Report on

A Hotel Recommendation System By Text Mining of Reviews

CSC 869 – Term Project

Submitted by,

Girish Tiwale (Student ID: 917941705),

Spring 2018

To,

Prof. Hui Yang,

Department of Computer Science,

San Francisco State University

Problem Statement

One of the tedious task of your trip can be the search for the perfect hotel. Not every time, people travel to the familiar places. Sometimes, people travel to places they have never been before. Keeping the large number of hotels opening nowadays, finding the hotel of your choice can be overwhelming.

My approach follows opinion mining of the hotel reviews to tackle this problem, giving the suggestion of the best suited hotels for the user.

Datasets

1. **Text mining**

The dataset [1] I used is available on Kaggle for everyone. The dataset is slightly different from the one on the website. I had personally contacted the researcher and got the dataset because, the dataset available on Kaggle doesn't have punctuations and it is crucial for this project that the dataset is properly punctuated.

The updated dataset contains over 571K reviews of 1490 luxurious hotels in Europe.

The dataset has various attributes like:

Additional_Numbers: No. Of Ratings given Without Giving Review

Address: Hotel Address

Date: Review Date

General_Score: Review Post Date

Hotel: Hotel Name

Name: Reviewer Name

Nationality: Reviewer Nationality

Negative: Negative Review

Negative_Counts: No. Of negative words

NumberReviews: Hotel's total reviews

Positive: Positive Review

Positive_Count: No. Of positive words

Review_Num – the total reviews the user gave

Score: Rating the user gave to the hotel

Scraping_Date: Date the data was scraped

Tags: Tags/keywords

2. **Sentiment Analysis**

For training the fastText text classifier, I used Labelled Dataset of Amazon Reviews [2] from Kaggle. This dataset has positive and negative labelled reviews from Amazon. It is publicly available dataset.

Program Compiling Instructions

Dependencies: The project needs sner (Stanford NER) [3], pandas, gensim, nltk, scikit_learn, vaderSentiment [5] & fastText [4] (not fasttext, because the functions are different in the modules). The dependencies are also specified in the requirements.txt file in the root directory. To install all the dependencies, you can run the following commands from the root folder or you can install all the modules separately (The module versions are critical). (**NOTE:** fastText should be installed separately as the module version difference may cause errors. The following commands clones the GitHub repository and installs the module).

```
pip install -r requirements.txt

./install_fast_text.sh
```

Downloading Instructions (NOTE: directory paths are critical):

1. Download the Stanford-ner-2014-06-16 from following link extract it. After extracting, put the inside folder(The folder will have nested folders after extracting. Put the inside folder to the project root folder.) into the project root directory. The folder should be named as 'stanford-ner-2014-06-16'

https://nlp.stanford.edu/software/stanford-ner-2014-06-16.zip

2. Download the datasets (both train and test). Extract them and put the .txt files under {root}/fastText_classifier/

https://www.kaggle.com/bittlingmayer/amazonreviews/data

Running the programs:

1. Separate the datasets

```
python3 separate_datasets.py
```

2. NER Tagger Training 1st Iteration

```
java -cp ./stanford-ner-2014-06-16/stanford-ner.jar edu.stanford.nlp.ie.crf.CRFClassifier -prop ./NER-Tagger/hotel.prop
```

3. Testing NER Tagger

```
java -cp ./stanford-ner-2014-06-16/stanford-ner.jar edu.stanford.nlp.ie.crf.CRFClassifier -loadClassifier ./NER-Tagger/hotel-ner-model.ser.gz -testFile ./test_files/test_file.tsv
```

4. Partition Dataset Hotelwise

```
python3 partition_dataset_hotelwise.py
```

5. Remove Stopwords and removing POS other than nouns by POS Tagging

```
python3 remove_stopwords.py
```

6. Run Apriori makefile

```
make -C ./apriori/apriori/src
```

7. Find Associations using apriori

```
./apriori/apriori/src/apriori -s0.176018531m2n2  ./text_files/file_without_stopwords.txt ./training_files/associations.out
```

8. Analyze associations and extend the training dataset

```
python3 analyze_associations.py
```

9. NER Tagger Training 2nd Iteration

```
java -cp ./stanford-ner-2014-06-16/stanford-ner.jar edu.stanford.nlp.ie.crf.CRFClassifier -prop ./NER-Tagger/hotel_2.prop
```

10. Testing NER Tagger

```
java -cp ./stanford-ner-2014-06-16/stanford-ner.jar edu.stanford.nlp.ie.crf.CRFClassifier -loadClassifier ./NER-Tagger/hotel-2-ner-model.ser.gz -testFile ./test_files/test_file.tsv
```

11. Word Embeddings

```
python3 word_embeddings.py
```

12. Analyze embeddings and extend the training dataset

```
python3 analyze_embeddings.py
```

13. NER Tagger Training 3rd Iteration

```
java -cp ./stanford-ner-2014-06-16/stanford-ner.jar edu.stanford.nlp.ie.crf.CRFClassifier -prop ./NER-Tagger/hotel_3.prop
```

14. Testing NER Tagger

```
java -cp ./stanford-ner-2014-06-16/stanford-ner.jar edu.stanford.nlp.ie.crf.CRFClassifier -loadClassifier ./NER-Tagger/hotel-3-ner-model.ser.gz -testFile ./test_files/test_file.tsv
```

15. Train FastText Classifier

```
python3 fast_text.py
```

16. Run the Stanford NLP Server (Keep it running in different Terminal for all the next steps) (All the next steps use Stanford NLP Server for extra performance)

```
java -cp ./stanford-ner-2014-06-16/stanford-ner.jar edu.stanford.nlp.ie.NERServer -port 9199 -loadClassifier ./NER-Tagger/hotel-2-ner-model.ser.gz
```

17. Extract Sentences, Tag Entities, Create Hotels_Features matrix with VADER Sentiment

```
python3 extract_sentences_vader.py
```

18. Extract Sentences, Tag Entities, Create Hotels_Features matrix with FastText Text Classification

```
python3 extract_sentences_fasttext.py
```

19. Run the recommender (Uses VADER and basic approach)

(Preferences as integers from 0 to 5) *Run without arguments for usage instructions

```
python3 recommender_VADER_Basic.py <arguments>
```

Eg. python3 recommender_VADER_Basic.py 0 2 3 1

20. Recommender – VADER & Cosine-similarity (*Run without arguments for usage instructions)

```
python3 recommender_VADER_cosine.py <arguments>
```

21. Recommender – FastText & basic approach (*Run without arguments for usage instructions)

```
python3 recommender_FastText_Basic.py <arguments>
```

22. Recommender – FastText & Cosine-similarity (*Run without arguments for usage instructions)

```
python3 recommender_FastText_cosine.py <arguments>
```

23. Evaluate VADER_Basic approach

```
python3 test_VADER_Basic.py
```

24. Evaluate VADER_Cosine Similarity approach

```
python3 test_VADER_cosine.py
```

25. Evaluate FastText_Basic approach

```
python3 test_FastText_Basic.py
```

26. Evaluate FastText_Cosine Similarity approach

```
python3 test_FastText_cosine.py
```

Main strategies

1. Separate the dataset

Firstly, I separated the dataset into three different parts. One for training the Named Entity Recognizer, second to extract features from the reviews and third to evaluate the recommender system.

2. Train NE Recognizer

The goal was to extract the words that are in correspondence with the features of the hotels i.e. 'ROOM', 'LOCATION', 'SERVICE', 'FACILITY', 'MEAL', 'VALUE'. There are a lot of Open Source Named Entity Recognizers readily available to use but they have their pre-defined Entity Classes defined. For my task, I needed a Named Entity Recognizer that can tag the entities into one of the above 6 classes.

So, for this task, I decided to train a Named Entity Recognizer with the help of Stanford NLP[6] library on my own data. For that, I needed training data. So, I used the first of my three separated datasets for this task. I tokenized the dataset and manually tagged the entities in the training data and trained a custom class Named Entity Recognizer. For testing this recognizer, I have already created one .tsv file that has their true labels in the file. But, the recognizer gave me the f1-score as low as 0.5405.

3. Improving NE Recognizer

It is practically impossible for a human being to tag every token. But, if a word often comes in the same sentence with another word, that means the two words are related to each other. To make use of this property, I decided to further mine the association rules from my original dataset. For that, I extracted the sentences from the original dataset and stored in different files with respect to the hotel names. I also had to do some preprocessing before mining the associations. The preprocessing consisted on removing the stop words and punctuations and tokenizing the sentences.

Another important preprocessing step I followed was Part of Speech Tagging. The only words that should be tagged should be nouns. So, along with the stop words removal, I also removed the parts of speech other than nouns and then this was the input to the association pattern mining algorithm.

a. Associations -

I used apriori [7] algorithm's implementation to find the association rules from the dataset. I used a support threshold of 0.176018531 (associations that appear in at least 2500 sentences) and searched for

the associations of two words that might be similar to each other. I added these new words to the training file in following way:

If both the words are already present in the training dataset, then it is of no use adding those to training dataset again. If one of the word is present, then I added the other word under the same Entity Class. I retrained the Named Entity Classifier over the newly added training data and got f1-score of 0.5789.

b. Word Embeddings using word2vec

Another way of finding the related words is by using the word embeddings. It represents the words in an n-dimensional space. The property of the word embeddings is that the words that are closest to each other are related to each other one way or the another (i.e. they occur in the same sentence more often than the others). I used word2vec module of gensim [8] to represent the word embeddings and added one word that is most similar to the words that are already in the training dataset and trained the Named Entity Recognizer for the third time. This approach increased the f1-score to 0.6316.

c. Comparison between the results

	Precision	**Recall**	**F1-score**
Manual Tagging	0.667	0.4545	0.5405
Adding Associations Mining	0.6875	0.5	0.5789
Adding Word Embeddings	0.75	0.5455	0.6316

4. Sentiment Analysis

The basic idea was to tag the Entities and using sentiment analysis to extract the opinion from the review sentences. For each tagged token to a feature, the Hotel's feature score was increased according to the sentiment of that sentence in which the token was present. For sentiment analysis, I used two different approaches. One was Sentence polarity calculation using VADER and the other was text classification approach using FastText.

a. VADER [5]

VADER provides us the sentence polarity in the form of compound, positive and negative polarity value. According to that, I scored the hotels based on the compound polarity value. If the compound value was between (0.5 & -0.5), then I termed that sentence as neutral and gave the hotel a score of 3 out of 5 for that particular feature. If the compound value was between (0.5 & 0.8), then I termed that sentence as slightly positive and gave a score of 4 out of 5. If the compound value was greater than 0.8 then I termed the sentence as strong positive and gave a score of 5 out of 5 and vice versa for negative sentiments (scores of 1 & 2).

b. FastText [4]

FastText provides text classification approach and provides the class predictions along with the probabilities score of each class. I needed a huge Hotel Reviews data that is labelled with five different classes to train a five-class classifier with FastText. It was difficult to get the exact data that I wanted. So, I used the publicly available Amazon reviews data that was tagged with 2 classes (positive and negative) and I used the class probabilities to score the hotels for their features. If the difference between the class probabilities was less than 0.5, then I termed the sentence as neutral and gave a score of 3 out of 5. If the difference was between (0.5 & 0.8), then according to the class prediction I gave the negative classified sentence a score of 2 and positive classified sentence a score of 4. Similarly, if the difference was more than 0.8, then a positive classified sentence was given a score of 5 and the negative classified sentence was given a score of 1.

7. Build a Hotels_Features Matrix

Using the Sentiment Analysis, I scored the hotels on a scale of 1-5 w.r.t their features, 'LOCATION', 'MEAL', 'VALUE', 'FACILITY', 'SERVICE', 'ROOM'. To remove the bias of the algorithm towards the hotels that has more reviews, I normalized the scores by taking the averages of those scores and I got a matrix of Hotels and the scores for their 6 features.

8. Recommender System

For recommending the hotels to a user, the recommender system needs the preference list from the user. The preference list should consist of the 6 hotel features. For recommending the hotels to the user, I followed 2 different approaches:

a. Regular Approach

This approach collects the hotels' scores for only those features that are present in the preference list and builds a dictionary. According to the scores of the hotels, considering the user's preference list, the system returns top 10 hotels to the user. The drawback of this approach is that this approach does not consider the features that are not in the preference list of the user.

b. Cosine Similarity

The second approach I used is cosine similarity. After I get the scores of each hotel w.r.t each feature, I created a 6-dimensional space with 6 dimensions being the features of the hotels. Then according to user's preference list, I restrict the hotel list in following way:

1) I restrict the hotels list to only those hotels that has a score of 4 or more w.r.t. the features in the user's preference list. So, there is no way that the user will get a bad hotel recommendation.

2) Then, I find the best hotel from the list i.e., the hotel with the best average score.

3) Then, I sort the hotels by their cosine similarity w.r.t. the best hotel and return the results.

Evaluation Strategy and Main Results

As far as the recommendation systems are concerned, it is really tough to get a well labelled dataset for the evaluation purpose. The dataset I have, has ratings that the users gave to the hotels. So, using that rating, I am evaluating my recommendation system.

The strategy I followed for evaluation:

I had already separated about 115 user's data from the original data set and kept that for testing. I extracted only the reviews and ratings from the dataset. The fact that the user particularly reviewed about the hotel features that means the user prefers those features. Using the reviews, I extracted the Entities and termed those features as the preference list of the user and passed them to the recommender system. The ideal recommendation system should give the results in the following manner:

a. If the user gave the rating 10 to the hotel, then it must be one of the perfect hotels for him. So, if his preference was passed to the recommender system, this hotel should be in the top 20 recommendations list.

b. If the user gave the rating greater than 9 to the hotel, then it may not be one of the perfect hotels for him, but he must have liked it. So, if his preference was passed to the recommender system, this hotel should be in the top 50 recommendations list.
c. If the user gave the rating greater than 8 to the hotel, then it may not be as good a hotel for him, but he didn't hate it. So, if his preference was passed to the recommender system, this hotel should be in the top 100 recommendations list.
d. But, if the user gave the rating less than 7 to the hotel, then it cannot be a good a hotel for him as he hated it. So, if his preference was passed to the recommender system, this hotel should not be in the top 200 recommendations list.

Following are the results of my evaluation process:

	Rating == 10	Rating > 9	Rating > 8	Rating < 7
VADER_Basic	0.0345	0.0667	0.1154	1.0
VADER_Cosine	0.0345	0.0833	0.1538	0.9412
FastText_Basic	0.0000	0.0667	0.0897	1.0
FastText_Cosine	0.0000	0.0667	0.1282	1.0

Result Analysis:

The low accuracy scores in the first three columns gives us the idea about correctness of the approach.

Another reason of low accuracy is that the dataset has 1490 hotels all over the Europe. There is no possibility that the user will chose a hotel in another city or country. There might be a better hotel in another city or a country, but he is never going to stay there if the hotel is in another city or country. The user gave review for that hotel means he lived in the hotel. But, recommending hotels from all over the Europe is slightly wrong. The user's location should be considered while recommending a hotel.

But the high accuracy in the fourth column gives us the idea that this recommendation hardly gives bad recommendation. The users who gave a bad review to the hotel, the recommendation system hardly recommended that hotel while evaluation.

Domain specific polarity calculation is a challenge. Some sentences might seem neutral but the meaning can be very different w.r.t the domain. Additionally, the

Pros and Cons

Pros:

1. The recommender never returns a bad recommendation.
2. Good Named Entity Recognition.

Cons:

1. Because of the uncertainty in the natural language and people's way of writing the review make the sentiment analysis of reviews a challenging task.
2. Need more accurate domain specific polarity calculation.
3. The way people write reviews or incomplete sentences.

Conclusion and Future Scope

Due to the uncertainty of natural language processing and the unstructured way of people's writing make the text mining task a very challenging one. Another conclusive point is that Sentiment Analysis is also a domain specific problem. A neutral statement in regular language might be a positive or negative in another domain. To get the best results, you must have the perfect training dataset.

To further better this task of Hotel Recommendation using Text Mining of Reviews, following approaches might be handy:

1. User to User Similarity: I f we can calculate the user to user similarity, then we can use it to give better recommendation to the user based on the other users that have same taste.
2. Use of Hotel Address so that only nearby hotels can be searched for recommendation

References

[1] https://www.kaggle.com/jiashenliu/515k-hotel-reviews-data-in-europe

[2] https://www.kaggle.com/bittlingmayer/amazonreviews

[3] https://nlp.stanford.edu/software/CRF-NER.shtml

[4] https://github.com/facebookresearch/fastText

[5] https://github.com/cjhutto/vaderSentiment

[6] https://nlp.stanford.edu/

[7] http://www.borgelt.net//doc/apriori/apriori.html

[8] https://radimrehurek.com/gensim/