Hotel Review Search

CS410 Class Project
Shruthi Sudhanva (sudhanv2)
Sinduja Subramaniam (ssbrmnm2)
Guided by Hongning Wang

Abstract:

Hotel reviews are available in abundance today. This project focuses on providing a text-based search interface for retrieval and recommendation of hotels based on their reviews. The ranking and displaying hotels to the users for the free-text search query entered is based on the aspects present in the query and the hotel reviews. We used a dataset crawled from TripAdvisor to work on the project. The ultimate aim of the project is to integrate our solution with the main system of ReviewMiner, a work in TIMAN group. For the purpose of the course project, we have designed our own user interface, query parsing, and indexing system. The novelty in this approach is the leverage on aspect ratings during retrieval. The overall aim was to understand text processing, text search, and retrieval.

Introduction:

Our project is an add-on facility for ReviewMiner, a system developed by Hongning Wang (TIMAN group). It is a unified framework for exploring review text data with companion overall ratings. It is based on the Latent Aspect Rating and Analysis model [1]. Our focus in this project was to enable users to enter a free text query to search for hotels based on its reviews. Our work was concerned with the subset of hotel reviews in the entire system. For the hotel reviews and search, currently the users are able to enter the hotel name or location as the query and get results of hotels and reviews corresponding to them. The system did not support free text based search queries. Our project provided an independent module, which will be integrated into the system to facilitate such querying.

Right now, ReviewMiner tool does not support free text search or aspects in a query. Aspects are features like cleanliness, service, room, location, value etc. on which a hotel is judged and commented about in reviews. The system can be more useful and attractive if the aspect-matching is done, the reviews are returned specifically for aspects matching the queries. More than often, the users would be interested in giving ad-hoc queries and not always the hotel name or the location alone. The system needs a strong query-parsing stage to complement its wonderful features in review content side.

This problem that we are trying to address i.e. text based querying for hotel search, is interesting since most review based hotel search sites do not provide this facility. For example, TripAdvisor or Hotels.com only provides querying based on a hotel name or a location. Our project provides text search facility by analyzing aspects in the query and matching them with aspects in reviews of a hotel, based on which the hotels are ranked.

Our project consists of the following modules: parsing the query and segmenting it based on the aspects, parsing and indexing the dataset of reviews into suitable format, perform search and retrieval to provide ranked results and a aesthetic user interface for smooth interaction. Further details of each module will be covered in the subsequent sections.

Related Work

ReviewMiner [1], which is the system our project extrapolates. This system also provides recommendation and search facilities, but as pointed out previously, it does not identify aspects from the free-text queries. It tries to look for hotel name or location only. Also, the ReviewMiner system is not limited to hotel reviews but extends to a number of other domains as well. ReviewMiner also does summarization and comparative visual analysis of hotels based on aspects and ratings in them.

FindiLike [3] is similar to our system in that the search and recommendation happens based on a set of text based specific inputs. The opinion preferences can be expressed as a set of natural keywords such as 'close to place xyz', 'safe neighborhood', 'good breakfast', etc. The difference with our project is that we allow the entire query to be a free text rather than just keywords and match the aspects in the query and their calculated rating with the aspects and ratings identified in the reviews and apply some heuristics for normalization and ranking.

Problem Definition

Our main focus and problem is to facilitate free-text queries for hotel search and recommendation by improving the semantic interpretation of queries. This involves taking into consideration aspects matching between query and the review content. For our purpose we consider five aspects - Value, Cleanliness, Service, Room and Location. With this we also address the problem of improving the ranked search results.

The expected input for the system will be a search text query. As an example, "Find hotels in San Diego with cheap price near the beach with swimming pool".

The expected output is a ranked list of hotels that best match the user's query, based on the hotels' reviews.

In addition, for the process of integrating our system with the ReviewMiner system, the expectation is an interfacing class whose objects contain all the processed information concerning an input query like the aspects identified, ratings for each aspect, phrases describing the aspect, etc.

Methods

Our project uses Apache Lucene for indexing and retrieval. The programming language used is Java.

Our approach consists of four main modules

- Dataset parsing: Parsing and indexing the dataset of reviews into suitable format.
- Query processing: Parsing the query and segmenting it based on the aspects.
- Ranking and retrieval: Perform search and retrieval to provide ranked results.

- User Interface Design: An aesthetic user interface for smooth interaction.

Each of these functional sets is separated into individual java packages.

Dataset Parsing:

The dataset for our project is a TripAdvisor hotel review dataset. The dataset was provided to us with preprocessing performed. Each file in the dataset has details such as Hotel name, and a URL (from which we get the information about location). Each hotel has a set of reviews associated with it and has the Hotel ID as well. Along with each review, the file also contains the ratings for each of the five aspects - Value, Location, Cleanliness, Room and Service. We parse the dataset accordingly, retrieve the hotel name and location details from the file and from the url respectively and add reviews along with hotel name and location. In addition, we also parse the files for the aspect ratings of each individual review. During this process, we identify the average aspect rating of all the reviews for the given hotel.

Another function of this module involved calculating the average number of reviews per hotel from the entire dataset. The use of this value is explained in the next section.

Indexing:

The existing ReviewMiner system has a pre-existing index with hotel IDs, hotel name and hotel location. But for our purposes we also require the review content itself and the aspect ratings of each review. In this regard, we built our index, consisting of the following fields:

Hotel ID, Hotel name, Hotel location, Room rating, Service rating, Location rating, Cleanliness rating, Value rating, Review contents.

All the fields except the five rating fields are string fields. The five ratings are float fields with real values in the range of 1 to 5. Each of these aspect ratings were the average rating values calculated during parsing as explained in the previous section.

An important observation that was made during the design of the retrieval system was that average ratings of hotels with more reviews was more reliable than those with less reviews and hence should be ranked higher. Thus, the decision of boosting hotel documents was made based on the number of reviews the hotel consisted. If the document consisted of more than average number of reviews, the document was indexed with a certain amount of boosting. If otherwise, the document was simply indexed without any boosting. This approach was a naive choice which can be improvised later as a future work.

Query Processing:

As discussed earlier on, the user inputs an almost free text query which needs to be parsed into a suitable format which can be used for two tasks further - searching in the index and retrieving relevant results and convert to suitable format to be used by ReviewMiner system after integration.

For ease of parsing, we established a few fixed templates using which user can enter his/her query. This is similar to Facebook's search facility, which forces users to fit their query into one of the suggested templates. This section will use one such template example to discuss the process and output of the same - Find hotel <hotelname> in <location> <free text with requirements>. This template forces the users to put the hotelname and location in the beginning. Anything after that can be free text. Thus, we call it an almost free text query. As an example query of this format, consider

“Find hotels in Detroit with non smoking room and very clean bathrooms in downtown”

or

“Find hotel Hilton in New York with discount price and large rooms with a swimming pool”

The steps in query processing consist of the following:

- Chunking the query and identifying noun phrases - The input text query is first chunked using the Illinois NLP group’s LBJChunker. Before tagging the words in the query, stop words are removed using a compiled list of the same. The remaining words are tagged using the chunker and only noun phrases are extracted. For the example query “Find hotels in Detroit with non smoking room and clean bathrooms in downtown”, the chunked output would look like

  Find hotels, Detroit, non smoking room, very clean bathrooms, downtown
- According to the template chosen to provide the input, we first extract the hotel name (optional) and location. For example from the above template, the hotel name appears in the first chunk (if any) and location Detroit in the second.
- In the next step, we use Stanford’s dependency parser to identify the dependency between words in the chunked query. We group the phrases according to their dependencies.
- The next step is to identify the aspect that is mentioned in each phrase. For this, we use a dictionary of words provided to us, consisting of a list of around 1000 words associated with each of the five aspects. In our example,

  non smoking room - Room aspect, very clean bathrooms - Cleanliness aspect, downtown - Location aspect
- Within each phrase we next identify the adjectives and adverbs. We assign a rating to each aspect based on the phrases identified. The weights of these adverbs and adjectives were defined by us in a dictionary. But this can be extended to include more efficient weighting methods. In our example,

  Room aspect - 2, Cleanliness aspect - 3, Location aspect - 2

  We have made some assumptions like mentioning an aspect will assign a minimum rating of 2 etc. But we also believe that improvements and experimentation will be a future arena. In this process we also identify the relative weightings between aspects in a query.

All the above information extracted from the query discussed are stored in an object called QueryObject defined by us. This will be used by the ReviewMiner system post integration.

Retrieval and Ranking:

For retrieval we consider matching hotel name, hotel location, aspect fields and the review content from the index. We tried two methods of retrieval and ranking while experimenting.

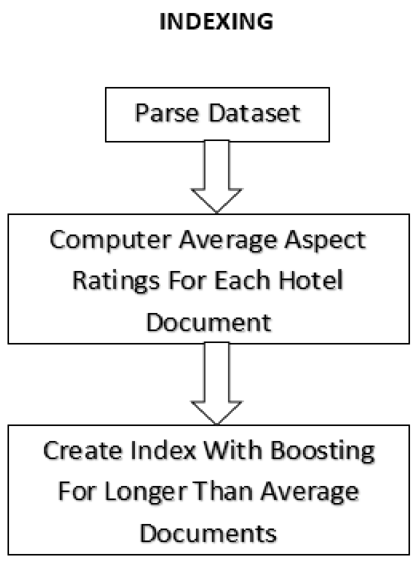
Method 1:

The first method was a very naive and basic approach where we just considered identifying the number of the reviews in the hotel document that mention the phrases identified in the query. The results were ranked according to the number of reviews that had the exact phrase match. This method did not prove very effective in terms of the results.

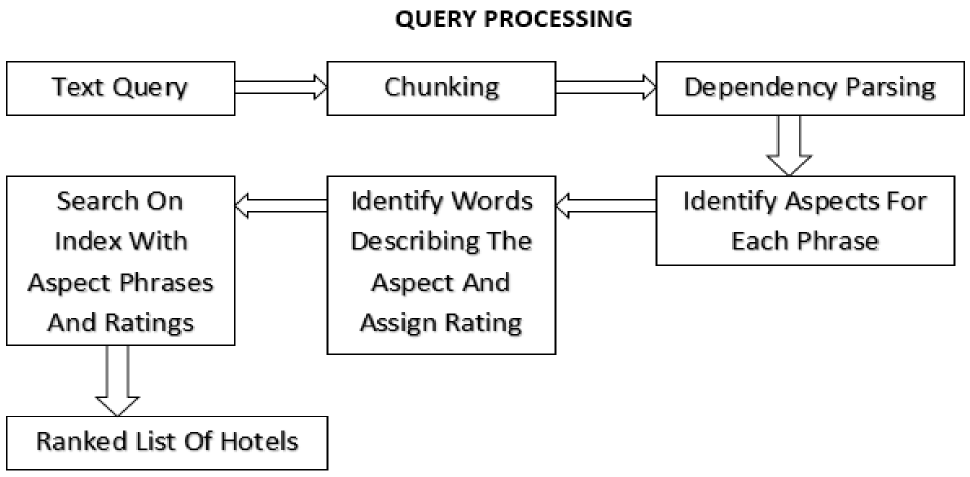
Method 2:

We planned to leverage on the aspect ratings for each review that were provided to us in the dataset. As explained in the earlier sections, we calculated the average aspect ratings while parsing the dataset itself. With the aspect ratings obtained from the query, we perform a range search on the index to identify all reviews for that hotel that have rating greater the rating in query for each aspect identified. For example, if the Room aspect has rating 3, we only consider all hotels that have average Room rating more than 3 only. We also consider the number of exact phrase matches for ranking. Also, we sort the aspect fields in descending order to obtain a good ranking.

An important heuristic that we have incorporated is about the number of reviews and reliability of the rating. For example, in case of naive ranking with Lucene, a hotel with one review that rates all aspects at 5 will be ranked above a hotel with 100 reviews that rates all aspects at 3. But, this is not a correct result. The user would want to see hotel which has more number of stable reviews. Thus we normalize this issue by boosting documents with higher number of reviews and hence reward them while penalizing hotel documents with smaller number of reviews. This is also combined with the ratings of each hotels to give an overall effective set of ranked hotels. Some design aspects are demonstrated in the figures below.


INDEXING
Parse Dataset
Computer Average Aspect Ratings For Each Hotel Document
Create Index With Boosting For Longer Than Average Documents

Indexing with hotel review content and each aspect's average rating computed for each hotel


QUERY PROCESSING
Text Query
Chunking
Dependency Parsing
Identify Aspects For Each Phrase
Identify Words Describing The Aspect And Assign Rating
Search On Index With Aspect Phrases And Ratings
Ranked List Of Hotels

User Interface Design:

Our backend processing is run as a service on a chosen port using Java IO sockets. We have used JSP to implement the client connectivity part of the project. For the front-end design we have used Twitter Bootstrap and HTML5. A screenshot of the front-end is displayed below. The user can enter a text query in the search box. Hotel results are shown along with the aspect ratings and the number of reviews. The aspect ratings of the query are displayed as well. The user can further click on a link to view all the reviews associated with that hotel.

Some technical challenges that we faced:

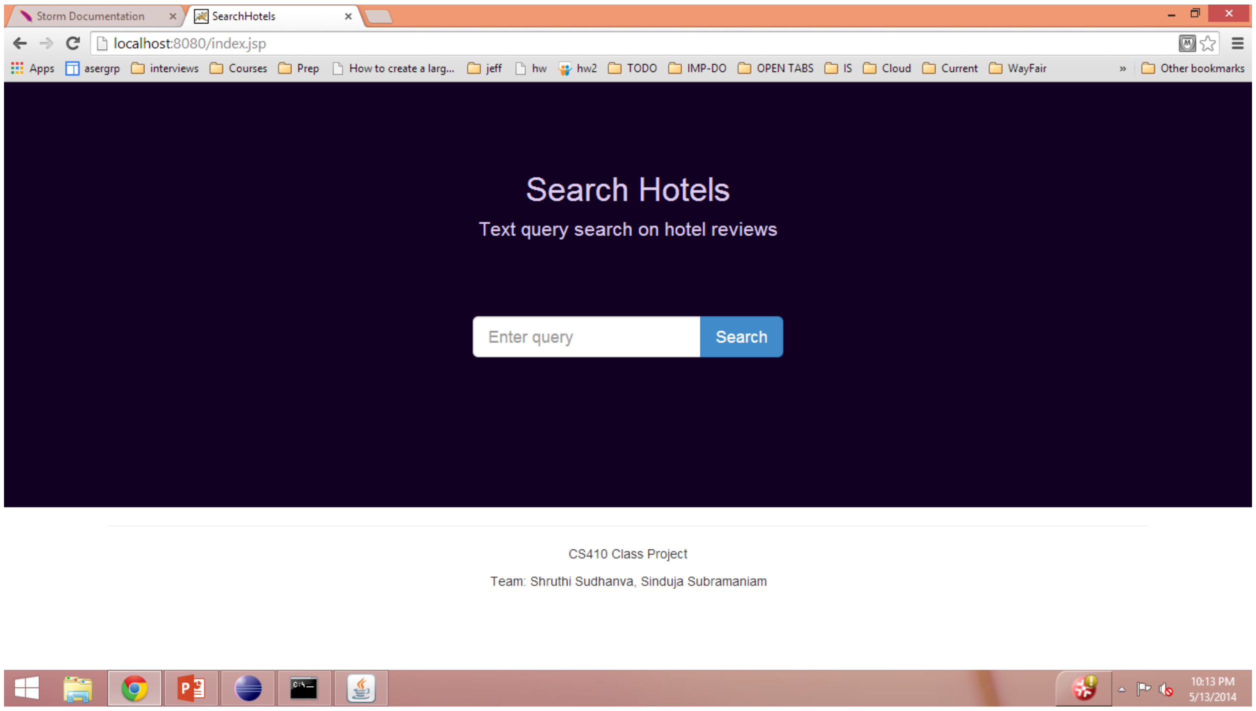
- Time taken to load the parser files and chunking was long - The first query takes a long time to execute compared to the consequent queries. We have changed it to load during the server load time to cut down the wait time of the user during query parsing stage.
- The hotels with lesser reviews (say, 1) would gain advantage over hotels with more reviews and higher rating in more in half the reviews. For normalization we tried to use lucene's boosting functionality to boost documents with lot of reviews instead of penalizing them. Normal techniques penalize long documents and this approach requires boosting of scores from long documents.

Sample Results:

We did some evaluation with a small user study. We asked a few friends to give a few input queries and give their feedback about the system. Overall, the users liked the user interface. The results were mostly relevant in many cases. It did fail in certain cases where sentence structures were a little complicated. It also failed in some cases where users entered words that were outside of our predefined dictionary words.

Some example screenshots are shown below. Screenshot 1 is the start page with a search bar. Screenshot 2 is a page that displays results for the given query. Screenshot 3 shows reviews of a particular hotel.

One of the things we tried but couldn't complete was to utilize the sentiment rating feature of LARA model that is used to form the aspect ratings in the review side. We tried a couple of natural language tool-kits available instead, but couldn't get ratings as accurate as the reviews. If the sentiment analysis feature of LARA model is incorporated into query-parsing stage as well, it would be a good improvement. During integration into the system, instead of using dictionaries as reference for rating the aspects, all that has to be done is application of the LARA model after our query-processing stage identifies aspects present and their corresponding values.



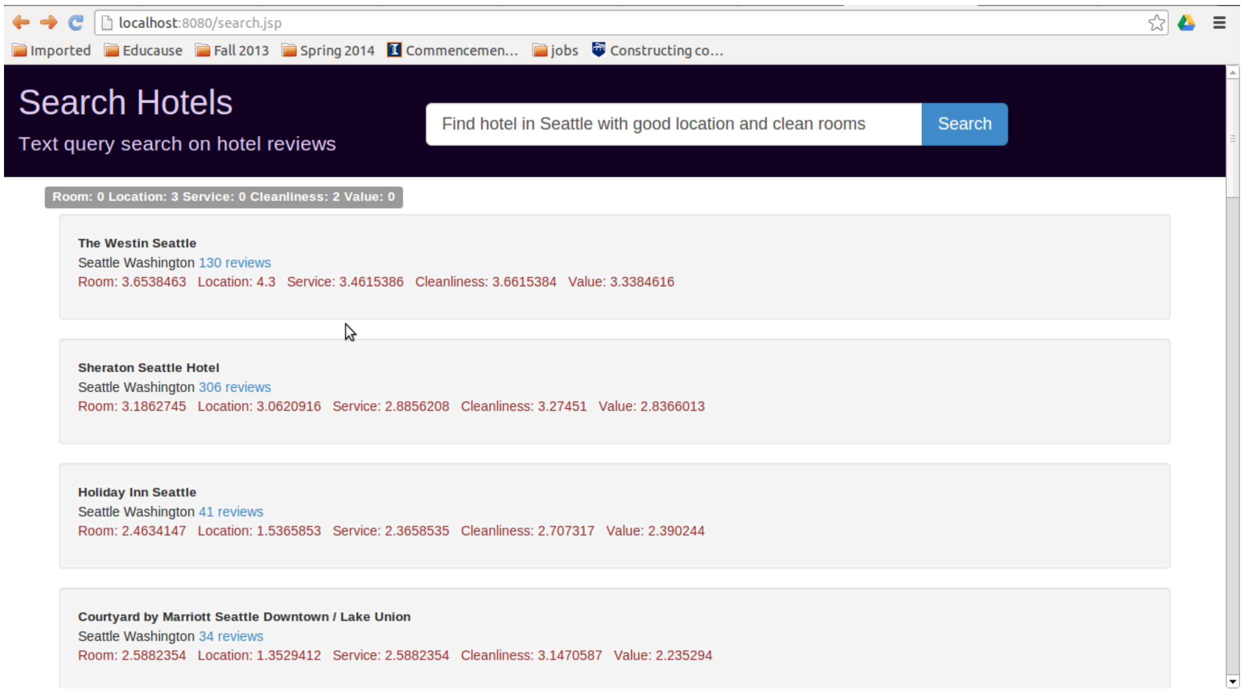
Search Hotels

Text query search on hotel reviews

CS410 Class Project

Team: Shruthi Sudhanva, Sinduja Subramaniam

Screenshot 1



Search Hotels

Text query search on hotel reviews

Find hotel in Seattle with good location and clean rooms

Search

Room: 0 Location: 3 Service: 0 Cleanliness: 2 Value: 0

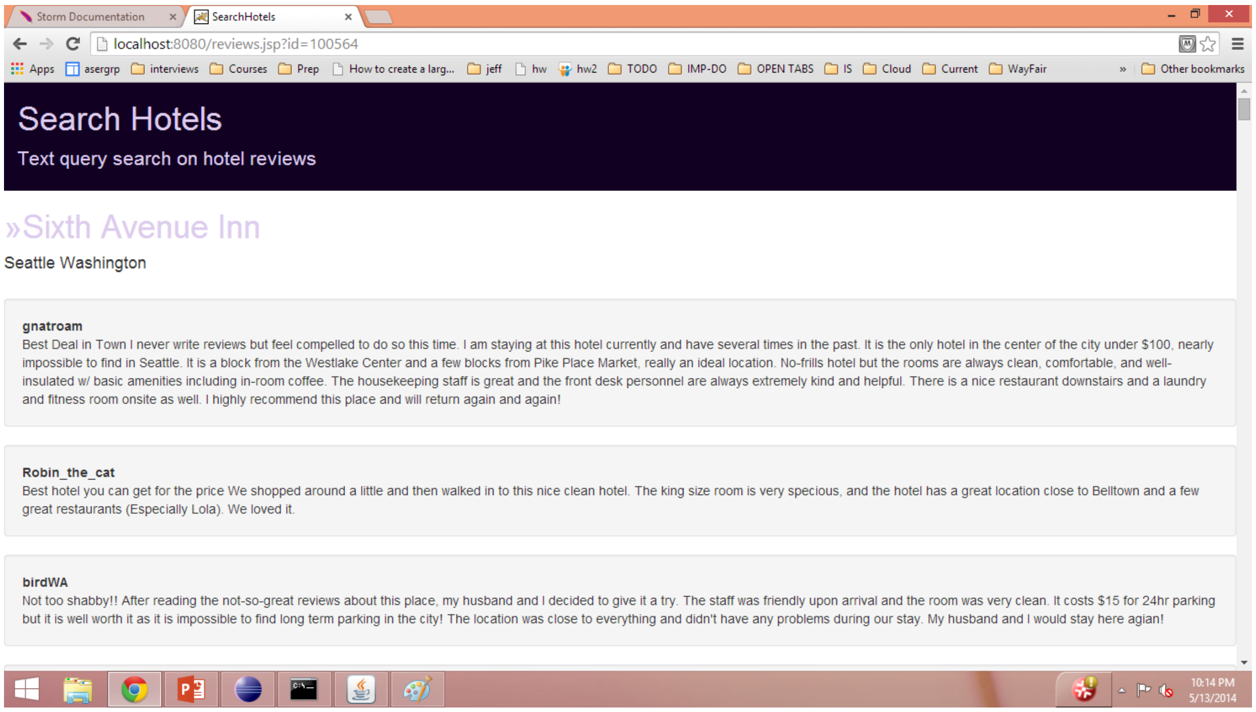
The Westin Seattle
Seattle Washington 130 reviews
Room: 3.6538463 Location: 4.3 Service: 3.4615386 Cleanliness: 3.6615384 Value: 3.3384616

Sheraton Seattle Hotel
Seattle Washington 306 reviews
Room: 3.1862745 Location: 3.0620916 Service: 2.8856208 Cleanliness: 3.27451 Value: 2.8366013

Holiday Inn Seattle
Seattle Washington 41 reviews
Room: 2.4634147 Location: 1.5365853 Service: 2.3658535 Cleanliness: 2.707317 Value: 2.390244

Courtyard by Marriott Seattle Downtown / Lake Union
Seattle Washington 34 reviews
Room: 2.5882354 Location: 1.3529412 Service: 2.5882354 Cleanliness: 3.1470587 Value: 2.235294

Screenshot 2



Search Hotels

Text query search on hotel reviews

»Sixth Avenue Inn

Seattle Washington

gnatroam
Best Deal in Town I never write reviews but feel compelled to do so this time. I am staying at this hotel currently and have several times in the past. It is the only hotel in the center of the city under $100, nearly impossible to find in Seattle. It is a block from the Westlake Center and a few blocks from Pike Place Market, really an ideal location. No-frills hotel but the rooms are always clean, comfortable, and well-insulated w/ basic amenities including in-room coffee. The housekeeping staff is great and the front desk personnel are always extremely kind and helpful. There is a nice restaurant downstairs and a laundry and fitness room onsite as well. I highly recommend this place and will return again and again!

Robin_the_cat
Best hotel you can get for the price We shopped around a little and then walked in to this nice clean hotel. The king size room is very specious, and the hotel has a great location close to Belltown and a few great restaurants (Especially Lola). We loved it.

birdWA
Not too shabby!! After reading the not-so-great reviews about this place, my husband and I decided to give it a try. The staff was friendly upon arrival and the room was very clean. It costs $15 for 24hr parking but it is well worth it as it is impossible to find long term parking in the city! The location was close to everything and didn't have any problems during our stay. My husband and I would stay here agian!

Screenshot 3

Conclusions & Future Work:

We have worked end-to-end from query parsing, indexing, retrieving matching results, and ranking the results. We have gained overall understanding of how information retrieval systems work, especially in the case of datasets where text content is present. We can relate to search engine and web pages from what we have learned using the text search.

As further extension, when one enters queries like “Maryland”, results with Maryland in the street address instead of Maryland as state are also displayed. Since we have not handled the identification of city, state, etc in the pre-processing stage of the dataset, there is no way to currently handle this using the current dataset. However, during crawling, we should get a dataset with the city and state details mapped correctly to be able to use it to index along with city and state details to avoid this issue.

Another improvement we can do is on the ranking function. Instead of using the basic approach of penalizing documents containing reviews less than average number and rewarding those with more number of reviews, we can plan a heuristic ranking function that can balance the ranking automatically based on the number of reviews, the aspect ratings and the number of exact phrase matches in the hotel reviews.

Appendix: Individual Contributions:

Shruthi Sudhanva (sudhanv2) : Dataset Parsing, Indexing, Query Processing, Java Server Pages and design, Retrieval & Ranking

Sinduja Subramaniam (ssbrmnm2): Dataset Parsing, Indexing, Query Processing, Retrieval & Ranking, Templates/cases Design and Testing

References:

List references of work related to our project or websites with related systems available here.

[1] Hongning Wang, Yue Lu, and ChengXiang Zhai. 2011. Latent aspect rating analysis without aspect keyword supervision. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining* (KDD '11). ACM, New York, NY, USA, 618-626. http://dl.acm.org/citation.cfm?id=2020505

[2] Jianxing Yu, Zheng-Jun Zha, Meng Wang, and Tat-Seng Chua. 2011. Aspect ranking: identifying important product aspects from online consumer reviews. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies - Volume 1* (HLT '11), Vol. 1. Association for Computational Linguistics, Stroudsburg, PA, USA, 1496-1505. http://dl.acm.org/citation.cfm?id=2002472.2002654

[3] FindILike system demo http://www.findilike.com/hotel-search.jsp