Nehaa Balaji - 2199107

Amazon Search Engine Project

Introduction

The objective of this project was to create a review-based search engine for opinions expressed on Amazon using an opinion-based approach.

The minimum requirements of the project is the baselineBoolean search which finds matches to all keywords. However, boolean search does not understand "sentiment" nor "context". As such, I created a number of additional NLP-based methods to better attempt to recognize sentiment, aspect-opinion relationships, and context of a review.

The original dataset included over 210,000 reviews, but I limited myself to a smaller dataset of 10,000 reviews to help with processing time within Google Colab. I uploaded a csv file, switching it from the excel file that was provided so that the data can be read better on colab and then I switched my runtime type to run on A100 GPU for faster results. This project is done using jupyter notebook on colab, completely in python. I'm also importing all libraries or dependencies that were necessary to the project. After that, I begin preprocessing and follow up with each method which will be run through the five user-defined searches, and the results from each method will be compared based upon the number of reviews returned, and their relevance.

Ultimately, the final objective of the project was not only to develop several different retrieval methods; but to demonstrate the differences in behavior between each method as they attempted to retrieve both the "aspect" and "opinion" from the reviews.

Methods

I implemented all of the preprocessing steps directly in the notebook. The preprocessing steps involved cleaning every review by converting everything to lowercase, removing punctuation, removing stopwords based on an extended list of stopwords from the github link provided by the professor, also performing POS tagging, and finally lemmatizing each word using the WordNet lemmatizer. After cleaning the text, I added a new column called "clean_text" to store this cleaned version of the text. Every subsequent step utilized this cleaned version of the text. Next, I constructed an inverted index: posting lists that maps each token to the set of review IDs that include that token. This postings list structure allows efficient processing of Boolean queries.

The baseline system uses only Boolean searches. Query terms are cleaned via the same cleaning pipeline used to clean the dataset. All postings lists for each aspect token and each opinion token are searched on the inverse index. The baseline method provides results based on the intersection of the aspect set and the opinion set. Thus, a review will be returned only if it has at least one aspect term AND at least one opinion term. Method 1 also considers neither the rating nor sentiment nor proximity nor context; it only determines whether an aspect and opinion occur together in the cleaned review text.

Method 1 first conducts the identical Boolean baseline search which is 4.2 on the pdf instructions to provide the set of potential matches. Following this, I apply a rating-based polarity filter to the baseline set. Whether the opinion is positive or negative is determined using a very limited polarity dictionary (polarity). If the opinion is negative ("poor", "problem", etc.) then only reviews with a rating ≤ 3 are retained. If the opinion is positive ("strong", "useful",

"sharp") then only reviews with a rating > 3 are retained. Additional NLP, machine learning or proximity rules were not added to Method 1. The only purpose of this was to remove baseline matches that would contradict the overall rating and sentiment of the opinion word.

Method 2 uses Naive Bayes classifier, I made a new binary labeled column that would take reviews with a rating above 3 were assigned as a positive label; reviews with ratings of 3 or below would be given as a negative label. This produced 7,046 positive labeled reviews and 2,954 negative labeled reviews. I next performed an 80 / 20 split of the data for training and testing purposes which resulted in 8,000 training reviews and 2,000 testing reviews. I used a TF-IDF model to convert the text into vectors that I could use to classify the reviews. I chose to use a TF-IDF model and configure it to have 5,000 features, a minimum document frequency of 2, and a maximum document frequency of .95. Using those features, I then used a Multinomial Naive Bayes Classifier to classify the reviews. After the classifier had been trained, I was able to predict the sentiment label for the reviews in my dataset, and store those predicted sentiment labels in the pred_label column. When retrieving reviews that relate to specific aspect terms, I first collect all reviews that contain those terms, then filter that collection to only include the reviews whose predicted sentiment corresponds to the opinion polarity.

For Method 3, I implemented a sentence-based filtering technique. For each review, I have developed a process in which I first split the raw review into individual sentences using sent_tokenize and then perform the same cleaning on each sentence as has been performed for the remainder of this project. Next, inside each processed sentence, I identify the positions of the tokens containing the aspect terms and the opinion terms and check if any two of them fall within a ten word distance from each other. When a match is found, I add the review to the result set. If there is not a match for the review at the sentence level, Method 3 completes a second pass

through the clean review. This is done by identifying the positions of the aspect and opinion terms in the complete list of tokens and again checking for a ≤10-word window. Only those reviews which satisfy either of the above mentioned proximity checks are included.

For Method 4, I applied TF-IDF Cosine Similarity to the baseline output in Method 4 to find reviews with similar semantics as my queries. I wrote a new TF-IDF Vectorizer, limiting features to 5000, which was trained on all of the cleaned reviews, creating a 10,000 x 5,000 TF-IDF Matrix. For each query, the aspect/opinion terms were joined together in one string and cleaned using preprocess_text prior to being converted to a TF-IDF Query Vector. Using this TF-IDF Query Vector, the code identifies the Baseline Review ID's associated with the review TF-IDF Vectors, computes the Cosine Similarity between the TF-IDF Query Vector and the TF-IDF Review Vectors, and includes the Reviews where the Cosine Similarity Score > 0.1. This is done so that the Retrieved Reviews include both the Baseline Query Terms, and have very similar Semantics as the Query Content.

Results

The evaluation code ran all 5 test queries against the completed retrieval pipeline and saved a whole set of all program output and all generated files. All of the queries went through the Baseline, Method 1, Method 2, Method 3 and Method 4 functions in order. Each of the four functions returned a Python set of review ID's. For each query, the notebook took the size of each of the four sets by calling len() on them. The sets and their respective lengths were saved as structural dictionaries in the results_data list. When the last query was done with the results_data list, the dictionaries were translated into the comparison data frame at the end of the notebook.

The baseline function consistently produced the most count out of the deterministic methods because the function only checked if either the aspect tokens and/or the opinion tokens appeared anywhere in the cleaned text. The baseline function produced 68 review ID's for the query audio quality : poor; 10 for wifi signal : strong; 167 for mouse button : click problem; 11 for gps map : useful; and 46 for image quality : sharp. These are the same number of review ID's as there were intersections between the posting lists in the inverted index.

Method 1 re-used the baselines' review ID's and performed a rating-based filtering to determine which reviews had enough positive ratings using the rating_dict look-up previously established in the notebook. As such, since Method 1 removed only the ID's that did not meet the required rating polarity threshold, the output of Method 1 was less than the baseline output for all queries. The Method 1 results were 45 for audio quality : poor; 6 for wifi signal : strong; 57 for mouse button : click problem; 8 for gps map : useful; and 37 for image quality : sharp. These values represent how many ID's remained after the notebook iterated over the baseline ID's and determined the rating of each review.

Method 2 produced the largest number of reviews due to the fact that it used the classifier's predicted sentiment labels instead of matching terms. The function collected all reviews that included at least one aspect token directly from the postings dictionary. The function then filtered the reviews to include only those reviews whose predicted sentiment label matched the required polarity of the opinion term. Due to the fact that the classifier predicted so many positive labels compared to negative labels, Method 2 returned 85 results for audio quality : poor; 145 for wifi signal : strong; 56 for mouse button : click problem; 120 for gps map : useful; and 1051 for image quality : sharp. The number of reviews returned is directly related to the

number of items remaining in the filtered_results set after the classification-based condition was applied.

Method 3, which utilizes sentence tokenization and window-based proximity checking, produced smaller sets due to the fact that this method only accepts reviews where the aspect and opinion occur in the same sentence or within 10 words of each other. The implementation of this method includes looping over the sentences and full-review token sequence to collect only ID's where both tokens satisfied the spatial requirements. This method returned 56 results for audio quality : poor; 10 for wifi signal : strong; 80 for mouse button : click problem; 3 for gps map : useful; and 21 for image quality : sharp. The number of reviews returned by Method 3 can be explained by the strictness of the distance checks. Therefore, even though Method 3 returns fewer reviews than baseline, it is still possible to have both tokens appear in the same document.

Method 4 filtered the baseline results using a TF-IDF Cosine Similarity Filter to produce smaller, semantically coherent subsets of the baseline sets; each review was converted to a 5,000 feature TF-IDF Vector, compared to a query vector created from the aspect and opinion words; then, only those reviews with a similarity of .10 or greater remained in their respective sets. As a result, the number of results that satisfied this condition included: 39 results for audio quality : poor, 9 results for wifi signal : strong, 68 results for mouse button : click problem, 5 results for gps map : useful, and 29 results for image quality : sharp. The numbers represent how many results passed the similarity threshold applied to the TF-IDF Matrix, and all retained IDs were saved to the corresponding test4 output files.

In addition to returning the retrieval counts, the notebook also generated a full set of output files for each query and method. For each row in results_data, the system wrote the sorted review ID's to a text file in the format {query_name}_testX.txt, where X represents the method number. This resulted in a total of 25 text files being written. These text files are the actual review ID's used to populate the comparison table and will be manually evaluated for relevance.

```
# Display the full table with all columns in the requested format
comparison_df
```

```
...
```

```
--------------------------------------------------------------------------------------------------------------------------------------------------------------
COMPREHENSIVE RESULTS TABLE
--------------------------------------------------------------------------------------------------------------------------------------------------------------
```

	Query	Baseline Ret	Baseline Rel	Baseline Prec	M1 Ret	M1 Rel	M1 Prec	M2 Ret	M2 Rel	M2 Prec	M3 Ret	M3 Rel	M3 Prec	M4 Ret	M4 Rel	M4 Prec
0	['audio', 'quality'] : ['poor']	68	22	0.3	45	45	1.0	85	85	1.0	56	56	1.0	39	39	1.0
1	['wifi', 'signal'] : ['strong']	10	10	1.0	6	6	1.0	145	145	1.0	10	10	1.0	9	9	1.0
2	['mouse', 'button'] : ['click', 'problem']	167	167	1.0	57	57	1.0	56	56	1.0	80	80	1.0	68	68	1.0
3	['gps', 'map'] : ['useful']	11	11	1.0	8	8	1.0	120	120	1.0	3	3	1.0	5	5	1.0
4	['image', 'quality'] : ['sharp']	46	46	1.0	37	37	1.0	1051	1051	1.0	21	21	1.0	29	29	1.0

Next steps: Generate code with comparison_df | New interactive sheet