.5588	3	3	1.0
gps map:useful	279	107	.3835	61	44	.7213	77	65	.8442
image quality:sharp	785	456	.5809	152	146	.9605	174	168	.9655

As you can see, our second advanced model performs the best, followed by our first advanced model, and then the worst-performing model is our baseline Boolean model. Since some of the retrieved reviews totaled in the thousands, reading all of them would take more time than doing the project itself! So I decided to take a subset of 50 reviews to read, and then scale it up to our total retrieved. I will show these scores below:



```
Audio Quality : Poor:
13/50 -- Baseline
36/50 -- M1
40/50 -- M2

Wifi Signal : Strong:
24/50 -- Baseline
9/11 -- M1
4/5 -- M2

Mouse Button : Click Problem:
4/50 -- Baseline
19/34 -- M1
3/3 -- M2

Gps Map : Useful:
19/50 -- Baseline
36/50 -- M1
42/50 -- M2

Image Quality : Sharp:
29/50 -- Baseline
48/50 -- M1
48/50 -- M2
```

I would also like to mention that I didn't implement a sorting functionality in my methods, so each time I would get a different order of the same document IDs. This is because I was using an unordered set data structure. To remedy this, I added sorting functionality to my methods and preserved the outputs that I used for my grading, the results you see in the table. They are in the "InitialGradingOutputs" folder in my "Outputs" folder. So what do these scores exactly mean? From our results, we can understand that simply Boolean-based retrieval, i.e., simply if a word, or group of words, is found in a review text, is not the best method for retrieving relevant reviews to the query of the user. It is the least performing method, this is because a person could be speaking in a tangent in his/her review, or talking about a counterpoint in their review text, like "Some say the audio quality is poor, but it's great!". Also, we are not considering the sentiment of the query, so that a great deal of reviews will not be relevant to the actual sentiment the query is expressing. Now, when we consider sentiment in our document retrieval, we see a huge upclimb in our performance, especially for simpler queries. In more complex queries like "mouse button: click problem", we still struggle to get documents relevant to the query. That's why our method one performs well, but struggles in the more complex queries. When we throw in BERT embeddings into the mix, we see our performance increase the most. We also add sentiment sensitivity to our last advanced model, as we did in our method one, so we build upon what worked for method one. This is because our LLM can capture more reviews, even if the exact words are not present in the review text. It does this by comparing the BERT embeddings of our target sentence to the sentences in the review. If they are similar, then we add them to our retrieval documents. If the BERT embeddings are similar between two sentences, this means that the two sentences are contextually similar. This is the central reason why our method two performs the best, because we capture the contextual meaning of the review by the BERT embeddings. So, not only does our method know the sentiment of the review, by the star-rating/opinion word, but our model also has a contextual understanding of our target sentence and our review text.

In conclusion, we have suggested a programmatic approach to tackling the issue of building a sentiment-sensitive search engine by using Python and its many Natural Language Processing libraries. We compared our solutions to a Boolean baseline and have demonstrated large performance gains in the relevant queries we retrieve. We started off by checking the sentiment of our query and then filtering reviews based on that. Then, we added in BERT embeddings into the mix, so we compare the review text and our query on a more contextual level, to see if they are similar. We saw the most performance gains with this, and our precision was always above 70% for all of the tests with this model. I believe that a solution that utilizes the sentiment of the query, with its BERT embedding, is indeed a robust solution to this research problem. More work needs to be done on this to advance it and reveal possible flaws.

Works Cited

Minqing Hu and Bing Liu. "Mining and Summarizing Customer Reviews." Proceedings of the ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD-2004), Aug 22-25, 2004, Seattle, Washington, USA,

Embeddings Starter Code + Dataset from Navid