# SI 650 Information Retrieval

## Final Report

# Tourist attractions Information Retrieval search engine

Guanhua Xue
uniqname: ourox
Lechen Zhang
uniqname: leczhang

# Contents

# 1 Introduction

In this project, we developed a tourist attraction information retrieval search engine that aims to improve the search experience for users looking for information about tourist attractions.

For users who care about other people's comments and are unwilling to bear the risk of being cheated by either attraction websites or recommendation websites, they would like to see an information retrieval system to take real visitors' reviews into consideration that saves their time to look at evaluations one by one. Also, for users who don't really know what they want, they would like to see a retrieval system to provide them some advice based on some abstract description like "where to date with my girlfriend" or "graduation travel with classmates". Nobody knows the true answer, but probably our system can provide some matches based on the review texts.

However, previous travel search engines didn't optimize for these abstract descriptions. If you search "graduation travel" on travel search engine, you are very likely to get a result of "Graduate Ann Arbor", which is a hotel name. There's also very few retrieval research that focus on travel Dataset. Previous research in this area has focused on using collaborative features, content queries, and ontology-based systems, but most of these approaches will take the User Profile as the most important factor.

Our approach is different in that we do not consider personal user information. Instead, our main dataset is the description and tourists' evaluation of the destination. We mainly focus on expanding abstract queries and using a combination of various features and machine learning models to rerank the results. The results show that the query expansion and learning to rank techniques were able to improve the performance of the search engine compared to the BM25 baseline by about 10% in all evaluation metrics includes MAP and NDCG.

The advantage of our approach is that it allows us to better capture the real human thoughts and experiences of tourists. Other than applying basic matching method like BM25 to different data field, we did a sentiment analysis on negative reviews to make full use of their relevance to our query. We also used neural network language model like BERT to understand the user's demand and compare them against the Reviews and Descriptions of the result. This allows the IR system to be able to retrieve abstract queries like "the most romantic place" and provide results that are as much close to real-life user feedback

as possible.

The main contribution from the project is giving us an in-depth understanding about the diversity and potential of travel dataset. We found that travel destination retrieval is a very complicated work. Many features can be effective on improving the performance. But they are also strongly correlated, meaning that simply adding features won't necessarily bring better results. Combining these features are like building blocks. Features should be reasonably picked and place to prevent overfiting collapse. We found that the features that reflects human's real thought can be very powerful, such as sentiment analysis and BERT encoder model. However, for such a diverse dataset, we still need to be careful about its robustness.

# 2 Data

## 2.1 Data Source

For this project, we use two websites for crawling tourist attractions in each state and their corresponding ratings, comments, review numbers separately.

The tourist attraction data from the official website https://www.city-data.com/articles/ contains all tourist attractions in the United States and the corresponding description. Using *BeatuifulSoup, requests* and *json* packages, I first use *BeatuifulSoup* to crawl all tourist attractions website contents and their corresponding URL for each state and store them in json format. Table1 is an example.

| Data |
|---|
| "https://www.city-data.com/articles/16th-Street-Baptist-Church-Birmingham.html": <!DOCTYPE html>°<html lang= ... |

Table 1: Json Example

Then I deal with json file for each state to find out the Name,State,Title and Description attributes for each attraction using BeatuifulSoup to parse the json file and write attractions into csv format with columns*Name,State,Title, Description* for each state. Table1 is an illustration.

| Name | 16th Street Baptist Church |
|---|---|
| State | Alabama |
| Title | 16th Street Baptist Church - Birmingham, Alabama - Historic Church |
| Description | 16th Street Baptist Church is located on Sixth Avenue North in Birmingham Alabama and is approximately a seven minute drive from Birmingham Airport. The church campus is open for tours 10am to 4pm Tuesday to Friday and 10am to 1pm by appointment only on Saturdays... |

Table 2: Csv Example

Now we are going to add the ratings, number of comments, and comments' content to each of these tourist destinations. These data can be crawled form https://www.tripadvisor.com/, which provides many detailed comments to each tourist attraction. However, we have to build up a mapping between the name of tourist attraction to the website of it on TripAdvisor.

To do so, we used a Google API (https://customsearch.googleapis.com/customsearch/v1). We can send the attraction name to this API, and it will return the website url of TripAdvisor. Then, we use the *requests* package of Python to simulate normal user access request, and get the *html* file as follows:


Figure 1: Crawl for TripAdvisor

After that, we used *BeautifulSoup* to extract the useful information from the page, including the tag of rating and review numbers, the rating of each review and its title/content. For each tourist destination, we crawled 2 pages of comments (20 comments), However, since it's useless to store them in 20 columns, we did a classification. For those reviews that have $rating \geq 4.0$, we will classify them as "positive review", which was stored into *pos_review_title* and *pos_review_content*. The remaining comments were stored into *neg_review_title* and *neg_review_content*.

After finishing the above steps, we can get enough real evaluation data. The crawling is a bit harder, since the website has protection measure for frequent access, but since we can do the crawling based on the list of tourist destinations we already have, the workload was less than simply crawling the TripAdvisor website.

## 2.2 Data Statistics

After getting both tourist attractions and evaluation datasets, we join them and integrate them into one document with the following columns for each destination: *docno*, *Name*, *State*, *Title*, *Description*, *hotel*, *overall_rate*, *review_num*, *pos_review_title*, *pos_review_content*, *neg_review_title*, *neg_review_content*. A simplified sample is shown below to show the data structure of one tourist destination.

| docno | doc_0 | hotel | FALSE |
|---|---|---|---|
| Name | Alabama Theatre | overall_rate | 4.5 |
| State | Alabama | review_num | 1797 |
| Title | Alabama Theatre, Birmingham, AL | | |
| Description | The Alabama Theatre was established in 1927 by Paramount Studios. It was meant to be a place to showcase the Paramount brand of films. It was used as a movie palace'' for 55 years, except for a time with the annual Miss Alabama Pageant and the weekly Mickey Mouse Club. etc. | | |
| pos_review_title | Fun family night. Great Show. Entertaining, but whiter than Antarctica . Simon and Garfunkel Story. Greg Rowles Country and Gospel Show. | neg_review_title | Duped at the door. Go see the Carolina Opry's Christmas Show instead of this one.. Ok Show But Felt Tired and Outdated. Disappointing. EH. |
| pos_review_content | The show was wonerful and we really enjoyed the music. The fact that we were able to take 5 kids and 3 adults for the price of the 3 adults tickets was fantastic. The only dissappointment was that the show was only the singing and dancing and the gentleman who does the juggling and stuff was not there that night. | neg_review_content | We entered the Alabama Theatre for the variety show and we directed to the left to get our programs. There was no program. It was a lady working for Wyndham who promised us an 8 day, 7 night stay for listening to a two hour pitch by Wyndham. |

Figure 2: Data structure sample

We can get some statistics from the dataset about the distribution of columns, which is plotted below.

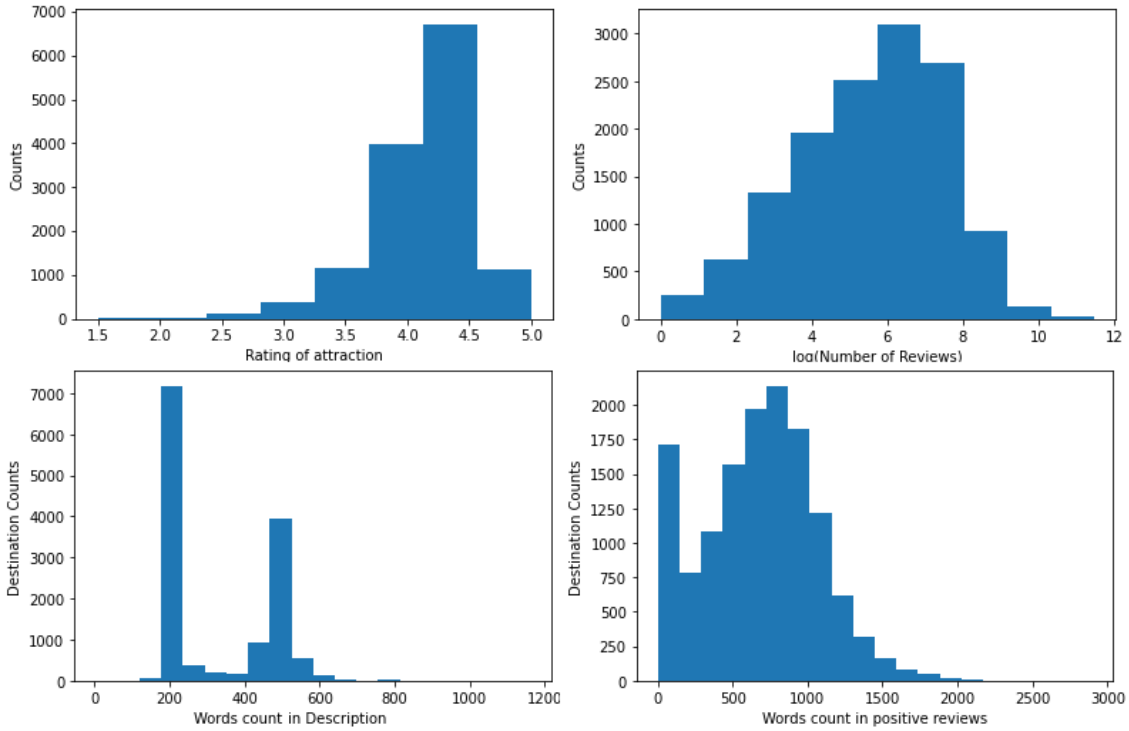


Figure 3: Distribution of Dataset variables

For our baseline system, we indexed 2 fields from the dataset, which are the "Description" and "*pos_review_content*". Summarizing the data, we can see that there are 13571 tourist attractions across America with 65000 terms, 4111997 postings and 7330675 tokens in total as shown in Fig4 .

```
Number of documents: 13571
Number of terms: 65000
Number of postings: 4111997
Number of fields: 0
Number of tokens: 7330675
Field names: []
Positions:   true
```

Figure 4: Data Statistics

## 2.3 Data Annotate

### 2.3.1 Query Design

Each of us designed 20 queries and the queries include various aspects the user may be interested in such as "most romantic place" and "most suitable ski destination" as shown in Fig5. The query like "where to date with girlfriend" is probably a popular abstract question among real users, even though the right answer differs from person to person. Also to improve diversity, abstract queries like "graduation trip destination with classmates" may not retrieve the best results according to each persona's preferences but these can be actual queries in a real-use context. Since our data include 20 comments, overall rating, and review number for each tourist attraction, queries like "best amusement park with many comments" and "highly rated parks for picnics" could test different abilities of our model, which is also a core principle in our query design.

| qid | query | qid | query |
|---|---|---|---|
| 1 | hiking in Michigan | 21 | best place for camping |
| 2 | play golf in California | 22 | the most romantic place |
| 3 | best place to take a selfie in summer | 23 | a place most suitable for barbecue |
| 4 | educational place for children and kids | 24 | highly rated parks for picnics |
| 5 | summer vacation destination | 25 | mountain climbing destinations |
| 6 | birthday party place in California | 26 | the best resort in Alabama |
| 7 | peaceful and relaxing forest | 27 | the best city museum to visit |
| 8 | hotel with swimming pool and beautiful scenery in Florida | 28 | popular ski destination in winter |
| 9 | must-see attraction in New York | 29 | a place best to watch sunset and not crowded |
| 10 | weekend camping destination and not crowded | 30 | seaside place with most delicious fish to eat |
| 11 | national park with convenient pulic transport | 31 | good family tour destinatioon with high rating |
| 12 | where to date with girlfriend | 32 | recommended barbecue place |
| 13 | graduation trip destination with classmates | 33 | most beatuiful beaches to swim |
| 14 | sites of Civil War | 34 | historical significant attraction |
| 15 | nice fishing place in Michigan | 35 | biggest art museum |
| 16 | fisherman wharf with delicious restaurants | 36 | national park with highest rating |
| 17 | landmark in San Francisco | 37 | popular sites to watch maple leaves |
| 18 | cheap holiday tour with friends | 38 | attraction with hot springs |
| 19 | national park with nice tour guide for driving | 39 | popular snow watching sites |
| 20 | beautiful valley and waterfall | 40 | best amusement park with many comments |

Figure 5: Query

### 2.3.2 Document Choice for Annotation and Annotation Metrics

To annotate the data, we developed the following metric for evaluating each document in the range from 1 to 5.

| Score | Metrics |
|---|---|
| 5 | Description and Comments completely agree with query |
| 4 | Description and Comments mostly agree with query |
| 3 | Only core concept of Description and Comments agree with query |
| 2 | Only minor concept of Description and Comments agree with query |
| 1 | Description and Comments completely unrelated to query |

Table 3: Annotation Metrics

Then, we designed a user interface with database, so that we can do a quick and comfortable annotation work without repeating. We used this system to finish 4000 annotation works for document-query pairs. The annotation program is shown below:

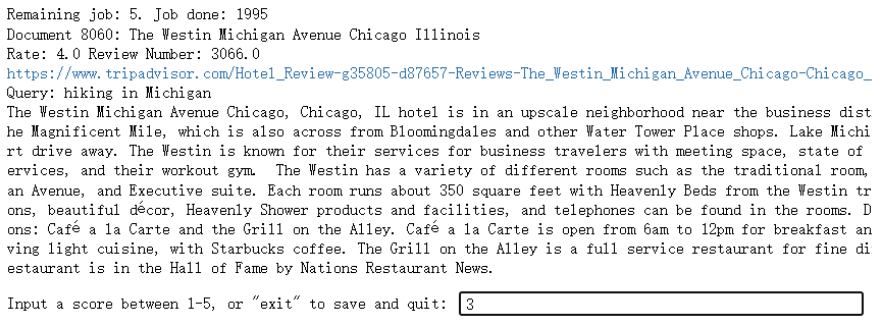


Figure 6: Annotator interface

# 3 Related Work

## 3.1 Data Source Comparision

The data source of information retrieval on tourist attractions has been changing over time from traditional attraction websites to social media platforms. As Wöber (2006) examined the visibility of tourism enterprises, particularly destination marketing organizations and individual hotel operations in Europe and found that many tourism websites suffer from very low rankings, which makes it extremely difficult for online travelers to directly access individual tourism websites. In addition, other tourists' reviews and information on social websites is becoming more and more important for traveler's reference. A

study conducted by Xiang and Gretzel (2010) proves that social media form a substantial part of the search results in search engines, which means that search engines probably lead travelers to social media sites. The fact is that social media is playing growing importance in online tourism while tourism businesses have little control.

In our own approach, the data we use combine both data from tourist attraction websites and rating and review data from TripAdvisor, which combines official descriptions and other travelers' information together.

## 3.2 Approach Comparision

Sara Evensen (2019) designed Voyageur, an application of experiential search to the domain of travel, applying TextRank which is a classic algorithm for sentence summarization to remove sentences of similar meaning or unimportant ones in their dataset. Voyageur also uses effective filters for tips including phrase patterns (e.g, sentences containing "make sure to") and part-ofspeech tags (e.g, sentences starting with verbs) for dataset preprocessing.

Another approach to retrieve tourist attractions is as Francesco Riccia, Karl Wöber and Andreas Zins (2005) do in their research using Collaborative Features (clf), Content Queries (cnq) and Cart to develop a travel recommender system with graphic design. The collaborative features they use contain user information including wishes, constraints or goals and Content Queries are queries built by constraining (content) features, i.e., descriptors of the products listed in the catalogues.

The ranking systems for hotels created by Anindya Ghose (2012) is also interesting. The method this author used to identify hotel characteristics is through assigning a predesigned survey to Amazon Mechanical Turk (AMT) to get original data. For example, to train a model to solve specific distraction restrictions, geo-tagging made by AMT becomes the original data for the machine learning model to train and test.

In addition, Dimitris N. Kanellopoulos (2008) developed ontology-based system for intelligent matching of travellers' needs for Group Package Tours uses ontology architecture mixing multiple aspects to meet travellers. The ontology system takes travellers' personal preferences, travel agencies information, possible offer, relations and other features into consideration to establish a trained model for the final system.

Unlike the above approaches, we do not take users' personal information into consideration and dig into query itself since the modeling for users' strategy and behaviors when

searching for attractions are complex. (Fodness and Murray 1997). Using filters to preprocess the data query mentioned in the articles above is an improvement for our model. Furthermore, the advantage of our model is to use query expansion models to expand the abstract queries that are very likely to come from people without information retrieval skills, and then use BM25 to collect enough data for our neural network and machine learning model to rerank, thus making our model much closer to the real human thoughts (thanks to the collection of reviews in our dataset, we are capable of building a connection between query and real user experience.)

# 4 Methodology

During the preprocessing section, we first filtered all tourist destinations that didn't have any positive comment, and places where average scores are lower than 3 (5 in total). We apply log function to the review number for data normalization since the original data is skewed data. We picked 2 fields in the dataset for indexing now, which is *Description* (The description of tourist destination) and *pos_review_content* (All review contents with $rate > 4.0$). In the future, we will try to index the negative review contents as well. Then, we used the default setting of Terrier to do some preprocess in indexing step, which is the $TerrierStemmer.porter$ for stemmer and $TerrierStemmer.terrier$ to filter the stopwords. Also, we added the positional indexing here, in order to improve the retrieval performance to some particular queries by using Query Expansion method (Will be explained later).

During the retrieval section, we used the default parameters for BM25 as our baseline. We will try some custom weighting to replace BM25, but we think that making improvements on pipelines can produce more significant improvement. There's many patterns we found during the data evaluation that emphasized the importance of "Query Expansion" and "Learning to Rank". We are planning to apply these pipelines on our basic model.

## 4.1 Query Expansion

During our early-stage baseline analysis, we found that the queries like "romantic place" or "where to date with girlfriend" are hardly understood by our baseline system. However, by using the "Query Expansion" method, we are expecting them to be expanded into words that describe places for couples. This method can significantly improve the recall/accuracy

rate, and hence improving the retrieval evaluation metrics like MAP and NDCG. However, each new Query Expansion model needs extra annotations for evaluation, which means we can only test limited number of models in this section. In our project, we compared 2 main Query Expansion model, which is $Bo1QueryExpansion()$ and $SDM()$(Sequential Dependence Model) in Pyterrier package.

For $Bo1QueryExpansion()$ model, I firstly use basic BM25 to find the top ranking documents, then use the Query Expansion tool to rewrite and assign weight to the terms that frequently occur in top ranking documents. This method might be effective for queries like "go fishing in Michigan", since the word "fish" itself can be linked to aquariums, but if new keywords like "lake" are added, then we are less likely to retrieve these disturbance.

For Sequential Dependence Model, it will give a higher score for documents when "the query terms occur in closer proximity". This model is very likely to improve the performance of queries like "national park with convenient public transport". Without positional indexing and Sequential Dependence Model, the word "national" and "public" will independently become the key matching word, giving the retrieval results like government buildings. However, if we use SDM model, we are more likely to retrieve destinations about "national park" and "public transport".

## 4.2 Learning to Rank

In our project, we will use the coordinate ascent from FastRank as a main structure to train a Learning to Rank model. We will also create a Random Forest model from sklearn to compare the weight of each feature. The split of train/valid/test dataset is 60:20:20, which means we have 24 training queries and 8 test/validation queries.

The Learning to rank model first retrieves a batch of documents from an index using the BM25 ranking function, applies a generic function to the retrieved documents, and then applies a series of transformations to generate various features (e.g. positive review titles, ratings, and so on). We will use 8 features in total, including BM25, TFIDF, Number of reviews in log scale, Rating, length of positive review's title, State Name, pairwise BERT reranker, sentiment and relevance analysis on negative comment, and Sentence transformers' cosine similarity. The first 5 features are considered "basic feature" in our pipeline. All these features will be explained in the following subsections.

### 4.2.1 Basic features

These features are likely chosen to capture important aspects of the documents that may be relevant for ranking.

**The TF-IDF feature** is important because it is a commonly used basic metric to evaluate the relevance score between query and document. Though it's worse than BM25 independently, it can still become a useful feature that reflects the Term frequency and Inverse Document Frequency.

**The Review Number in log scale feature** allows the IR system to weigh the popularity of a document in relation to other documents in the collection. Since the raw review number distribution is skewed, we applied a log scale to make it more interpretable by our model. This can be useful because documents with more reviews may be more popular and therefore more likely to be relevant to the query.

**The overall rate feature** is very helpful because it enables the search system to consider the quality of a tourist attraction in relation to other tourist attractions in the collection. This can be useful because tourist attractions with higher overall rates may be of higher quality and therefore more likely to be preferred by the user. No user will like a zero score destination, even if the description meets their demand. Also, almost no user will try to search "the worst" destination (probably), so prioritizing these high ratings will bring us positive effects in most cases.

**The length of the positive review title** is useful for ranking tourist attractions in the IR system because it scores the depth or detail of the review in relation to other tourist attractions in the collection. This can be useful because longer review titles may indicate more in-depth or detailed reviews. Also, they are very likely to be popular destinations because people are willing to write long reviews. Therefore, prioritize reviews with longer titles can improve the possibility to retrieve popular and informative tourist destinations, thus improving the overall ranking score.

**State Name Matching** might also be a important factor for ranking. Our Learning to Rank system is expected to give a higher score for those results that meet the users' geographical demand. For example, for queries containing specific state name like "Best beach in Hawaii", whether or not the results' State name matches "Hawaii" could be very important to users. In practice, no user would like a search result of beach in California after they typed in "Hawaii", even if their BM25 scores are the same. Therefore, whenever

the state name occur, it should become a deterministic factor, and that's why we select it as an independent feature. Since we crawled the State data for all tourist destinations, we can easily get a Boolean feature by testing whether or not the document's geographical keyword is mentioned in query.

### 4.2.2 Pair-wise Bert Reranker

We try to use pair-wise BERT, a transformer-based neural network language model, as part of a reranking feature for information retrieval tasks. In this process, the BERT model is implemented as a vanilla transformer using the reranker function from the ONIR module. It will do a pair-wise train to find the relationship between query and document, and return a score for ranking. For training, the first step is to specify the field of the dataset to train. Due to the limitation of GPU memory, using all positive review titles to train will cause the memory to exceed, so we use half of the positive review titles for comparison. This is achieved by applying a generic function extracts the "revised_title" field from the dataset at an index determined by the "docid" field of the input data. The 'reviesd_title' attribute is a combination of half of all the positive review titles of this tourist attraction site.

The BERT model is trained using the fit method, which takes as input the training data (train_data), validation data (valid_data) and relevance judgements (qrels). After finishing the training, we stored our trained bert model into a checkpoint "VBERT_CACHE_NAME" (trained_bert55.tar.gz). Also, we can load this pre-trained BERT model directly from this checkpoint.

### 4.2.3 Negative comment relevance and Sentiment Analysis

We have only indexed Positive Reviews into our baseline model. However, negative reviews can also be a helpful information. But we can't directly add them to the index, since "worst place for fishing" or "disgusting campsite" may return a high relevance score to queries like "fishing" and "camping". But if we add a sentiment score to the relevance score of negative reviews, then we can differentiate the difference between "worst place for fishing" (neg sentiment score = 0.577) and "just so so for fishing" (neg sentiment score = 0). In this project, I used the *nltk.sentiment.SentimentIntensityAnalyzer*() function in *nltk* (Natural Language Toolkit) library, which returns a few scores that reflects the

proportion of each component (positive/negative/neutral), and I used the "neg" score as a weighting parameter to the BM25 relevance score of Negative reviews. Hence, our new feature should be the BM25 score of Negative review multiplies its Sentiment intensity.

Furthermore, these sentiment scores can be computed in advance to save the training/running time, because it only reflects the properties of dataset (*neg_review_content*). Therefore, I pre-computed the sentiment scores of all my documents, and stored them into *dataset_sentiment.json*. These scores can be directly read by our pipeline.

### 4.2.4 Sentence to Vector (Sentence-transformers)

Inspired by the bi-encoders in Homework 4, we want to find a way to transform our query and our documents into vectors, and compare their cosine similarity. We tried to train bi-encoders by ourselves but failed (will be mentioned in last chapter), and finally we directly used other's pre-trained model to vectorize the text. We used SentenceTransformer class from sentence_transformers library (https://www.sbert.net/) to simplify the steps, and we used the all-MiniLM-L6-v2 pre-trained model from https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2. The model can convert text into 384 dimensional vector, which is very economical but powerful.

We use this transformer to transform our query text and the Description text of our tourist destinations into vectors. Then, we use the cosine similarity between them as one of our Learning to Rank feature. If our query set is fixed, we can even pre-calculated all the query-doc pair similarity in advance to save retrieval time. But in order to make our code interactive, we call this model everytime we input a new query. This makes our training time doubled.

# 5 Evaluation and Results

The naive system we used randomly matches documents with queries.

The baseline system, as mentioned above, is the BM25 model with default parameters. The preprocessing includes the *TerrierStemmer.porter* for stemmer and *TerrierStemmer.terrier* to filter the stopwords.

Here's some statistics of our BM25 annotation results. We use these statistics to evaluate the quality of our queries and the dataset.

| qid | query | Acc_all(%) | Acc@10(%) | qid | query | Acc_all(%) | Acc@10(%) |
|---|---|---|---|---|---|---|---|
| 1 | hiking in Michigan | 10 | 40 | 21 | best place for camping | 100 | 100 |
| 2 | play golf in California | 90 | 100 | 22 | the most romantic place | 62 | 60 |
| 3 | best place to take a selfie in summer | 70 | 80 | 23 | a place most suitable for barbecue | 30 | 30 |
| 4 | educational place for children and kids | 92 | 80 | 24 | highly rated parks for picnics | 68 | 60 |
| 5 | summer vacation destination | 78 | 70 | 25 | mountain climbing destinations | 86 | 90 |
| 6 | birthday party place in California | 26 | 20 | 26 | the best resort in Alabama | 20 | 20 |
| 7 | peaceful and relaxing forest | 64 | 80 | 27 | the best city museum to visit | 44 | 50 |
| 8 | hotel with swimming pool and beautiful scenery in Florida | 24 | 20 | 28 | popular ski destination in winter | 96 | 90 |
| 9 | must-see attraction in New York | 6 | 0 | 29 | a place best to watch sunset and not crowded | 86 | 90 |
| 10 | weekend camping destination and not crowded | 66 | 70 | 30 | seaside place with most delicious fish to eat | 44 | 50 |
| 11 | national park with convenient pulic transport | 10 | 20 | 31 | good family tour destinatioon with high rating | 56 | 70 |
| 12 | where to date with girlfriend | 90 | 60 | 32 | recommended barbecue place | 68 | 60 |
| 13 | graduation trip destination with classmates | 24 | 30 | 33 | most beatuiful beaches to swim | 92 | 70 |
| 14 | sites of Civil War | 98 | 100 | 34 | historical significant attraction | 94 | 100 |
| 15 | nice fishing place in Michigan | 6 | 10 | 35 | biggest art museum | 28 | 50 |
| 16 | fisherman wharf with delicious restaurants | 32 | 40 | 36 | national park with highest rating | 32 | 40 |
| 17 | landmark in San Francisco | 62 | 90 | 37 | popular sites to watch maple leaves | 62 | 100 |
| 18 | cheap holiday tour with friends | 38 | 50 | 38 | attraction with hot springs | 98 | 100 |
| 19 | national park with nice tour guide for driving | 38 | 60 | 39 | popular snow watching sites | 76 | 100 |
| 20 | beautiful valley and waterfall | 90 | 90 | 40 | best amusement park with many comments | 22 | 0 |

Figure 7: Annotation Statistics of BM25

Here accuracy means relevant, which means annotation score >= 3. Based on the Annotation Statistics, we found that the performance of our baseline system differs a lot for different queries. Some of them are due to the limit of document size. For example, since we only have 134 tourist destinations of Michigan in our database, it makes sense that we can only get 10% precision for the query "Hiking in Michigan". In contrast, California has 3093 tourist destinations in total, which makes everything related to California has a very high accuracy rate (like "play golf in California"). In future query design, we should add some query that has appropriate sample size to better test our system.

Also, some queries are not understandable to our system temporarily. For example, our baseline system can't understand "amusement park with many comments", since "comments" should function as a filtering keyword, not a retrieval keyword. Queries like "cheap holiday tour" is also not understandable to the system. Most of the retrieval result are related to "cheap markets" or comments like "not cheap".

For queries like "lovers dating places", though it has a high accuracy rate, the main

reason is that most tourist destinations are to some extent related to "dating", but they are not special to the query. Therefore, most annotations for this query is 4 points, and very few of them can get 5 points (Due to the limited space, the result is not listed in this Figure). Therefore, Query Expansion method is needed for this kind of abstract queries to improve the performance.

The evaluation method is by using the *Pyterrier.Experiment* function. We used 2000 annotated document-query pairs for this evaluation. First of all, we define "relevant" to be $score >= 3$. Then, we picked the following criteria.

- $[M]AP(rel = 3)$: The average precision scores at each relevant item.
- P(rel=3)@10: The percentage of documents in the top10 results that are relevant.
- NDCG: The normalized Discounted Cumulative Gain of all results.
- NDCG@5: The normalized Discounted Cumulative Gain of top5 results.
- NDCG@10: The normalized Discounted Cumulative Gain of top10 results.
- mrt: Mean response time.

The resulting evaluation metrics is shown below. The first row is the naive random system. The second row is the baseline BM25 system.

| | name | AP(rel=3) | P(rel=3)@10 | ndcg | ndcg_cut_5 | ndcg_cut_10 | mrt |
|---|---|---|---|---|---|---|---|
| 0 | random | 0.037669 | 0.1100 | 0.307560 | 0.290831 | 0.295594 | 4296.856476 |
| 1 | bm25 | 0.482408 | 0.5425 | 0.651365 | 0.616126 | 0.636918 | 6.116863 |

Figure 8: Evaluation Metric for Naive and Baseline system

Based on the Evaluation Metric, we found that BM25 has made a huge increase on all aspects of data. The random accuracy rate is only 4%, while the BM 25 model can provide 48% accuracy rate. It's worth mentioning that our BM25 model's *ndcg_cut_5* is lower than *ndcg_cut_10*, and *ndcg_cut_10* is lower than *ndcg*. It indicates that we should put emphasis on the reranking algorithm. An ideal search engine should has a large *ndcg_cut_5* and *ndcg_cut_10* value to improve the user experience. Also, $P(rel = 3)@10$ for BM25 should also be further improved for the same reason. In the future evaluation, we are going to compare these 6 criteria to determine whether or not our design is successful.

## 5.1 Query Expansion Results

The evaluation metrics are shown below: We compared the performance of Baseline, Bo1 Query Expansion and Sequential Dependence Model (SDM).

| | name | AP(rel=3) | P(rel=3)@50 | ndcg | ndcg_cut_5 | ndcg_cut_10 | mrt |
|---|---|---|---|---|---|---|---|
| 0 | BM25 | 0.505849 | 0.570 | 0.740277 | 0.667947 | 0.684460 | 18.046762 |
| 1 | Bo1 QE | 0.539878 | 0.600 | 0.763332 | 0.680896 | 0.695181 | 48.344207 |
| 2 | SDM | 0.512819 | 0.568 | 0.742006 | 0.692826 | 0.699737 | 30.140683 |

Figure 9: Evaluation of Query Expansion methods

From the evaluation metrics we can see that all Query Expansion methods work well in all criteria. Bo1 Query Expansion has the best performance of Precision (AP/P@50) and overall ranking (NDCG). However, SDM model outperformed Bo1 QE in the high ranking results (NDCG@5/@10). To see the query expansion effect on each query, we made following Precision rate table for later analysis.

| qid | query | BM25 Acc(%) | Bo1 QE Acc(%) | SDM Acc(%) |
|---|---|---|---|---|
| 1 | hiking in Michigan | 10 | 10 | 10 |
| 2 | play golf in California | 90 | 52 | 90 |
| 3 | best place to take a selfie in summer | 70 | 82 | 70 |
| 4 | educational place for children and kids | 92 | 96 | 92 |
| 5 | summer vacation destination | 78 | 82 | 80 |
| 6 | birthday party place in California | 26 | 20 | 26 |
| 7 | peaceful and relaxing forest | 64 | 70 | 62 |
| 8 | hotel with swimming pool and beautiful scenery in Florida | 24 | 12 | 24 |
| 9 | must-see attraction in New York | 6 | 30 | 24 |
| 10 | weekend camping destination and not crowded | 66 | 98 | 66 |
| 11 | national park with convenient public transport | 8 | 6 | 8 |
| 12 | where to date with girlfriend | 86 | 86 | 86 |
| 13 | graduation trip destination with classmates | 24 | 28 | 24 |
| 14 | sites of Civil War | 98 | 100 | 98 |
| 15 | nice fishing place in Michigan | 6 | 12 | 6 |
| 16 | fisherman wharf with delicious restaurants | 32 | 34 | 32 |
| 17 | landmark in San Francisco | 62 | 62 | 62 |
| 18 | cheap holiday tour with friends | 38 | 48 | 38 |
| 19 | national park with nice tour guide for driving | 38 | 38 | 38 |
| 20 | beautiful valley and waterfall | 90 | 94 | 88 |
| 21 | best place for camping | 100 | 98 | 100 |
| 22 | the most romantic place | 62 | 60 | 62 |
| 23 | a place most suitable for barbecue | 30 | 48 | 30 |
| 24 | highly rated parks for picnics | 68 | 66 | 64 |
| 25 | mountain climbing destinations | 86 | 88 | 84 |
| 26 | the best resort in Alabama | 22 | 22 | 22 |
| 27 | the best city museum to visit | 50 | 82 | 48 |
| 28 | popular ski destination in winter | 96 | 98 | 96 |
| 29 | a place best to watch sunset and not crowded | 86 | 82 | 84 |
| 30 | seaside place with most delicious fish to eat | 44 | 42 | 44 |
| 31 | good family tour destinatioon with high rating | 56 | 54 | 50 |
| 32 | recommended barbecue place | 68 | 62 | 68 |
| 33 | most beatuiful beaches to swim | 92 | 92 | 92 |
| 34 | historical significant attraction | 94 | 100 | 94 |
| 35 | biggest art museum | 28 | 30 | 28 |
| 36 | national park with highest rating | 32 | 40 | 32 |
| 37 | popular sites to watch maple leaves | 62 | 70 | 62 |
| 38 | attraction with hot springs | 98 | 98 | 98 |
| 39 | popular snow watching sites | 76 | 100 | 76 |
| 40 | best amusement park with many comments | 22 | 40 | 22 |

Table 4: Precision rate (Relevance$\geq$ 3) of different Query Expansion model

## 5.2 Learning to Rank Results

### 5.2.1 Pair-wise BERT Reranker

We use Pair-wise BERT Reranker as one of the features in our final pipeline, but it should be trained independently. We tried to trained vanilla BERT on different train/test

split, but the results are really different. We compared the performance of Baseline and our trained BERT model, and the different performance in evaluation metrics are shown in Fig 10 and Fig 11.

| | name | AP(rel=3) | P(rel=3)@10 | ndcg | ndcg_cut_5 | ndcg_cut_10 | mrt |
|---|---|---|---|---|---|---|---|
| 0 | BM25 | 0.527218 | 0.6250 | 0.778395 | 0.658551 | 0.698940 | 26.180424 |
| 1 | BM25 >> VBERT | 0.464210 | 0.5375 | 0.744747 | 0.576042 | 0.611716 | 339.892923 |

Figure 10: Vanilla BERT Result 1

| | name | AP(rel=3) | P(rel=3)@10 | ndcg | ndcg_cut_5 | ndcg_cut_10 | mrt |
|---|---|---|---|---|---|---|---|
| 0 | BM25 | 0.474105 | 0.58 | 0.764226 | 0.630902 | 0.680780 | 22.102478 |
| 1 | BM25 >> VBERT | 0.504119 | 0.64 | 0.761744 | 0.695832 | 0.688307 | 292.123758 |

Figure 11: Vanilla BERT Result 2

From the evaluation metrics we can see that all BERT reranker performs worse in overall ranking (NDCG) criteria for both training sets. However, for the first training, all evaluation metrics are lower than the baseline, while the second training gives most metrics higher than the baseline. Temporarily, we will pick the first training model into our Learning to Rank model, but it's still worth discussing this effect in the next chapter.

### 5.2.2 Feature weight analysis

We have 8 features in total, which includes BM25, TF-IDF, Number of Reviews(log scale), Rating, State, Sentiment analysis on negative reviews, Sentence Transformer similarity, and VBERT reranker. We will define the first 5 features as our "basic_pipeline" because they reflects the basic database feature (Bo1 Query Expansion model is also added here). For the remaining 4 features, we will add them separately on the "basic_pipeline"

to see their independent effect, and then add all of them into "pipe_baic+ALL features" to see the overall effect. The evaluation metric is shown below.

| | name | AP(rel=3) | P(rel=3)@10 | ndcg | ndcg_cut_5 | ndcg_cut_10 | mrt |
|---|---|---|---|---|---|---|---|
| 0 | BM25(Baseline) | 0.527218 | 0.6250 | 0.778395 | 0.658551 | 0.698940 | 21.523456 |
| 1 | pipe_basic(5 feature) | 0.567570 | 0.6500 | 0.796696 | 0.704020 | 0.720504 | 76.003694 |
| 2 | pipe_basic+BERT | 0.564231 | 0.6375 | 0.795791 | 0.702084 | 0.712919 | 388.308150 |
| 3 | pipe_basic+Sentiment | 0.576403 | 0.6500 | 0.803166 | 0.735688 | 0.730244 | 127.150704 |
| 4 | pipe_basic+SentenceTransform | 0.566731 | 0.6875 | 0.806556 | 0.768570 | 0.757627 | 832.408216 |
| 5 | pipe_baic+ALL features | 0.577326 | 0.7000 | 0.810882 | 0.765056 | 0.763536 | 1201.799446 |

Figure 12: Evaluation metrics for all pipeline combination

It can be seen from the figure that, the basic 5 features have already provided a large improvement on our baseline model. Each time we add a new feature to it, the ranking performance will be further improved. Finally, the "pipe_basic+ALL features" combined all these superior features and get the highest evaluation score in almost all kinds of metrics. However, it's still worth noticing that the Sentence Transform feature and BERT is very time consuming.

We can also see the ranking performance of these features independently.

| | name | AP(rel=3) | P(rel=3)@10 | ndcg | ndcg_cut_5 | ndcg_cut_10 |
|---|---|---|---|---|---|---|
| 0 | BM25 | 0.510320 | 0.6250 | 0.780160 | 0.654993 | 0.684473 |
| 1 | TF-IDF | 0.573852 | 0.6500 | 0.799244 | 0.708145 | 0.718224 |
| 2 | revienum | 0.480125 | 0.5000 | 0.761884 | 0.583944 | 0.605530 |
| 3 | overall_rate | 0.525961 | 0.6250 | 0.773886 | 0.606003 | 0.659752 |
| 4 | pos_reviewLen | 0.547511 | 0.6750 | 0.789753 | 0.718946 | 0.710416 |
| 5 | State | 0.508380 | 0.4875 | 0.768920 | 0.600673 | 0.600827 |
| 6 | Sentiment of neg_review | 0.499994 | 0.6000 | 0.763875 | 0.622076 | 0.635395 |
| 7 | Sentense transformer | 0.534188 | 0.6750 | 0.786907 | 0.698437 | 0.707197 |
| 8 | VBERT | 0.484065 | 0.5500 | 0.761977 | 0.596652 | 0.624728 |

Figure 13: Evaluation metrics of each independent feature

We can plot the weight value of each feature in our FastRank Coordinate Ascent model.

For comparison, I trained another Random Forest model and plotted its feature importance. The bar plot is shown below:

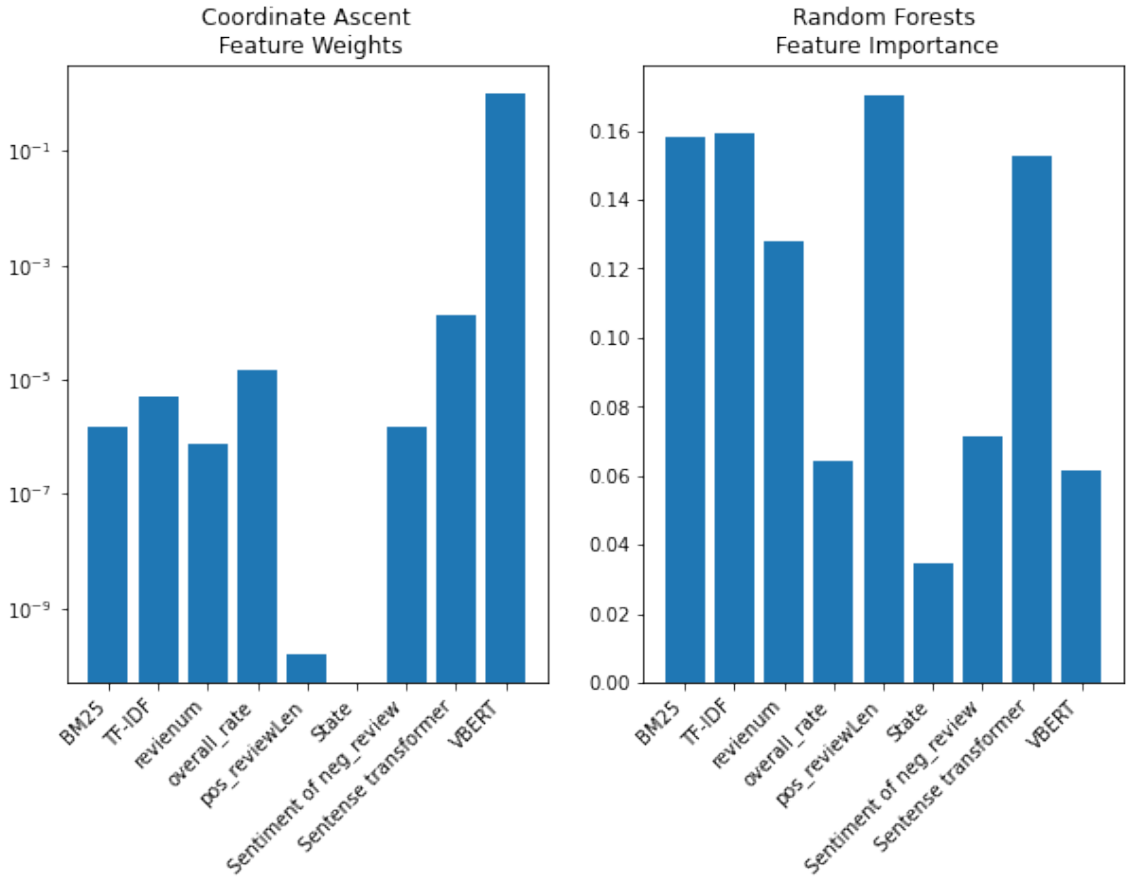


Figure 14: Featrue weight comparison between Fast Rank and Random Forest

# 6 Discussion

Generally speaking, we did quite a nice job on improving the overall performance of our Tourist Destination Retrieval System. For end-users, our retrieval system can now understand most of the abstract expressions, and do effective reranking based on user's requirement. It also beats our Baseline model in all aspects. For evaluation metrics like MAP, Precision, NDCG@5 and NDCG@10, the improvements are over 10%, which ex-

ceeded our expectation. The Query Expansion method mainly improved the performance of MAP and Precision, and the Learning to Rank method mainly improved our NDCG performance, especially for the TOP5 and TOP 10 results. Now I'm going to explain these factors in detail.

## 6.1 Query Expansion

Both of our Query Expansion model out performs our baseline. Bo1 Query Expansion has the best performance of Precision (AP/P@50) and overall ranking (NDCG). However, SDM model outperformed Bo1 QE in the high ranking results (NDCG@5/@10). This result makes sense, since SDM is able to rank the results higher if they contain close query words like "national park", and Bo1 QE can efficiently search similar results of the top ranking ones.

Based on the Precision table of each query (Table 4), We may found that Bo1 QE did a better job than other two models when retrieving queries like "must-see attraction in New York" or "popular snow watching sites". The reason is that the keyword "must-see" is hardly to be directly retrieved, but "must-see attractions" may share some other keywords like "famous" in its documents. Also, when "snow" can't be found in documents, we can still use keywords like "mountain" or "ski" provided by Bo1 QE to guess some possible results.

However, we can find that not all precision values are improved by QE. For example, BM25 has 90% precision rate for "play golf in California", but Bo1 Query Expansion only has 52% precision rate. If we look into the reassigned weight of this query, we can find some weird term like "pasatiempo" (maybe the name of a famous golf club) was given a extremely high weight, and it misled our system to search for some hotels with these weird term and finally reduced the accuracy. Also, for queries like "romantic place" which we expect our query expansion to work well previously, there's no actual improvement.

Therefore, we can conclude that Bo1 Query Expansion model is very dependent on the searching quality of the initial top results. It can do quite a good job in most cases, but if some keywords shared in top results didn't reflect the query's meaning properly, then the accuracy rate may be significantly influenced. Therefore, we think that Bo1 Query Expansion model can be combined with another model which can effectively retrieve the best top matches to improve its stability.

For SDM model, the improvement is not very evident. We think the possible reason is that it's highly dependent on the structure of query. If we have some phrases in query, then it may become helpful. But for queries that are only composed of single keywords, the improvement may not be evident.

## 6.2 Learning to Rank

### 6.2.1 Pair-wise BERT Reranker

The BERT Reranker is trained independently, so it's reasonable to evaluate it first before putting it into our Learning to Rank model. As the results shown before, different train/valid/test split can cause very different test results. Depending on different selection of random seed, the evaluation metrics can be either higher or lower than the Baseline. This result is far from our expectation. We both expect BERT to outperform the baseline model even if we change the random seed, but the result is showing that the model still don't understand what we are doing after 1 hour's training. This indicates that our model has a very poor robustness. Here are some possible reasons:

- The quality/criteria of the annotations may vary among queries. Even if the criteria is designed and agreed by both of us, it may still be different in practice, which may mislead the BERT model.
- The data field we used is too limited and simple. In order to reduce the memory cost, we only trained the pair relationship between query and positive review's title. However, review titles do not necessarily contain useful information about user's demand.
- The size of the training data can also affect the performance of a BERT model. In our project, we only choose 50 results for each of our 40 queries to annotate, which is not enough. A larger training dataset is required to provide more information for the model to learn from, which can lead to improved performance.
- Particular kinds of queries may not be trained/tested. In order to improve the diversity of queries, we tried not to duplicate when creating our queries. However, our query set is very small, which means it's possible that abstract queries like "romantic place" are put in test set, and concrete query like "beach in Hawaii" are

put in train set. This may cause large difference between train result and test result, and lead to serious problems of underfitting and overfitting. Therefore, we may need more queries to narrow down this difference.

- The hyperparameter and choice of rankers of BERT model can also be reasons for low performance. Maybe Vanilla BERT is not a good enough ranker for our dataset to train and other rankers and transformers will have better performance.

### 6.2.2 Feature weight analysis

From Figure 12 we can find that our basic 5 features already improved the overall performance a lot, and the additional 3 features provides improvements on different kinds of evaluation metric.

The largest contributor is the sentiment analysis model, especially in MAP, Precision and NDCG. It proves our hypothesis that the sentiment score of negative reviews can be an useful weighting factor on the relevance score of these negative reviews. If users provide strongly negative feedback that are closely related to our query, then its ranking should be changed lower accordingly.

Our sentence transformer model mainly contributes to the Precision, NDCG@5 and NDCG@10 metrics, but it doesn't outperform the MAP metric. It proved that transforming our query and documents into vectors is very powerful in our project. Even if the vector only has 384 dimensions, it successfully represents the feature of text. For example, "climbing mountains" and "hiking" can be considered very similar by this model, but traditional model won't think they are related. Therefore, we can conclude that sentence transformer model is very powerful in reranking, hence leading to the highest scores of NDCG@5/@10.

However, if we took a look at Figure 13, we may find that the features independently didn't perform well in reranking. For example, if we add sentiment analysis into pipeline, it can bring a large improvement. But if we only use sentiment relevance score for reranking, it will give the lowest MAP metric, and relatively low NDCG and Precision than the baseline model. It gives us the inspiration that we can't determine good/bad feature independently. Instead, we need to consider the correlation among features, and make best combination to get the "best" pipeline. To improve the interpretability of our model, we need to do more research.

The simplest way of checking the contribution of each feature is by building a Feature

weight comparison bar plot, which is shown in Fig. 14. We can see that the function of vBERT reranker differs a lot between Fast Rank and Random Forest: vBERT is the most important factor in Fast Rank, but almost the least important factor in Random Forest. It also confirmed our conclusion above about vBERT that there might be a serious underfitting and overfitting for our BERT model, making its performance not very stable. Also, the positive review length's importance is significantly different between these 2 models. It's very likely that the Random Forest overfits this feature to make it "the most important factor".

For the remaining features, their weights are still similar to our expectation and conclusions above. For example, Sentence transformer and Sentiment analysis is really powerful in both training models, and the State feature has a very small weight in both models. The possible reason is that we only have 5 out of 40 queries that contains the State name, and it's very likely that these queries didn't occur in our training set. If we change a different random seed, the importance of State name will also significantly change. Therefore, we can expand our query set to better see the real effect of this feature. Alternatively, we can turn the State name as a threshold instead of a feature to prevent the influence of training set's difference.

# 7 Conclusion

In conclusion, this project developed a tourist attraction information retrieval search engine that aims to improve the search experience for users seeking information about tourist attractions. Our approach does not rely on personal user information and instead focuses on expanding abstract queries using query expansion models and using a combination of BM25 and machine learning models to rerank the results. During the preprocessing stage, we filtered and indexed certain fields in the dataset and used Terrier's default settings for stemmer and stopword removal. For the retrieval stage, we used the default parameters for BM25 as our baseline and explored the use of query expansion and learning to rank techniques to improve performance. We compared two query expansion models and found that the Bo1QueryExpansion model performed better in terms of MAP and NDCG. We also implemented the BERT model on different training/test data and with different rankers for learning to rank and observed limited improvement and result difference due to different train/valid/test splits. This indicates the trained BERT model is low in robust-

ness. Two outstanding parts in learning to rank section sentiment analysis and sentence transformer. Both improve the overall performance greatly.

The process of building a pipeline is like building blocks. To illustrate, the performance of the trained BERT model alone is not good or even worse than the baseline, but adding features can bring a huge 10% performance leap overall. This is not only true for rerankers like BERT, but also for the union and design of features. Some features are not ideal when running alone, but once they are combined, they will improve the overall performance. This interesting fact made us realize that IR itself is a whole, and individual features cannot be viewed in isolation and should be considered as a whole when designing.

# 8 Other Things We Tried

## 8.1 Bert using knrm as Rankers

Besides Vanilla BERT we used above, we also want to discover how BERT will perform given different rankers. So trying knrm as an ranker and keep using bert as trainer, we train bert on the same dataset and after two hours' training, the results tend to be lower than Vanilla BERT before and even the basic BM25 method as shown in Fig 15.

| | name | AP(rel=3) | P(rel=3)@10 | ndcg | ndcg_cut_5 | ndcg_cut_10 | mrt | AP(rel=3) + | AP(rel=3) - | AP(rel=3) p-value | ... | P(rel=3)@10 p-value | ndcg + | ndcg - | ndcg p-value | nd |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | BM25 | 0.474105 | 0.58 | 0.764226 | 0.630902 | 0.680780 | 19.459105 | NaN | NaN | NaN | ... | NaN | NaN | NaN | NaN | |
| 1 | BM25 >> knrmBert | 0.503926 | 0.60 | 0.743255 | 0.577769 | 0.608497 | 485.312852 | 2.0 | 3.0 | 0.459858 | ... | 0.846643 | 2.0 | 3.0 | 0.475083 | |

Figure 15: knrm Bert Result

## 8.2 Failure of doc2query Implementation

We also try to use doc2query to convert a document into a query that can be used to retrieve and rank similar documents. However, after building up the doc2query pipeline with provided t5-base model, the indexing step turns out to be wrong with compiling error. After one day debugging, we found that the indexer pipeline is not being applied correctly to the dataset, resulting in an error when creating the indexref and when attempting to retrieve and rank documents using the indexref. The root is the format of our data passed

to the doc2query component do not match the requirements, but we are not able to crawl these required data again. One lesson that can be learned from this failure is the importance of properly setting up and executing each step based on documentation. It is important to ensure that all components are being applied correctly and that the necessary data and arguments are being passed to each component in order to avoid errors and ensure the pipeline is running smoothly.

## 8.3 Failure of Neural Network Query Expansion

After trying the Bo1 QE and SDM model, we are still not satisfied with the performance on precision. Only 60% retrieved documents are related, and it still performs very bad on some queries like "national park with convenient public transport" (Only 8%). Under this circumstances, any reranking method is useless, because we don't have enough stuff for it to rerank. That's why we tried BERT-QE, a Query Expansion model based on BERT (https://github.com/cmacdonald/BERT-QE). However, directly using this model didn't bring us much improvement because it was not trained on travel dataset. If we input a query like "play golf", it will return new queries that is about "football" or "baseball". We failed again when we were trying to train it, because we didn't have enough train data that is labeled. What we only have is the labeled pair between query and documents, but the BERT model didn't allow us to do such pair-wise training. Also, it's too time-consuming to tune a standard BERT model. Therefore, we have to abandon this method, and use Bo1 QE instead.

## 8.4 Other useful features but not included

Actually we tried over 40 different features and 5 pretrained model in our project. Most of them have a positive impact on the relevance score, but we have to pick 8 most representative and interesting ones from them in our final pipeline. These features are usually basic but helpful, for example, whether or not the Description matches its Reviews can be an effective way to filter some invalid results. The more the Description matches the Review, the more likely it will be a good search result. Also, its matching score can be precomputed. It's really hard to pick "best 8 feature combination" from so many features, because different combinations can bring unexpected boost or decrease. If we have enough time, we may find a better combination that includes all these features.

# 9 What You Would Have Done Differently or Next

The first thing is to reorganize our dataset. Our dataset still needs more preprocessing steps to improve the retrieval quality. For example, some tourist destinations are already closed, or repeated by another page. When organizing the data structure, we should first check the documentation of the models we want to apply to make sure the data structure and parameters could be transferred properly.

Secondly, we need to be more careful about neural network models. We didn't notice the low robustness of our BERT model until we started to draft the report. It reminded us to consider whether or not our train/test data size are reasonable before we do the training, and to do a robustness check each time after the training. Neural network is powerful but dangerous. It's very likely to overfit and underfit in a small dataset.

Thirdly, we should find a efficient method to evaluate each feature. As mentioned above, we can't determine a feature based on its independent performance. Manually selecting features are very time-consuming and may cause overfitting problems. Therefore, finding a better method to select feature can be very helpful to our model's performance.

Finally, we can add a new feature that may improve our retrieval quality further, which is the Named entity recognition model. As mentioned above, geographical information is usually more important than other words if user mentioned them in their query. We can use a Named entity recognition model to find these specific named entities in the text. For example, a pre-trained named entity recognition model could be used to identify locations or organizations mentioned in the text, and score these keywords independently.

# 10 Reference

[1] Xiang, Z., Gretzel, U. (2010). Role of social media in online travel information search. *Tourism Management*, 31(2), 179-188.

[2] Wöber, K. (2006). Domain specific search engines. In D. R. Fesenmaier, K. Wöber H. Werthner (Eds.),*Destination Recommendation Systems: Behavioral Foundations and Applications.* Wallingford, UK: CABI.

[3] Ricci, F., Wöber, K.W., Zins, A.H. (2005). Recommendations by Collaborative Browsing.*ENTER.*

[4] Fodness, D., Murray, B. (1997). Tourist information search.*Annals of Tourism*

*Research* 24: 503-523.

[5] Kanellopoulos, D. N. (2008). An ontology-based system for intelligent matching of travellers' needs for group package tours.*International Journal of Digital Culture and Electronic Tourism*, *1*(1), 76-99.

[6] Werthner, H., and S. Klein. (1999). *Information Technology and Tourism: A Challenging Relationship.* Vienna: Springer.

[7] Evensen, S., Feng, A., Halevy, A., Li, J., Li, V., Li, Y., ... Wang, X. (2019, May). Voyageur: An experiential travel search engine. In *The World Wide Web Conference* (pp. 3511-5).

[8] Anindya Ghose, Panagiotis G. Ipeirotis, Beibei Li, (2012) Designing Ranking Systems for Hotels on Travel Search Engines by Mining User-Generated and Crowdsourced Content. Marketing Science 31(3):493-520. https://doi.org/10.1287/mksc.1110.0700