ILS - Z534

Search

Project Report

Yelp Dataset Challenge

Fall 2017

Indiana University Bloomington

Jivitesh Poojary
Prathamesh Jangam
Shreyas Rewagad
Vighnesh Nayak

December 12, 2017

Abstract

Yelp provides an online application where crowd-sourced reviews about local businesses are shared. The yelp dataset provided to us contains the information pertaining to businesses such as ratings, operation timings, business categories, location, check in information, photos, user information, reviews and tips posted by users and more. This dataset helps the business owners to improve services and users to seek more information about businesses they intend to visit.

In this project, for the first task, we intend to create a restaurant recommendation system for users. We have used the content based recommendation approach as well as collaborative filtering approach for making the recommendations. For creating the matrix for collaborative filtering we have used a sentiment analysis model as well as review star average model. We compare item-item collaborative filtering with the user-user collaborative filtering, our results match with the theory.

In the second task, we aim to generate a description of the business using user reviews. We have used the Lexrank algorithm for obtaining the most relevant sentences from the reviews and have to order them to obtain a meaningful description.

Github Repo: https://github.iu.edu/srewagad/srewagad-jpoojary-vnayak-pjangam-information-retrieval

Project Contribution

Jivitesh Poojary	Code - Task 1 - CBR - Data Extraction, CF - Pearson & Cosine similarity, CF - Mean Review Rating technique, Presentation, Documentation.
Prathamesh Jangam	Code - Lexical Centrality, Graph generation, Evaluation
Shreyas Rewagad	Code - Task 1 - Entire workflow - CBR & CF, CF - Sentiment Analysis and Model Evaluation, Github repo, Documentation.
Vighnesh Nayak	Code - continuous Lex Rank, IDF modified cosine.

Table of Contents

TASK 1: Business Recommendation System

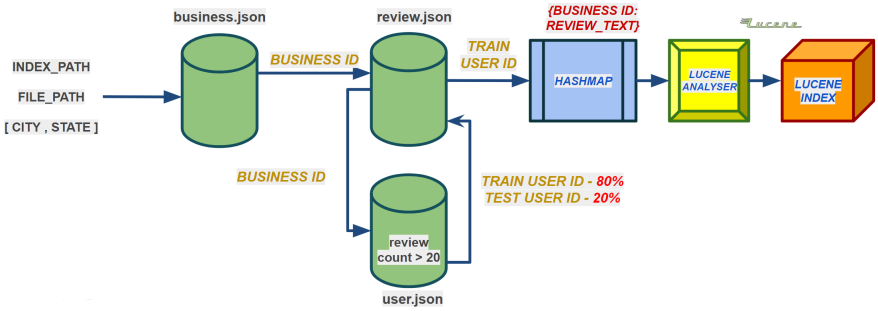
This is system is the core of the business model at Yelp. It helps in improving the user experience by proving relevant suggestions of restaurants. The most common way of implementing this using a content based recommendation system or a collaborative filtering approach.

TASK 1: Algorithms

Content based recommendation

In this approach we use Lucene to generate an index using user reviews. Later we take a sample of these users and try to predict the business score for a particular user using their reviews. Finally the model is evaluated using TREC EVAL.

Generating Index in Lucene



$$\text{tfidf}(t,d,D) = \text{tf}(t,d)\cdot\text{idf}(t,D)$$
$$\text{tf}(t,d) = 0.5 + 0.5\cdot\frac{f_{t,d}}{\max\{f_{t',d} : t' \in d\}}$$
$$\text{idf}(t,D) = \log\frac{N}{|\{d\in D : t\in d\}|}$$

TF IDF

$$\text{score}(D,Q) = \sum_{i=1}^{n}\text{IDF}(q_i)\cdot\frac{f(q_i,D)\cdot(k_1+1)}{f(q_i,D)+k_1\cdot\left(1-b+b\cdot\frac{|D|}{\text{avgdl}}\right)},$$
$$\text{IDF}(q_i) = \log\frac{N-n(q_i)+0.5}{n(q_i)+0.5},$$

BM25

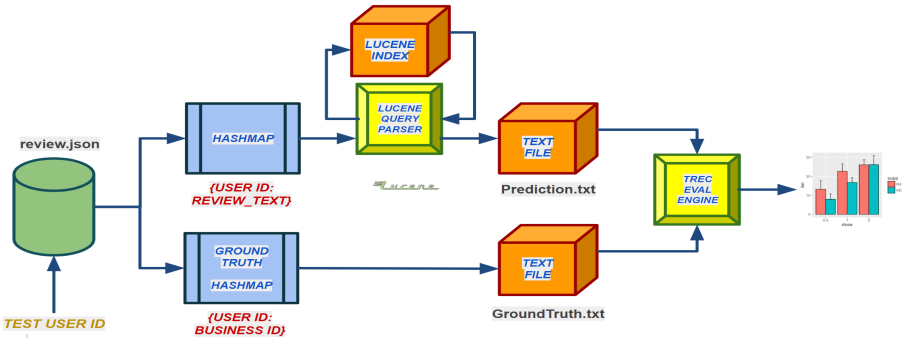
$$P_\beta(w\mid\hat{\theta}) = (1-\beta)\frac{c(w,D)}{|D|} + \beta P(w\mid C)$$

JELINEK-MERCER SMOOTHING

$$P_\mu(w\mid\hat{\theta}) = \frac{c(w,D)+\mu P(w\mid C)}{|D|+\mu} = \frac{|D|}{|D|+\mu}\cdot\frac{c(w,D)}{|D|} + \frac{\mu}{\mu+|D|}\cdot P(w\mid C)$$

DIRICHLET PRIOR SMOOTHING

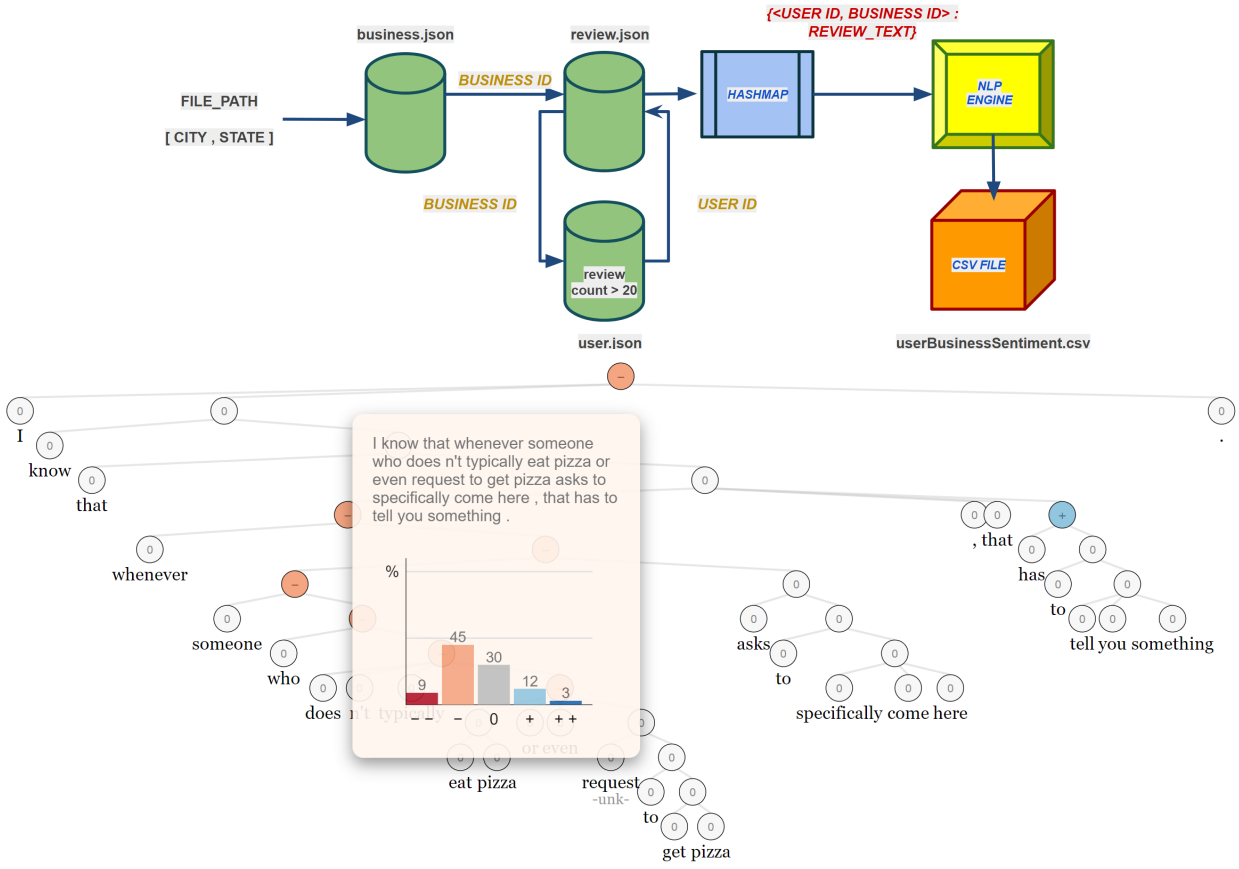
Prediction and Evaluation



Memory based Collaborative Filtering

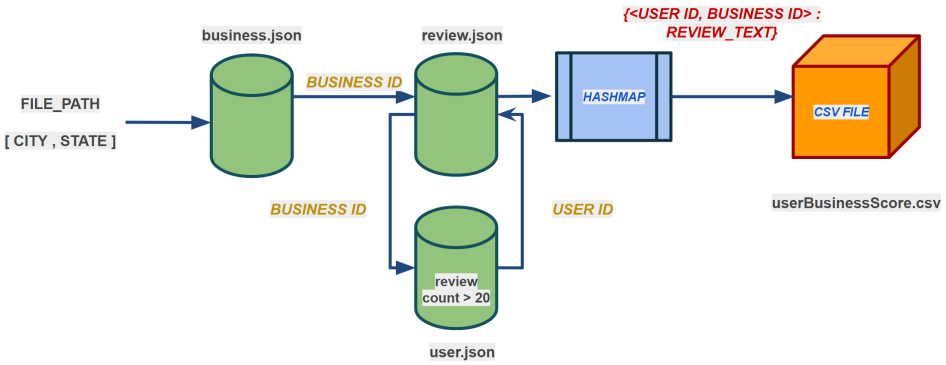
Matrix Generation using Sentiment Analysis

We considered this approach to see if there was any bias involved in user ratings. Some user would be more generous in providing ratings while some would be too stingy. Our hypothesis was that the sentiment score would in some way normalize this bias.



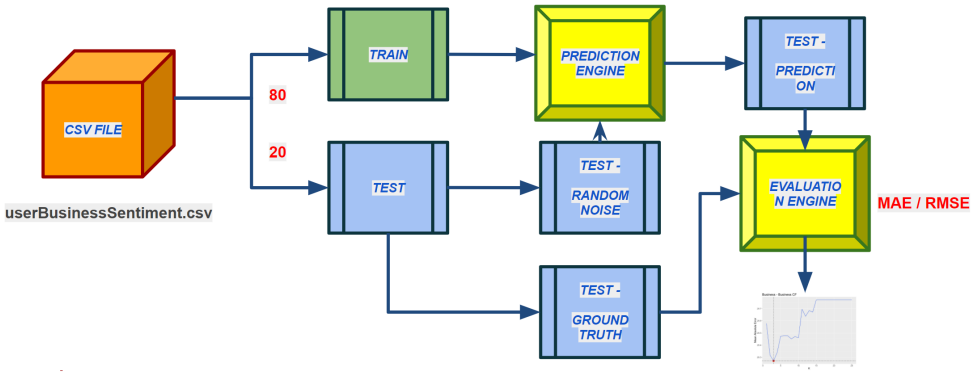
Matrix Generation using Average Rating Score

This approach of Matrix generation is the straight forward method of averaging the review star score provided by user for each business.



Finding the Optimal K value for Collaborative Filtering

This form of collaborative filtering works like K-Nearest Neighbours and requires K value to be one of its parameters. For prediction we have used implicit feedback for Item to Item CF and weighted sum for User to User CF.



$$w(u,v) = cos(\boldsymbol{u},\boldsymbol{v}) = \frac{\boldsymbol{u}\cdot\boldsymbol{v}}{||\boldsymbol{u}|| * ||\boldsymbol{v}||} = \frac{\sum_{i\in I} r_{ui} r_{vi}}{\sqrt{\sum_{i\in I} r_{ui}^2}\sqrt{\sum_{i\in I} r_{vi}^2}}$$ COSINE

$$w(u,v) = \frac{Cov(u,v)}{\sigma_u \sigma_v} = \frac{\sum_{i\in I}(r_{ui}-\bar{r}_u)(r_{vi}-\bar{r}_v)}{\sqrt{\sum_{i\in I}(r_{ui}-\bar{r}_u)^2}\sqrt{\sum_{i\in I}(r_{vi}-\bar{r}_v)^2}}$$ PEARSON

SIMILARITY MEASURE

USER

$$P_{ai} = \frac{\sum_{u\in U^k} \mathbb{1}(r_{ui})\cdot w_{au}}{\sum_{u\in U^k} w_{au}}$$

$$\mathbb{1}(r_{ui}) = \begin{cases} 1 & \text{if } u \text{ has listened to } i \\ 0 & \text{if } u \text{ has not listened to } i \end{cases}$$

ITEM

$$P_{ai} = \frac{\sum_{n\in N} r_{un} w_{in}}{\sum_{n\in N} |w_{in}|}$$

PREDICTION

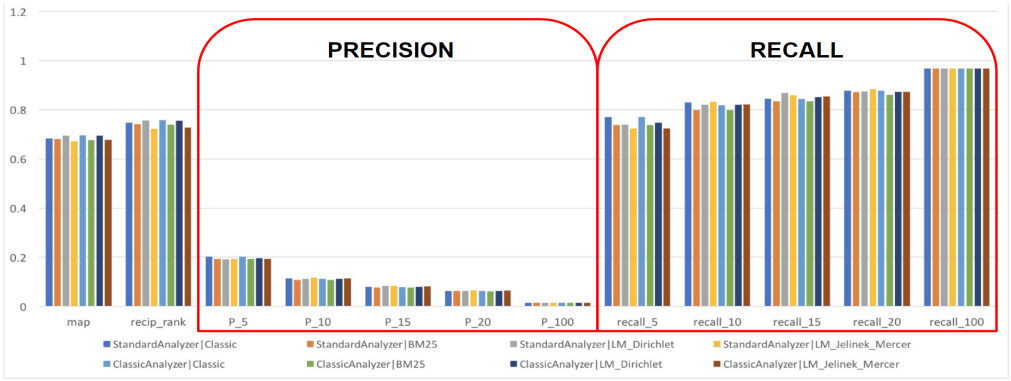
$$MAE = \sqrt{\frac{1}{|\mathcal{T}|}\sum_{(u,i)\in\mathcal{T}} |\hat{r}_{ui} - r_{ui}|}$$

$$RMSE = \sqrt{\frac{1}{|\mathcal{T}|}\sum_{(u,i)\in\mathcal{T}} (\hat{r}_{ui} - r_{ui})^2}$$

ERROR CALCULATION

TASK 1: Evaluation

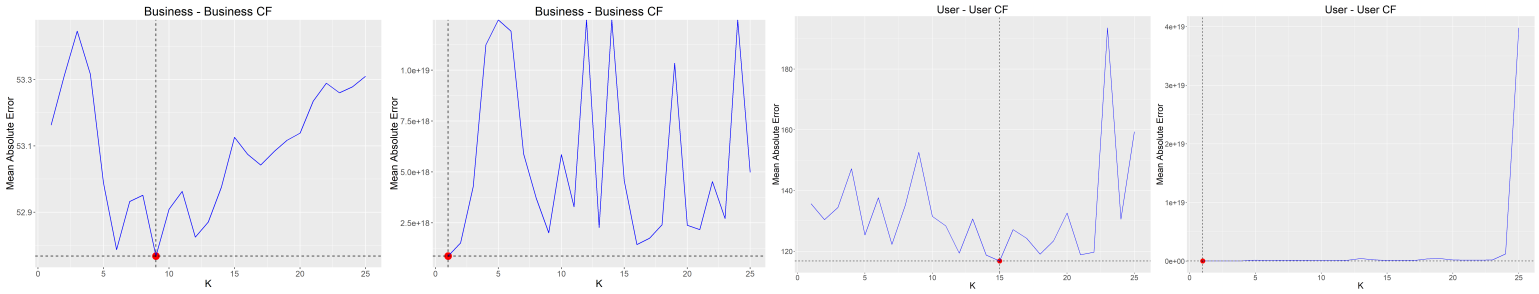
Content Based Recommendation



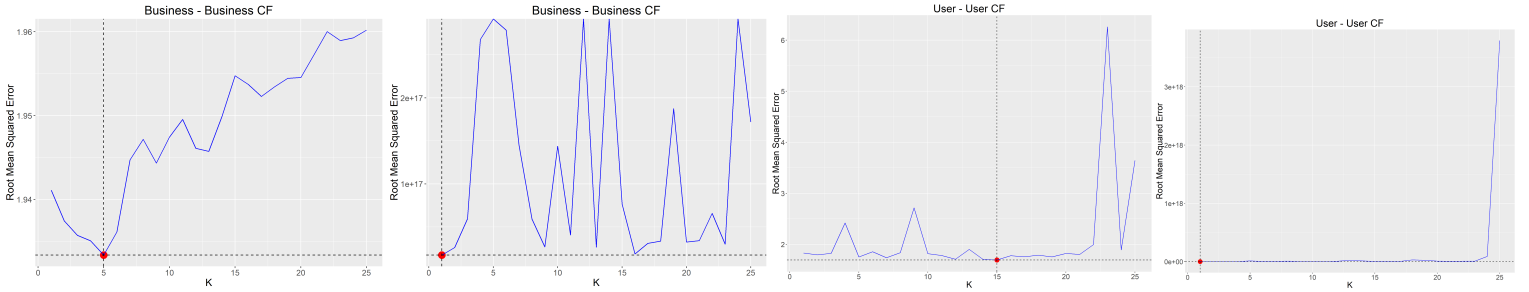
- As expected there is a linear trend in precision and recall value with varying top K values.
- We observe that the Content Based Recommendation approach had recall values in the 95% and the combination of Standard Analyzer with the Language model of Jelkin Mercer Smoothing performing consistently well.

Memory Based Collaborative Filtering

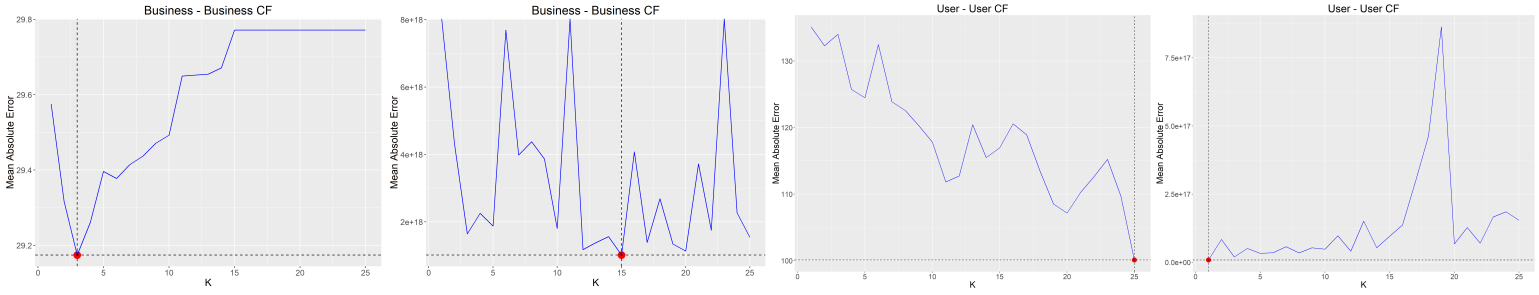
- The approach we take here is, iterate the models for k values from 1 to 25.
- Consider the K value which gives the least value of Mean Absolute Error and Root Mean Square Error for each model. Use this K value to calculate the model statistics.
- We implemented this approach to get a good bound on the model statistics.
- The Cosine similarity based approaches are relatively smoother compared to Pearson Similarity based ones
- Average Review Score: Mean Absolute Error (Cosine - Pearson)



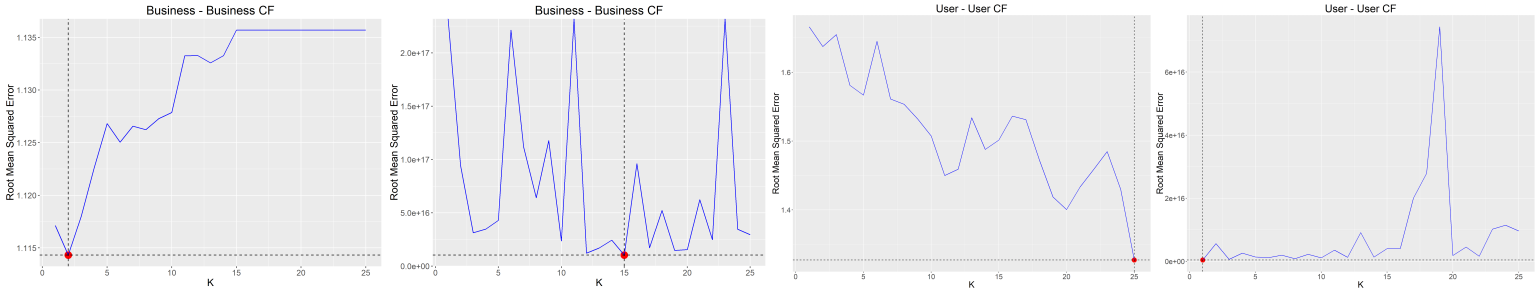
Average Review Score: Root Mean Square Error (Cosine - Pearson)



Sentiment Analysis: Mean Absolute Error (Cosine - Pearson)



Sentiment Analysis : Root Mean Square Error (Cosine - Pearson)



	Average Review Score				Sentiment Analysis			
	Item to Item		User to User		Item to Item		User to User	
	Cosine	Pearson	Cosine	Pearson	Cosine	Pearson	Cosine	Pearson
Accuracy	0.9746	0.4735	0.937	0.5831	0.9745	0.3772	0.9407	0.4134
Precision	0	0.02414	0.02255	0.051014	0.039473	0.02345	0.03478	0.05011
Recall	0	0.5222	0.007462	0.58278	0.003984	0.63213	0.00104	0.61458
Sensitivity	0.9990	0.47232	0.983736	0.59039	0.997684	0.37107	0.98594	0.40362
Specificity	0.0000	0.52224	0.007463	0.43781	0.003984	0.63214	0.01042	0.61458
Detection Rate	0.9746	0.46080	0.936629	0.56212	0.974403	0.36241	0.94022	0.38490
Detection Prevalence	0.9990	0.47245	0.984157	0.58904	0.997645	0.37099	0.98611	0.40278
Balanced Accuracy	0.4995	0.49728	0.495599	0.51410	0.500834	0.50160	0.49818	0.50910

- We see that Cosine similarity based approaches have a higher accuracy, but lower precision and recall values. This probably means that there are many False Negatives and False Positives in the algorithms.
- We also see that sentiment analysis approach has a higher precision and recall values than the review ratings approach we developed.

Comparing Content based recommendation and Memory Based Collaborative Filtering

- Content based similarity approach in our case seems to be performing better for precision and recall values. We attribute this to the much optimized lucene code base as compared to the basic implementation of Memory based collaborative filtering.
- The content based recommender had recall values in the 90% range while for the collaborative filtering technique we had recall values in the 60% range.

TASK 2: Review Summarization Using Lexrank.

The goal of this task to generate a summary of all the reviews of a business. We plan to generate a subset of five sentences from a set of reviews. Summarizing reviews helps owners understand the key points affecting their business. It also enables the users to get a short overview of the restaurant and its key factors. We used Lexrank to calculate sentence importance by representing sentences as nodes in a graph. We implemented Lexrank using lexical centrality as well as continuous Lexrank. The results show that Lexrank using threshold and lexical centrality tend to perform better than continuous Lexrank.

TASK 2: Algorithm Design

- ### Preprocessing

  In this step we first extracted the top fifty restaurants with the most reviews. All the reviews of each restaurant was then grouped together to form a single file. Every file had every sentence stored as a document. In our algorithm we summarize reviews of different restaurants independently.

- ### Indexing

  In this step, we index our file using Lucene. The indexing helps get the TermFrequency vector of any sentence and IDF of any word in our file.

- ### Inverse Document Frequency

  In this algorithm, we need to measure the importance of each word. Thus we calculated the Inverse Document Frequency of every word in our file. The inverse document frequency of any word can be defined as .

  $$\mathbf{idf}_i = \log\left(\frac{N}{n_i}\right)$$

  N – Total number of documents/Total number of sentences
  N_i – Total number of sentences containing the word.
  Thus, the IDF for frequent words will tend to zero.

- ### Graph Creation

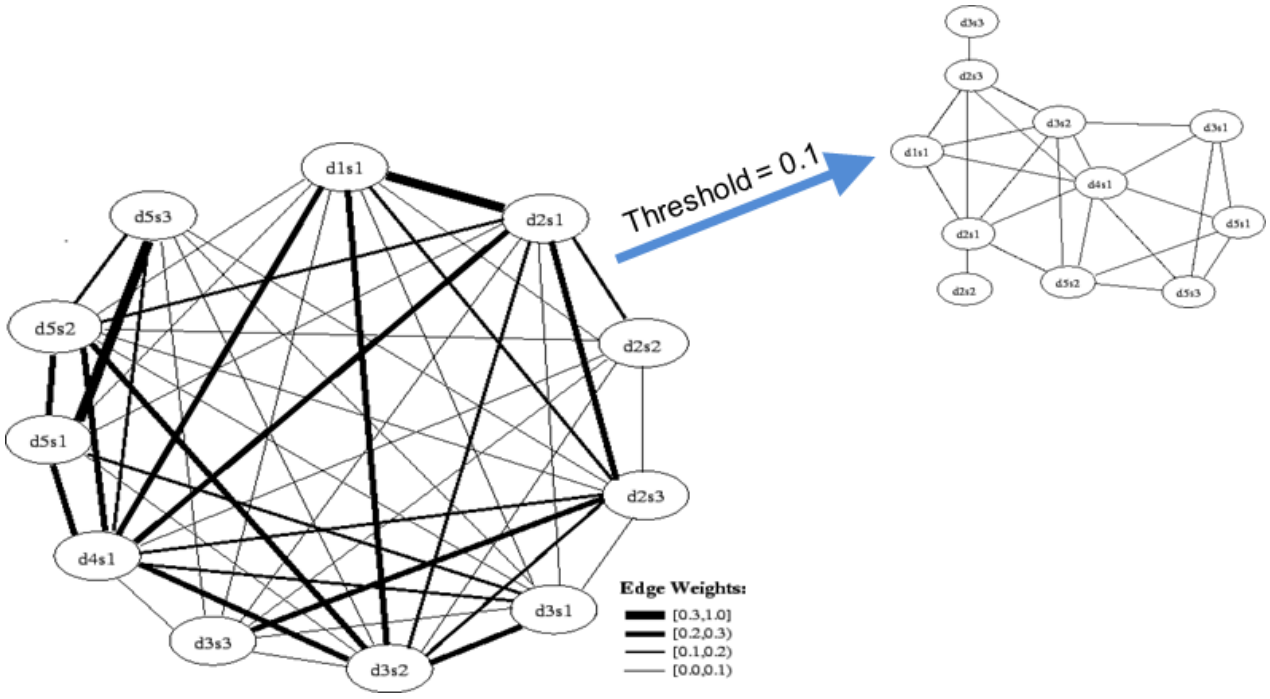
  We then create a graph where every node of the graph is a sentence from our file. The edge weights are a measure of similarity known as Intra Cosine Similarity between our two sentences/nodes. We used the JUNG package to create a Undirected Sparse Graph. The similarity between two sentences is defined by the formula.

$$\text{idf-modified-cosine}(x,y) = \frac{\sum_{w \in x,y} \text{tf}_{w,x}\,\text{tf}_{w,y}(\text{idf}_w)^2}{\sqrt{\sum_{x_i \in x}(\text{tf}_{x_i,x}\text{idf}_{x_i})^2} \times \sqrt{\sum_{y_i \in y}(\text{tf}_{y_i,y}\text{idf}_{y_i})^2}}$$

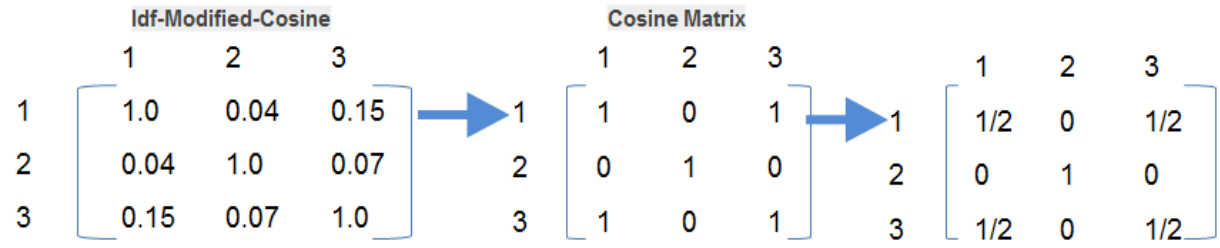
$TF_{WORD,SENTENCE}$ – The frequency of the word in the sentence.

- Lexical Centrality using Threshold and Binarization

  After graph creation, we can use two methods to compute the importance of a node in a graph. As we know the importance of a node in the graph can be computed by using Pagerank algorithm which is based on incoming and outgoing edges. But in our case we have an undirected graph with edge weights. Thus the probability of jumping from one node to any other connected node varies. Thus when we create our graph, we only add those edges where the similarity between two nodes is greater than a threshold.



As we see from the graph, adding a threshold trims certain edges from the graph. Thus higher the threshold, we will get edges only with sentences having significantly higher similarity. Once we obtain the final edges, we set edge weights as 1 where there exists an edge and 0 vice versa. We then normalize our edge weights by the number of outgoing edges from that node. Thus this gives us the uniform probability of jumping from one sentence to another having similarity greater than some threshold.



If we had, 3 sentences with following Intra-sentence similarity, we would then use 0.1 as a threshold to get the following binarized cosine matrix, and would normalize the weights using row sums. Thus our graph for lexical centrality has now probability of a sentence being similar to another one based on Intra-sentence similarity. Once we obtained this graph, we used to JUNG package again to run the Pagerank algorithm on our graph to calculate importance score of the sentences.

- Continuous Lexrank

  In the previous method, the graph that we used did not contain weights. It was just reciprocal of the degree of that node. This was because of our binarization. In continuous Lexrank, we use the intra-sentence cosine similarity values directly to construct the graph. In this algorithms we too normalize based on row sums. An important part of this algorithm is that it does not trim the graph based on a threshold. The graph becomes dense and weighted. We then run the Pagerank algorithm on our graph. The formula for Lexrank is given by:

$$p(u) = \frac{d}{N} + (1-d) \sum_{v \in adj[u]} \frac{\text{idf-modified-cosine}(u,v)}{\sum_{z \in adj[v]} \text{idf-modified-cosine}(z,v)} p(v)$$

  This approach takes into consideration the actual strength of the similarity between two sentences.

- Output

  Once we run the Pagerank algorithm our graph, every node/sentence in the graph is assigned an importance score. We then sort all our nodes based on their importance in descending order. The top five sentences are then chosen to best extract the summary of all reviews for a business.

TASK 2: Evaluation

- ROUGE-N

  The rouge-n also called Recall-Oriented Understudy for Gisting is one of the most widely used metrics in text summarization or extraction. The metric compares a summary generated by a machine to the actual reference summary generated by a human. Rouge has an open-source java package that we used to generate the evaluation. When evaluating with Rouge, we create two files, System Summary (the one that is generated by the machine) and Reference summary (human generated ground truth summary). Rouge uses two main parameters to evaluate the system generated summary. Recall and Precision.

  Recall is given by

$$\frac{number\_of\_overlapping\_words}{total\_words\_in\_reference\_summary}$$

  Precision is given by

$$\frac{number\_of\_overlapping\_words}{total\_words\_in\_system\_summary}$$

  Recall is a measure of how much of the reference summary is captured by the system summary, whereas precision captures how many system generated words are actually important. Rouge also generates F-Score which a measure of how good your test was. It is given by the harmonic mean of recall and precision. The `N` in rouge stands for 1,2,3. So if we decide to use rouge-2, it divides the reference and system summaries into N batches of 2 consecutive words. Once we generate the system and reference summaries, we name them in a certain way defined by the Rouge Documentation. We can run rouge by typing java -jar rouge2.0_xx.jar.

- Ground Truth

  Generating ground truth is and always has been a major challenge in text summarization. As each business has more than 1500 reviews which is more than 2000 sentences, it seemed very difficult to humanly generate summaries. To generate ground truth, we looked at the most useful reviews of that particular business and humanly selected five sentences from those reviews.

- Results

  The first table displays evaluation using Unigram Model.

	Continuous Lexrank			Lexrank with Threshold 0.05			Lexrank with Threshold 0.1		
Task Name	Avg_Recall	Avg_Precision	Avg_F-Score	Avg_Recall	Avg_Precision	Avg_F-Score	Avg_Recall	Avg_Precision	Avg_F-Score
1	0.16854	0.17045	0.16949	0.16854	0.15957	0.16393	0.19101	0.28814	0.22973
2	0.26829	0.352	0.3045	0.26829	0.34646	0.30241	0.2378	0.375	0.29104
3	0.27378	0.52486	0.35985	0.26225	0.4715	0.33704	0.27378	0.60897	0.37773
4	0.14524	0.38854	0.21144	0.14762	0.35632	0.20875	0.13571	0.42857	0.20615
5	0.29221	0.47872	0.3629	0.4513	0.49643	0.47279	0.01299	0.07843	0.02228
6	0.35079	0.29911	0.32289	0.35079	0.27801	0.31019	0.49738	0.4185	0.45455
7	0.32584	0.56863	0.41429	0.32584	0.56863	0.41429	0.30712	0.53947	0.39141
8	0.23497	0.27564	0.25369	0.39891	0.365	0.3812	0.07104	0.11607	0.08814
9	0.20879	0.21591	0.21229	0.03297	0.03797	0.03529	0.01099	0.02564	0.01538
10	0.61503	0.69409	0.65217	0.61731	0.69309	0.65301	0.52847	0.67836	0.59411

The second table here displays evaluation results using Bi-Gram Model.

Task Name	System Name	Avg_Recall			Avg_Precision			Avg_F-Score		
		Continuous	Threshold 0.05	Threshold 0.1	Continuous	Threshold 0.05	Threshold 0.1	Continuous	Threshold 0.05	Threshold 0.1
1	CONT.TXT	0.15029	0.15029	0.16185	0.15385	0.14365	0.25	0.15205	0.14689	0.19649
2	CONT.TXT	0.22152	0.22152	0.20253	0.29661	0.29167	0.32653	0.25362	0.2518	0.25
3	CONT.TXT	0.25074	0.24779	0.00885	0.48571	0.4492	0.04545	0.33074	0.31939	0.01481
4	CONT.TXT	0.11622	0.11622	0.11622	0.32	0.28743	0.38095	0.17052	0.16552	0.17811
5	CONT.TXT	0.27152	0.41722	0	0.45055	0.45985	0	0.33884	0.4375	0
6	CONT.TXT	0.30978	0.30978	0.4837	0.26512	0.24569	0.40639	0.28571	0.27404	0.44169
7	CONT.TXT	0.29231	0.29231	0.27692	0.51701	0.51701	0.49315	0.37346	0.37346	0.35468
8	CONT.TXT	0.18539	0.37079	0	0.22	0.34021	0	0.20122	0.35484	0
9	COUNT.TXT	0.18644	0	0	0.19412	0	0	0.1902	0	0
10	COUNT.TXT	0.60739	0.61201	0.51732	0.68848	0.6901	0.66667	0.6454	0.64871	0.58257

- ## Conclusion

  We can see from the above tables that Lexrank with threshold performs better than continuous Lexrank. As you can see we get low accuracies with increasing threshold. Thus, to decide on a accurate threshold value can also be seen as a limitation of this method. We observed that as we increasing the threshold the precision seems to increase even though the generated sentences do not make sense at all. This seems to happen because applying greater threshold tends to increase the degree of sentences with very less words and common words. Higher threshold gives sentences that are less informative and sometimes even misleading. We use a extractive technique, which only returns a subset of the original text. It does not perform any abstractive summarization, sentiment analysis, lexical analysis or text generation. Overall the extracted sentences pretty much captured the most important words with the highest IDF. Thus, we can say that extractive summarization is based on words rather than extracting meaning out of sentences yet performs better than most of the text summarization algorithms.

Conclusion

- Content based similarity approach in our case seems to be performing better for precision and recall values.
- Within collaborative filtering Pearson similarity with Item to Item approach performed best. Again in this case the sentiment analysis approach seemed to be better than the mean review stars approach.

Future Work

Task - 1:

- Ground truth creation needs to be handled in a better way.
- Efficient way for parsing and lemmatization of the json input needs to be implemented.
- Try Matrix decomposition approach to make the system more scalable.

Task-2:

- Ground truth can be obtained from experts in the fields such as restaurant critics.
- LexRank can be augment with machine learning to refine scores of vertices based on evaluations w.r.t. Ground truth.
- Abstractive summarization can be performed using LSTM-RNN. Domain related keywords can be obtained from external sources(e.g. Wikipedia) can be used to pre-train vectors for LSTM.
- System can be made dynamic by modifying the graph as new reviews come in.

Acknowledgments

We would like to thank Professor Xiaozhong Liu for all of the support and guidance and the helpful comments he provided for us during the project. Without his patience and counsel we would never have taken on this project and would never have been able to see it through to completion. We would also like to thank Andreas Bueckle and Zheng Gao and for helping us to become familiar with project and for their constant advice and support during this course.

References

1. Yelp Dataset Challenge Round 10: http://www.yelp.com/dataset_challenge
2. Previous Winners: https://www.yelp.com/dataset/challenge/winners
3. Understanding basics of Recommendation Engines (with Case Study): https://www.analyticsvidhya.com/blog/2015/10/recommendation-engines/
4. SearchKit Programming Guide: https://developer.apple.com/library/content/documentation/UserExperience/Conceptual/SearchKitConcepts/searchKit_basics/searchKit_basics.html
5. Term Frequency Inverse Document Frequency: https://en.wikipedia.org/wiki/Tf%E2%80%93idf
6. BM24 Similarity Algorithm:https://en.wikipedia.org/wiki/Okapi_BM25
7. Jelkin Mercer and Direchlet Smoothing Similarity Algorithm: http://mlwiki.org/index.php/Smoothing_for_Language_Models
8. LexRank: Graph-based Lexical Centrality as Salience in Text Summarization [https://www.cs.cmu.edu/afs/cs/project/jair/pub/volume22/erkan04a-html/erkan04a.html]