Topic Modeling and Analysis of Yelp Reviews using Predefined Topic List

Adit Jindal - ajindal40@gatech.edu
Akshay Iyer - aiyer89@gatech.edu
Preethi Narayan - pnarayan3@gatech.edu

Georgia Institute of Technology
CS 7650
Wei Xu

Abstract

Accurate unsupervised classification is a crucial task in today's world where large amounts of unlabeled data exist. Modern day approaches primarily focus on single class classification and might perform poorly in multiclass classification. Through the example of restaurant reviews, we introduce *Multi-Lbl2Vec,* a modified version of *Lbl2Vec* specifically for classifying unlabeled data based on predefined topic lists. The additions include a new *max_docs* variable, outlier detection, and a multiclass evaluation metric. On comparing this approach with industry standard approaches like *Lbl2Vec* and *BartMNLI,* we notice slightly better performance. To showcase the utility of *Multi-Lbl2Vec,* we target reviews of a particular business and run the classifier followed by a trained semantic analyzer. The results are analyzed to give recommendations to the business.

1 Introduction

1.1 Task and Motivation

In our project, we aim to classify unlabeled text documents into predefined topics. We noticed on Google and Yelp that reviews are often not segregated into different topics. An ability to sort reviews by certain topics would improve customer experience and help restaurants identify where they need improvements. Hence, we decided to build a model that will take in unlabeled Yelp reviews and determine whether the reviews relate to predefined topics like Food, Service, and Price. The motivation for this project is for companies or restaurants to be able to better understand which areas their reviews are written about. This is especially useful for analyzing negative reviews to understand where a company can improve its business. To extend this project further, we have employed semantic analysis on a group of reviews that belong to the same business so that we can better understand whether the general customer sentiment regarding each topic is positive or negative.
The scientific question we aim to investigate is "How well can an unsupervised learning method that utilizes predefined topics capture the trends and patterns of human-generated text?". This goal is relevant because a large portion of textual data available from the Internet is not labeled. The specific NLP task that we are performing for this project is using an embedding-based approach to retrieve unsupervised documents given predefined labels. In addition, we will also perform semantic analysis to determine the positive or negative sentiment of the human-generated text.

1.2 Past Work Done

The project we want to work on is closely related to the research paper Lbl2Vec: An Embedding-based Approach for Unsupervised Document Retrieval on Predefined Topics, published by Schopf et al (2021). The authors in their paper consider the problem of retrieving records to fit predefined topics from an unlabeled dataset. The lack of labeled data makes this situation difficult, as it pushes the research into the scope of unsupervised learning where finding an appropriate evaluation tool is challenging. The proposed methodology does a great job in classification and does not require much text processing like manual labeling of data points. It is able to increase F1 scores from 76.6 (an unsupervised classification baseline) to 82.7 for one dataset and from 61.0 to 75.1 for another.
The authors of the above-mentioned paper wanted to solve the common issue of document classification. The example they gave to motivate their research was of large numbers of news articles from sports newspapers and how to classify them based on the sport they talk about. The only source of information would be the content of the articles which do not have a predefined sports label. Conventional classification is not suitable here due to a) the high cost of labeling training data and making this a supervised learning problem and b) only wanting to classify the data on user-defined topics and ignoring other topics. Thus, the authors wanted to create a novel unsupervised approach that would be able to classify documents with high accuracy and without the need for high-cost labeling.

The approach defined was Lbl2Vec which allowed for the retrieval of wanted sports articles regarding basketball, soccer, and hockey without annotation of the corpus of documents. As described by the authors, Lbl2Vec works by creating jointly embedded word, document, and label vectors. The first step the authors take is defining a small number of keywords (keyword embeddings K) for each topic T and learning embeddings (label vectors) L from them. These keywords need to be related to the topic; for example, FIFA for soccer, and LeBron for Basketball. The second step is to create jointly embedded documents and word vectors. This is done by interleaving PV-DBOW (paragraph vector, distributed bag of words) and Skip-gram to get the embeddings in the same feature space. Thirdly, cosine similarity is employed to find the document embedding closest to the keyword embeddings of a topic. Next, label vectors for a topic are found. They are defined as the centroid of document vectors similar to the keywords of that topic. Moreover, the distance between the label vectors and document embedding measures semantic similarity. Hence, after the five steps of training, the label vectors of each topic are used to classify new documents using cosine similarity. Documents are classified as having the highest label vector and document vector similarity. Moreover, a threshold is defined which discards assignment if cosine similarity is less than it.

To evaluate their model, the authors compared classification results for common datasets - AG's Corpus and 20Newsgroups. They found that Lbl2Vec outperformed a state-of-the-art KE+LSA approach in all metrics. However, in comparison to a supervised Naïve Bayes approach, the metrics fell short, signaling that there is a tradeoff between the cost of labeling and accuracy.

1.3 Limitations

Although the paper focuses on approaches for classifying unlabeled datasets, the authors use two datasets that contain labels for evaluation purposes. The Yelp review dataset is unlabeled, and hence a novel approach would be needed for classifying it. Further, the paper shows the power of *Lbl2Vec* to find a single top label (single class classification). It would be interesting to see whether the same methodology can be utilized for multiclass labeling of documents where labels could be highly correlated. Our paper aims to find the answer to these questions.

2 Data and Methodology

2.1 Dataset

We utilize the open-source Yelp dataset from https://www.kaggle.com/datasets/yelp-dataset/yelp-dataset, specifically the yelp_ academic_dataset_review.json and yelp _academic_dataset_business.json files. In our primary analysis, we look deeper at the text column.

2.2 Classes and Keywords

To start with topic modeling, we needed to define the classes into which our model will segregate the reviews. We chose $k = 5$ classes for our initial study which were *Food, Time, Ambience, Service*, and *Price*. To identify them, a random sample of 35000 reviews were separated from the dataset. The reviews were first tokenized using *word_tokenize* from NLTK standard library, and then frequencies of each tokenized word were calculated. After ignoring pronouns, adjectives, prepositions, and adverbs, 5 words were identified having high frequencies: food (17530 times), place (16608 times), time (10396 times), service (9696 times), and $ (5792 times). Classes were chosen based on these five words as food, ambience, time, service, and price respectively.
After deciding the classes, the next step was to identify appropriate keywords per class. This was done by manually segregating common words from the separated reviews sorted based on frequencies. Words having frequencies above the 90th percentile (n > 70.0) of the dataset were chosen for this process. A hard upper bound of 20 words was decided upon to prevent one class from dominating the others. The final keywords used per class arranged in descending order of frequencies are:

Keywords per class	
Food	food, delicious, menu, chicken, fresh, cheese, sauce, eat, pizza, meal, salad, coffee, burger, hot, tasty, flavor, choices, yum, beer
Time	time, wait, room, table, 2, minutes, hour, long, weekends, busy, reservations, slow, crowded, rush, fast
Am-bience	place, experience, bar, area, location, clean, music, atmosphere, environment, patio, rooftop, seating, decor, lighting, vibe
Service	service, staff, friendly, people, happy, server, professional, hire, waiter, rude, waitstaff, attentive, tip
Price	$, price, worth, free, dollars, cost, expensive, money, cheap, overpriced, economical, luxury, reasonable

2.3 Manual Labeling

In order to analyze the results of the model's training and testing components, we decided to manually label the first 200 reviews of the dataset. More specific details about the comparisons between our model and the manually labeled reviews can be found in the evaluation section. We split up the first 200 reviews into different portions. For each review, we listed the top two themes from [Food, Time, Ambience, Service, Price]. We selected the top two themes based on human intuition and reading comprehension. We paid attention to keywords and specific aspects of the venue that each review mentioned. Then to address individual bias, each of us reviewed one other team member's reviews. This was our method of trying to capture the two most accurate labels for each review.

3 Models and Experimentation

3.1 Lbl2Vec Library

In our approach to segregate Yelp reviews, we first identified five classes and their associated keywords. The next step was creating multiple models and comparing their effectiveness.

The first model we used was *Lbl2Vec.* This model was introduced in *Lbl2Vec: An Embedding-based Approach for Unsupervised Document Retrieval on Predefined Topics, by Schopf et al (2021)* and provides a novel solution to the problem of retrieving records to fit predefined topics from an unlabeled dataset. To identify possible issues with currently used models and also have a baseline evaluation standard, we first employed *Lbl2Vec* in our dataset. We used the standard python library *Lbl2Vec* to do so. The first step was taking 100000 fresh reviews and preprocessing them which included tokenization, padding, and truncating to 120 characters. The following step was creating a document vector for each review. We utilized *Doc2Vec* (Document to vector) from *gensim.models.doc2vec* library to create vector representations of each review. These embeddings were then passed into the Lbl2Vec model as a parameter which outputs the similarity scores to the five classes and the most similar class label. The different parameters in the *Lbl2Vec* model are the *similarity_threshold* and the *min_num_docs*. The similarity threshold allows only documents with higher similarity scores than it to be counted for label vector creation while the *min_num_docs* states the minimum number of documents needed to calculate the label vector.

We use varied combinations of the above two parameters to test the implementation. As mentioned in the manual labeling section, we followed a set procedure to annotate 200 reviews. We then used our trained model to predict the output labels for each of them and calculate two F1 scores: i) F-1 score I: On the predicted most similar label and the first actual label ii) F-1 score II: On the predicted label with second highest similarity and the second actual label. Standard library sklearn.metrics is used to find the F1 score. Below is a table showing the various combinations of hyperparameters and their respective F1 scores:

similarity_threshold	**F1 score - I**	**F1 score - II**
0.2	0.475	0.165
0.3	0.48	0.195
0.4	**0.49**	**0.20**
0.5	0.42	0.14
0.6	0.45	0.225
0.7	0.44	0.18
0.8	0.325	0.16

Doc2Vec Trained on 30,000 reviews

similarity_threshold	**F1 score - I**	**F1 score - II**
0.2	0.495	0.17
0.3	**0.51**	**0.20**
0.4	0.47	0.225
0.5	0.445	0.225
0.6	0.45	0.21
0.7	0.445	0.16
0.8	0.40	0.20

Doc2Vec Trained on 70,000 reviews

min_num_docs was shown to be ineffective in changing the model's accuracy as the number of reviews per label were very large. The parameter would be more useful in cases where the number of reviews for a particular label is less in the dataset. Thus, the only variable we permute is the *similarity_threshold.* A trend we detected was that the F1 scores increased on increasing similarity threshold up to a point keeping other variables constant. Choosing a very low similarity threshold makes the learned label vectors generalized as almost all reviews are used in its computation. Similarly, choosing a too-high threshold makes the label vector very specific to certain reviews, decreasing the F1 score. Another trend was that with a greater number of docs used to train the doc2vec model, a small increase in the F1 scores could be produced. Hence, moving forward we decided to use the larger doc2vec model.
The maximum F1 scores we were able to get were (0.51, 0.20) by using 0.3 as the similarity threshold and

the doc2vec model trained on 70,000 reviews. However, we notice here that the F1 scores are low, which is in contrast to the expected performance of this model as stated by Schopf et al (2021). This is further seen through F1 score II of merely 0.2. Clearly, this model is unable to detect secondary topics of each review. On further analysis of the predictions, we notice that the model tends to generalize label vectors and predicts one label for most of the reviews. This is because reviews about restaurants mainly talk about topics like food and service, which are usually correlated. On training the standard *Lbl2Vec*, the label vectors might incorporate the majority of document vectors and their centroid (label vector) will have a high correlation to most of the reviews. Hence, that label will have the highest cosine similarity and will be outputted. In conclusion, the results point to the need for a different approach: Multiclass classification. In the next sections, we discuss certain solutions to the problems faced by *Lbl2Vec* on detecting more than one label through our model: *Multi-Lbl2Vec.*

3.2 Our Model: Multi-Lbl2Vec

In order to improve upon the standard *Lbl2Vec* library model, we implemented our own version, *Multi-Lbl2Vec*, which allowed for more customization of the model to fit the needs of our Yelp reviews dataset. As with the library model, the first step was to preprocess the data; we took 100,000 reviews and tokenized them. In the next step, we needed to create document embeddings for each of the reviews. We utilized the same Doc2Vec (trained on 70,000 reviews) used in the library *Lbl2Vec* implementation for a fair comparison of the models. The Doc2Vec model generated vector representations for each of the reviews.

Having created document embeddings for each of the reviews, the next step was to determine the label embeddings for each of the predefined topics. For each of the topics, we first took the associated keywords and found the corresponding word vectors by accessing the *word_vectors* attribute of the Doc2Vec model. Then, we generated a list of documents whose embeddings were most similar to the set of associated keyword vectors. Here we introduced a *max_docs* variable that allowed us to limit the number of similar documents for each label. Our thought process with introducing this variable is that certain frequent labels like *Food* and *Service* are similar to almost all documents, and as a result, their label embedding, which is a centroid of similar document embeddings, becomes centered among practically all the documents. We predicted that this may be a reason why the library *Lbl2Vec* model outputs labels like *Food* and *Service* for practically all the reviews. By using the *max_docs* variable to limit the number of documents that are used for the centroid calculation, we ensure that the centroid will be highly correlated to those documents that are most similar to the label.

We also implemented outlier detection using the Local Outlier Factor method (LOF). This makes sure that documents that are not sufficiently dense compared to *k* of their nearest neighbors are removed from consideration and hence do not play a part in label vector calculation. LOF was incorporated to limit the counting of reviews that are not primarily focused on a label, decreasing the generalization of label vectors.

After finding the documents whose embeddings are similar to the keyword embeddings, we computed the centroid of those document embeddings to determine the label embedding for each of the predefined topics. Finally, our model uses cosine similarity to the label embeddings to determine the class of any new documents. Another modification that we implemented is that our model outputs two labels for each review by selecting the two topics whose label embeddings have the highest cosine similarity with the document embedding. Our thought process here is that even if *Food* or *Service* is the primary topic of a review, now our model can provide further insight into an underlying secondary topic as well. We have accordingly adjusted the F1 score calculation to account for the model having two label outputs, and those adjustments will be covered in detail in the Evaluation section below.

3.3 Experimentation

In order to select the hyperparameter values of our *Multi-Lbl2Vec* model, we tested various combinations of hyperparameter values and compared the resulting F1 scores. Specifically, we were interested in determining the best values for the *similarity_threshold* and *max_docs* hyperparameters. The following tables display the F1 scores for various hyperparameter value combinations.

Similarity Threshold	Max Docs	F1 Scores
0.35	20	F1 Score for Ambience: 0.20512820780277252 F1 Score for Food: 0.6857142448425293 F1 Score for Service: 0.2666666805744171 F1 Score for Price: 0.5106383562088013 F1 Score for Time: 0.5271317958831787 Macro F1 Score : tensor(0.4391)
0.45	20	F1 Score for Ambience: 0.20512820780277252 F1 Score for Food: 0.6857142448425293 F1 Score for Service: 0.2666666805744171 F1 Score for Price: 0.5106383562088013 F1 Score for Time: 0.5271317958831787 Macro F1 Score : tensor(0.4391)
0.55	20	F1 Score for Ambience: 0.581818163394928 F1 Score for Food: 0.6993007063865662 F1 Score for Service: 0.15094338357448578 F1 Score for Price: 0.0 F1 Score for Time: 0.5413534045219421 Macro F1 Score : tensor(0.3947)

From the table above, we learned that a similarity threshold between 0.35 and 0.45 provided best results:

Similarity Threshold	Max Docs	F1-Score (Max)
0.35	500	0.4321
0.35	100	0.4355
0.35	20	0.4391

From the table above, we learned that a lower *max_docs* value, such as 20, provided the best results.

We noticed here that even when the macro F1 score showed improvement, the F1 scores for one or two of the labels were drastically lower compared to the other classes. To improve this, we implemented a custom outlier remover. We ran additional experiments to determine the best *number_of_nearest_neighbors* to use in our outlier remover function. The table below displays the F1 scores for various *number_of_nearest_neighbors* and compares them to the baseline F1 score without outlier removal. From the table, we determined that using an outlier remover improves the performance of our model. We selected a *similarity_threshold* of 0.35, a *max_docs* value of 5, and 2 *nearest_neighbors* for the outlier remover as the hyperparameter values for future iterations on the model.

Similarity Threshold	**Max Docs**	**Outlier Detection: Number of Nearest Neighbors**	**F1 Scores**
0.35	20	OFF (baseline)	F1 Score for Ambience: 0.205128 F1 Score for Food: 0.6857142448 F1 Score for Service: 0.2666666 F1 Score for Price: 0.510638356 F1 Score for Time: 0.5271317958 Macro F1 Score : tensor(0.4391)
0.35	20	10	F1 Score for Ambience: 0.22222 F1 Score for Food: 0.671532869 F1 Score for Service: 0.266666 F1 Score for Price: 0.53061223 F1 Score for Time: 0.541353404 Macro F1 Score : tensor(0.4465
0.35	5	2	F1 Score for Ambience: 0.2857142 F1 Score for Food: 0.62992125749 F1 Score for Service: 0.81218272 F1 Score for Price: 0.4938271641 F1 Score for Time: 0.54237288236 Macro F1 Score : tensor(0.5528)

3.4 Evaluation

Given that our methodology is a type of unsupervised learning, we still wanted to have a good way to evaluate the model. We were especially curious to know how well the model's output matches human intuition. We wrote two separate evaluation functions: The first function we wrote is meant for training and the second function calculates an F1 score to serve as our testing metrics. The first function determines the average similarity between the top two predicted labels and each of the training reviews. It also calculates the average similarity for all the non-matching labels. Our calculations for similarity were done using Pytorch's cosine similarity tool. The embeddings for the labels as well as the reviews were calculated via our model and passed into both the evaluation functions. Our first function also outputs all the predicted labels. In general, we were able to find that when training, the model frequently guesses one of the labels correctly, but often mistakes the other label. We were able to see this qualitatively when we mapped the outputted indices back to the corresponding string labels and compared the model's output with our own verified training reviews.
So, within training, one thing to improve on is to create a greater level of separation between the similarity scores of the matching and non-matching labels. Though our project falls under the scope of unsupervised learning we wanted to correct inconsistencies in training with the goal of achieving a higher F1 Score in testing. The way we went about doing this was to correct the label vectors from the initial Doc2Vec model in the training phase. For example, say one of the true labels for a review was "Ambience", but Ambience was not one of the two outputted vectors. In this case, we would set the new label vector for Ambience equal to a weighted sum of its current value plus the vector representation for the current review. Then we would repeat this process for any misidentified labels in training. In doing so, we were able to see incremental improvements in the test F1 score of about 2 percent. This shows how semi-supervised learning can further improve the score.
The second function that we wrote computes the F1 score using the last 100 manually annotated reviews. For each label we computed the number of true positive, false positive, and false negative instances. Then we calculated the precision, recall, and F1 scores. Since we were comparing two manually annotated labels with two labels from the output, we defined a custom False Positive/False Negative metric. For example, suppose the ground truth labels for a particular review are [Food, Service] and our model outputs [Time, Service] as the most likely labels. In this case, Service would receive one point for True Positive, Time would receive 0.5 points for False Positive, and Food would receive 0.5 points for False Negative. We believe this scoring system of 0.5 for mismatches would accurately capture the model's performance while not penalizing too harshly. Using

this above approach, we obtained the following F1 Scores:

```
F1 Score for Ambience: 0.2857142686843872
F1 Score for Food: 0.6299212574958801
F1 Score for Service: 0.8121827244758606
F1 Score for Price: 0.4938271641731262
F1 Score for Time: 0.5423728823661804

Macro F1 Score : tensor(0.5528)
```

From these results, we determined that the model performed well in describing the "Food" and "Service" labels but lacked in performance in the other labels, especially ambience. To try and improve these results, we combined the model's outputs with another model, bart-large-mnli (see 3.4), and in doing so saw overall improvement:

```
F1 Score for Ambience: 0.48275861144065857
F1 Score for Food: 0.7555556297302246
F1 Score for Service: 0.7810651063919067
F1 Score for Price: 0.42424243688583374
F1 Score for Time: 0.5

Macro F1 Score : tensor(0.5887)
```

When combining these models, we essentially used the output of MNLI when a threshold probability was exceeded for either the Ambience or Food labels. Otherwise, we used the output from our own custom implementation.

3.5 Pretrained Model

One pre-trained model which we found to be very relevant to our project was bart-large-mnli developed by Facebook. This model is described as a "ready-made zero-shot sequence classifier (6)." In practice, this means that given a passage of text and user-provided labels, it will generate scores between 0 and 1 for each label based on how closely it matches the input text. By default, the sum of the scores for each label will sum up to 1, so the scores can easily be translated into probabilities. Given that we are defining five key labels, we felt it was important to analyze the model and juxtapose its results with our own *Lbl2Vec* implementation. When we ran bart-large-mnli on the portion of the dataset deemed as test data, we found that the model overall performed strongly on identifying if ambience or food was a key component of a review. This is evidenced by the high F1 scores for both categories. To elaborate, the F1 Score given for Ambience was 0.7017 and the F1 Score for Food was 0.933. Yet the model struggled to identify if service, price, or time were relevant to a review. In fact, bart-large-mnli seemed to consistently rank scores for these three labels below the scores for ambience or food when run on our set of test reviews.

3.6 Model Comparisons

Here are the best F1 scores we were able to get for each model after parameter tuning:

Library Lbl2vec:
Macro F1-Score: 0.6433

Multi-lbl2vec:
Macro F1-Score: 0.5887

Pretrained:
Macro F1-Score: 0.327

4 Use Case

4.1 Semantic Analysis Model

Once we developed an effective method for classifying unlabeled reviews into categories, we wanted to gain a better understanding of what those reviews indicated about their respective topics. Thus, we implemented a semantic analysis model that can detect whether a given review expresses positive or negative sentiment.
To build the semantic analysis model, we first needed to obtain the data to train and test our model. We used an unseen portion of 50,000 reviews from our Yelp reviews dataset to train and test our model. The dataset has a 'stars' attribute that indicates the number of stars out of 5 that the person who wrote the review assigned to the review. We considered all reviews with 4 or more stars as positive reviews and all reviews with 3 or fewer stars as negative reviews in order to determine the labels for our dataset.
In order to gain the best semantic meaning from the reviews, we cleaned the reviews by tokenizing the reviews and removing all non-alphanumeric characters. We then normalized the clean reviews using stemming, removing stop words, and finally vectorizing the reviews.
Then, we needed to select the model to train. Based on our research, several sources indicated that ensemble techniques, like random forest classifiers, tend to perform well on sentiment analysis tasks (Gonçalves). Thus, we decided to implement our semantic analysis model by training a random forest classifier from the *scikit-learn* library.
We ran experiments during the training process to determine which train-test split would result in the best semantic analysis model performance. We tested three different holdout values (0.2, 0.3, and 0.4) and measured the accuracy and F1 scores of the trained model on an unseen validation set. The table below shows the accuracy and F1 scores for the models trained using each holdout value.

Holdout Value	Accuracy	F1-Score
0.2	0.792	0.864

0.3	0.784	0.859
0.4	0.782	0.847

We were also interested in how the train-test split would affect the time it takes to process the data and train the model. We tested the same three holdout values (0.2, 0.3, and 0.4) and recorded the time consumed by data processing and training for each. The table below shows the data processing times for the models trained using each holdout value.

Holdout Value	Time Consumed for Data Processing
0.2	0.0168 s
0.3	0.0144 s
0.4	0.0135 s

Our experiments show that the optimal holdout value is 0.2, where 20 percent of the data is held separately for validating the model while the remaining 80 percent of the data is used to train the model. The experiments show that the semantic analysis model has greater accuracy when the holdout value is 0.2 and the decrease in speed is very small. Thus, we used a 0.2 holdout value for training our semantic analysis model for the remainder of this project.

Our semantic analysis model takes in a group of reviews and determines positive or negative sentiment for each. Since our goal with this project is to use Yelp reviews to provide businesses with more insight into what is working well and what can be improved, we wanted to develop a way of aggregating the sentiment information in a useful way. Thus, in addition to classifying the sentiment of each review, our model also outputs a percentage of reviews with positive sentiment.

4.2 Example

Having successfully trained both the review segregation model and the semantic analysis model, we can use them on a certain subsection of reviews to show the importance and power of classifying unsupervised data like Yelp Reviews.

A use case for our model is a complete end-2-end system for businesses that have a list of their customer reviews. Businesses can benefit from various analyses of reviews as they can pinpoint weaknesses and take necessary steps to fix them. By initially adding another model like *Multi-Lbl2Vec* to segregate reviews into categories, better in-depth analysis can be performed. Here, we show an example:

We first identify a business to target and then perform segregation and semantic analysis to give recommendations for improving the average customer rating. The business we chose was *Baileys' Range,* a hamburger restaurant in Saint Louis. The restaurant was found in the subset of reviews we have been using of size 100000. Restaurants were first arranged in descending order of the number of reviews (R) followed by the average star rating (S) calculation for each. The chosen criteria were R > 300 and S < 3.8 as these bounds would give an ample number of reviews to analyze as well as enough margin to improve star ratings in the future. *Baileys' Range* had R = 327 and S = 3.74 in our subset of reviews.

After choosing the target business, the reviews specific to that business were collected and passed through *Multi-Lbl2Vec*. The model returned the top 2 labels for each review, which allowed us to segregate the reviews into each of the 5 chosen categories: *Food, Time, Ambience, Service,* and *Price.* Then for each of the five categories, the semantic analysis model was run which showed the percentage of positive reviews and the percentage of negative reviews per class.

4.2.1 Results

The total number of reviews for the chosen restaurant was 327. On running *Multi-Lbl2Vec* and the semantic analyzer*,* the reviews were classified as shown in the table below:

Topic	Number of reviews	% Total reviews	% Negative reviews
Ambience	49	14.98	20.4
Food	141	43.12	17.7
Service	251	76.76	16.3
Price	130	39.76	16.9
Time	83	25.38	14.5

4.2.2 Analysis and Recommendation

From the outputs, one can notice that *ambience* has the highest percentage of negative reviews (~1 in 5). Moreover, the number of reviews targeting *ambience* is only 49 (~15% of the total reviews). Thus, the ambience of the restaurant is probably not extraordinary for customers to remember it. At the same time, certain customers have something negative to say about it. Hence, a recommendation for *Baileys' Range* is to work on improving its ambience by reading the negatively classified reviews to pinpoint weaknesses. This would have a two-fold effect on rating: negative reviews will decrease, and positive reviews might increase. Another interesting trend is that *time* has the least number of negative reviews along with 25% of the total reviews. Seeing this data point, one recommendation could be for targeted advertisements showing speedy cooking times and low waiting. This would appeal to people who enjoy fast food like college students and will boost reviews for the restaurant. Clearly, many more such recommendations can be derived from analyzing the above data. This example shows the power of *Multi-Lbl2Vec* when used together with other models and is one among its many possible use cases.

5 Conclusions and Future Work

In this project, we presented a novel end-to-end solution starting with an analysis of reviews and culminating in a recommender system for a particular restaurant via sentiment analysis. Overall, we have demonstrated the promise of using unsupervised learning in Natural Language Processing by showing how applying different vectorization metrics and combining models together can lead to steady improvements in our F1-Score. Given the overall nuance of the project, the variance of writing patterns across reviews as well as the time/hardware constraints we encountered, we consider our project a valid proof of concept for further unsupervised topic modeling/analysis.

Moving forward, we can take steps to expand upon our current progress. The evaluation diagrams demonstrate that our model's F1 score is just under 60%. We can look into potential ensemble methods where we combine the results of our *Multi-Lbl2Vec* model with another type of model to improve the overall F1 Score. Another possibility is to look at keyword extraction/summarization techniques to infer the label in training. Further, we noticed that increasing the Doc2Vec training size from 30,000 to 70,000 saw an improvement in the F1 score. Hence, by using more powerful GPUs, we can potentially train Doc2Vec on millions of reviews to get better vector representations. This should have a positive effect on the F1 score. Further, for evaluation we test on 200 reviews. For a better understanding of the model's efficiency, a larger annotated dataset would be required. Another area for exploration is whether we can take the semantic analysis one step further; instead of just identifying sentiment, we can look into summarizing groups of reviews. Thus, instead of simply showing how positive the general sentiment within a review category is, perhaps we could provide recommendations based on the ideas that appear in the reviews repeatedly.

6 References

1) Schopf, T.; Braun, D. and Matthes, F. (2021). Lbl2Vec: An Embedding-based Approach for Unsupervised Document Retrieval on Predefined Topics. In Proceedings of the 17th International Conference on Web Information Systems and Technologies - WEBIST, ISBN 978-989-758-536-4; ISSN 2184-3252, pages 124-132. DOI: 10.5220/0010710300003058
2) https://www.scitepress.org/Link.aspx?doi=10.5220/0010710300003058
3) Chang, M.-W., Ratinov, L.-A., Roth, D., and Srikumar, V. (2008). Importance of semantic representation: Dataless classification. In Proceedings of the Twenty-Third AAAI Conference on Artificial Intelligence, pages 830–835. https://www.aaai.org/Papers/AAAI/2008/AAAI08-132.pdf
4) Pollyanna Gonçalves, Matheus Araújo, Fabrício Benevenuto, and Meeyoung Cha. (2013). Comparing and combining sentiment analysis methods. *Association for Computing Machinery,* New York, NY, USA, 27–38. DOI: https://doi.org/10.1145/2512938.2512951
5) https://github.com/sebischair/Lbl2Vec
6) https://huggingface.co/facebook/bart-large-mnli?candidateLabels=urgent%2C+not+urgent%2C+phone%2C+tablet%2C+computer&multiClass=false&text=I+have+a+problem+with+my+iphone+that+needs+to+be+resolved+asap%21%21