The Report committee for Sonia-Roxana Marginean Certifies that this is the approved version of the following report:

VoyageWithUs: A recommender platform that enhances group travel planning

APPROVED BY
SUPERVISING COMMITTEE:

_____
Christine Julien, Supervisor

_____
Suzanne Barber

VoyageWithUs: A recommender platform that enhances group travel planning

by

Sonia-Roxana Marginean, B.E.

Report

Presented to the Faculty of the Graduate School of
The University of Texas at Austin
in Partial Fulfillment
of the Requirements
for the Degree of

Master of Science in Engineering

The University of Texas at Austin
December 2016

VoyageWithUs: A recommender platform that enhances group travel planning

by

Sonia-Roxana Marginean, M.S.E.

The University of Texas at Austin, 2016

SUPERVISOR: Christine Julien

Group travel planning poses unique challenges such as choosing hotels, restaurants and venues while catering to everyone's wants and needs, or sharing trip itineraries and artifacts among trip participants. State of the art travel planning applications such as Yelp and TripAdvisor, while integrating with social networks and making recommendations, don't offer recommendations for specific groups of travelers. On the other hand, while TripCase offers trip planning capabilities and email sharing, it doesn't offer a full interactive travel planner that allows groups to contribute to the travel planning process. This report proposes an approach to making personalized group travel recommendations based on hybrid recommendation techniques that aggregates individual recommendations to find common ground between trip participants. This is achieved by designing a recommender system that uses data from a location based social network(LBSN) and makes recommendations based on the trip location, then refines them by applying incremental filters which are responsible for incorporating user preferences, similarity to other users and user context. Finally, it takes the generated recommendations for each trip participant and ranks them such that the items most highly ranked are the ones most likely to fit everyone's preferences. The rationale for choosing a hybrid recommender system is to address common issues such as the cold start problem, where the quality of the recommendations is

affected by either too few reviewers for a certain point of interest(POI) or too few reviews generated by trip participants. These issues, along with a coverage of related work is detailed in the first part of this report. In order to make the applicability of the recommender more tangible, I integrated it into a proof of concept mobile application that also allows travelers to collaborate and share travel planning artifacts, and generates itineraries based on the recommendations made. The recommender accuracy was measured against recommendations made by state of the art applications, while individual filters were evaluated using commonly used metrics. The recommender was tested in a series of relevant scenarios proving the effectiveness of the approach in making group travel recommendations, versus individual recommendations generated by other applications.

Table of Contents

Chapter 1: Introduction

Travel planning has moved in the past decade from travel agencies into the online environment and, more recently, into the mobile environment. The first phase of travel planning, namely research, is strongly influenced by crowd-sourced information such as ratings and reviews on travel websites and social streams. This creates a need for mobile travel planning tools to augment their existing functionality and incorporate user's opinions and preferences, the so-called sentiment analysis, when making travel recommendations in order to assist the travel planning process.

Another aspect that seems to be ignored by travel planning apps is catering to groups of travelers. Traveling is more often than not a social activity, where people travel in pairs or groups. While some apps offer the ability for trip members to participate in the travel planning process, none of them offer personalized suggestions for group of travelers such as hotel, venue and restaurant recommendations based on the common interests and preferences of the specific group of users that is traveling together.

This paper proposes a new approach to group travel planning, by creating a mobile travel planning platform that makes personalized group travel recommendations and generates travel itineraries for those groups of travelers. The proposed approach uses a hybrid recommender system for making group specific recommendations and shortest path algorithms for generating travel itineraries. It also employs trip and artifact sharing to support user collaboration.

1.1 Motivation and Objectives

According to the US Travel Association[1], the US travel industry has generated over $2.1 trillion in 2015, with $650 billion accounting for leisure travel. Moreover, the US travel industry has seen a 5% increase in the past year. When surveyed[2], 87% of travelers responded that they perform the bulk of travel planning online, with 43% of them reading online reviews and 70% documenting their trips on social networks. As travel planning moves from the traditional brick and mortar agencies to the online travel sites and users move to mobile platforms, 30% of travelers use mobile apps for their travel planning.

These statistics indicate that more of the travel planning process is moving to mobile, with a strong influence from social media, which opens the door to new opportunities that make use of social streams to enhance the mobile travel planning process.

With leisure travel accounting for a third of total tourism, travel in pairs, with family or groups is a natural consequence. When confronted about the challenges of collective travel, these vary from difficulty in communication to not being able to satisfy everyone's time/budget constraints or their preferences. Out of the top ten most pressing challenges[3], I have identified three that would be addressable in the context of a mobile travel planning application:

1. Communicating trip details within a group, via trip and trip artifact sharing.
2. Working with different budgets, by aggregating multiple users' constraints in making travel recommendations.

---

[1] https://www.ustravel.org/answersheet

[2] http://infographicsmania.com/online-travel-statistics-2012/

[3] http://blog.makeitsocial.com/misc/the-top-ten-challenges-of-travelling-as-a-group/

3. Addressing different preferences for activities, accommodations, destinations, by computing an affinity model between the users' preferences.

Putting all this together and my passion for travel, my proposal is to build a mobile platform for travel planning to capitalize on the massive amounts of travel ratings and reviews available via Location Based Social Networks such as Yelp or TripAdvisor, and address the shortcomings of traveling in groups.

My objectives for this report are:

- To research different strategies for performing personalized recommendations for groups of travelers.
- Design a recommender architecture focused on group travel.
- Deliver a proof of concept mobile application that incorporates the recommender architecture and travel planning tools, to make group focused recommendations and generate appropriate itineraries.
- Measure the effectiveness of the chosen approach both by using quantitative methods, like accuracy, coverage and novelty, and qualitative methods such as real user input.
- Refine the approach to minimize poor recommendations by incorporating action feedback into the learning agents.

1.2 State of the Art

With a plethora of travel planning apps that are emerging daily from the app stores, I often end up with multiple applications that serve different purposes, from research, to booking, to keeping track of flights and reservations. In order to identify which are the most frequently used travel apps and determine their suitability for group travel, I queried AppAnnie[4] and chose the best performing apps from each of the following categories.

Travel Research Apps

In this category, I chose travel apps that help in the initial research phase of the travel planning process, namely when users search for hotels, things to do and read reviews. One of the most popular apps in this category is **TripAdvisor**. With a huge worldwide user base and an excellent feature vector for their Points Of Interest (POIs), TripAdvisor users can filter results by contextual criteria such as: time of year, suitability for a couple, a group or a family. This filtering is made possible by the existence of a set of numeric and boolean attributes that each POI possesses, the so-called feature vector mentioned above. However, it offers little personalization based on individual user profiles and no option to aggregate multiple profiles. Although my profile has more than 200 places visited and more than 600 POIs rated, ranked or reviewed, the only recommendations I get are for certain hotel categories based on my previous stays.

Yelp also ranks high in this category, as less of a travel planning app and more of a Location Based Social Network. It is mostly suited for researching businesses and places to eat. I often use it while traveling in the US, as most of the countries I visited do not have a strong Yelp presence. While Yelp offers more context to researching POIs (like the ability to filter by opening hours and the

[4] https://www.appannie.com/indexes/all-stores/rank/aggregator/?month=2016-07-01&country=US

ability to zoom in on a certain map area) and personalized recommendations (mostly collaborative-based), it also lacks a group recommendation solution.

Travel Booking Apps

As the name says, this app category is mainly used for booking hotels, flights and travel packages, but also for window shopping and price comparison. At the top of the travel booking apps stays the giant booking engine **Expedia,** who owns Travelocity and Homeaway and other travel booking engines and is one of the few to offer their EAN dev API for free. While their recommended packages are often subject to paid advertising, Expedia offers some flexibility by allowing groups of travelers to book multiple rooms and cater to specific room arrangements. Much like TripAdvisor, it can filter based on user constraints and preferences, but again it falls short on making personalized recommendations.

Travel Planning Apps

This category of apps contains apps that are widely used to create and keep elaborate travel itineraries. Two notable mentions go to **TripCase** and **Travefy.** TripCase is the longer standing app that allows to combine different trip artifacts like flights, hotels, packages and rental cars, integrating with booking engines and airlines to retrieve trip details from booking codes, as well as integrating with flightstats to keep track of flight statuses. Despite its usefulness, it has no intelligence built in with respect to recommendations or group travel. Sharing an itinerary means sending an email to someone who can "follow" your trip, but there are no import features per se and no way to truly collaborate on a trip itinerary.

Travefy is a new-comer app launched in 2016 that focuses on group travel. It allows trip participants to contribute to building the trip itinerary, add artifacts and even split the bill. It incorporates a chat platform that allows group

communication, but it also fails short on making group or any other type of POI recommendations.

My proof of concept app incorporates elements of travel planning from TripCase and Travefy, enhanced with POI recommendations based on user feedback, group preferences, contextual information and given constraints.

Chapter 2: Related Work

This chapter summarizes the main takeaways from my research into recommender architectures, the use of social streams and LBSNs in recommenders and the application of recommender systems in travel planning. In order to better understand the mechanisms used in recommender systems, I have conducted research into the following areas: general recommender systems, location based recommender systems, and contextual recommender systems. I then looked into how streams from Location Based Social Networks can be integrated with recommender systems to add context to recommendations and how they can be used in collaborative-based recommenders. Finally, I researched travel planning papers that had as their main focus generating itineraries based on context, constraints, and recommendations.

2.1 Recommender Systems and the role of Social Networks

According to the literature [2,3,6], recommender systems can be grouped into three main categories, based on the strategies they employ to make recommendations:

1. Content-based recommenders use the user's past experiences in order to make future predictions. The advantage of this approach is that it incorporates feedback from the user based on their history and preferences, predicting to a higher degree their future preferences. The downside is that content-based recommenders are subject to the cold start and over-specialization issues described below.
2. Collaborative-based recommenders rely on the user's similarity to other users and make predictions based on user affinity models. The authors of [3] start from the premise that if users were similar in the past, they will be similar in the future and therefore one can make recommendations for one

user based on the preferences of their kins. This type of system is ideal if I do not want to deal with item specifics, since the system only needs to know about user preferences, abstracting completely from the recommended items. Its drawback is addressing the cold start situation, when I do not know much about a specific user, therefore it is difficult to infer which users are going to be similar to them. This system is also problematic when new items are introduced.

3. Knowledge-based recommenders match factual knowledge about the recommended item with the user's needs. This is the ideal system to use in absolute cold start environments, when not much is known about a user and there are not many users in the system to begin with, since it matches objective facts to inputted user needs.

Some of the shortcomings identified in recommender architectures [2] come from insufficient user or item profile data, either because the user has not generated enough actions to be able to make high confidence prediction, there are not many users in the system for collaborative-based recommenders, or there are not enough items or item ratings in the system. The main shortcomings according to the literature [2] are:

- The new user problem or the cold start problem, present in content-based and collaborative-based recommenders. When a new user joins that has no previous history, one cannot make high confidence recommendations since she has no preference history that can match other items or other users. This issue can be addressed by integrating social network activity into the user profile, such as accessing their friends list in order to find users that are like them or their checkins to find places they have visited and liked in the past. A few papers that discuss the importance of social networks in recommender systems are [2] and [3].

- The sparsity problem is manifested when there are too few items in the recommender. This can be addressed by tapping into LBSNs such as Yelp, TripAdvisor, or Foursquare and importing new items.
- Limited content analysis goes hand in hand with the sparsity problem and is characterized by the lack of sentiment content for a specific item. This can make an item less desirable for a recommender, but can be addressed by curating content from other sources.
- Over-specialization is when the recommender is inflexible and does not have any novelty and serendipity factors built into it. It can be addressed by introducing some randomness or novelty factor when making user recommendations, regardless of user preferences.

In order to address shortcomings of each strategy, most recommender systems in the literature are hybrid [2], employing multiple strategies and sometimes adding dimensions, like user context [6] and constraints (time, location, cost) [5,9] and social network artifacts such as check ins, ratings and reviews from other LBSNs. In [3], the authors combine all three strategies using filtering and fuzzy clustering for computing user affinity.

In [2], the authors emphasize the importance of social networking in making recommendations. They start from the premise that users want to make informed decisions, therefore they seek information from past experiences of other users: friends and influencers. With the ability to make ratings and rankings and leave reviews, Location Based Social Networks (LBSNs), such as Yelp, Foursquare and now Facebook have become an increasingly powerful tool in building user trust. The paper also briefly addresses techniques for group focused recommendations:

- Merging sets of recommendations and recommending the resulting set.
- Aggregation of recommendation items and recommending only the items that are present in all individual sets of recommendations.

- Construction of group preference models, which is the technique that I will be adopting in this paper and allows aggregation based on similarity rather than perfect matches.

The authors of [4] discuss recommender systems based on cellular learning automata (CLA) for website navigation prediction patterns. The proposed innovation comes from dynamically incorporating feedback into recommendations and improving the reward-punishment criterion of the CLA, evaluating the improvement using formal metrics such as recommendation accuracy and coverage.

Some other metrics commonly used in evaluating recommender systems are discussed in [2,7]:

- Recommendation accuracy metrics - Mean Absolute Error, Classification Error, Precision and recall, ROC Curves, Rank accuracy and prediction-rating correlation.
- Coverage of system - percentage of items in the system recommended.
- Learning rate - measured by how fast the recommender converges.
- Novelty and serendipity - the ability of the recommender to make novel suggestions.
- Recommendation confidence - based on probabilities.
- User evaluation - direct feedback from user actions rating/ranking recommendations.

2.2 Location and Context based POI recommendations

The authors of [1] focus their research on the efforts made to predict future user locations based on social network check ins and make appropriate recommendations. Their approach compares three machine learning techniques: Artificial Neural Networks (ANN), Support Vector Machines (SVM), and Probabilistic Neural Networks (PNN), proving the superiority of the latter over

the former two. Their proposed architecture consists of a mobile agent that collects the data and a cloud computation service that makes the recommendations. While they mainly focus on individual preferences, they incorporate affinity with other users (friendship relationships on LBSNs) when making recommendations.

VISIT [6], the prototype for a mobile tour guide system, focuses on the importance of incorporating context in recommendations. The authors discuss the boom of LBSNs with the use of mobile devices and the rise of interactive travel as users use mobile for navigation and discovery. While most apps, like Yelp, use distance from location to make suggestions of POIs nearby, there are many other sources of context coming from both hardware and soft sensors that can help make recommendations, sensors such as the user's current location, weather, day of the week and time of day. Therefore, VISIT proposes to add elements such as: current weather, time, user preferences and social media sentiment gathered from social activity. The proposed architecture is an ANN, however the paper lacks experimental results, since the VISIT architecture had not yet been implemented.

In [7], the authors focus on predicting user checkin behavior by relying on soft sensing, namely app usage, phone calls, messages, and social media activity. Similar to VISIT [6], the paper points out the importance of context when making predictions and how location alone is not a sufficient indicator of user context, proposing a learning agent that samples soft sensor data in the background in order to be able to make suggestions readily available. They address the shortcomings of sampling hardware sensors and sampling very often, which could lead to battery depletion, however they do not discuss privacy issues arising from the use of soft sensing.

2.3 Travel planning and itinerary generator tools

The travel planning tool proposed in [5] recommends travel packages that conform to time and budget constraints by modeling the travel itinerary generation as a knapsack problem. The travel packages are composed of multiple POIs that each have a cost associated to them (which is a combination of time, money, and other constraints), and they are ranked by that cost. The recommender then iterates over the same knapsack algorithm until it converges to an optimal solution. The solution takes into account also the cost to travel between POIs and the compatibility between them (i.e., if a user likes landmarks and museums, the suggested POIs will be from the same family of POIs).

Another similar itinerary generator, this time for flight+hotel packages [10] takes a set of destinations and a time interval and uses a shortest path algorithm to find the best packages. It models the itineraries as a Traveling Salesman Problem and uses Dijkstra's algorithm for recommending the best flight options.

A more elaborate travel planning tool is discussed in [9], which takes different types of POIs from LBSNs: food, venue, and entertainment and builds personalized travel packages, similar to VISIT. It uses collaborative-based filtering to generate personalized itineraries, while employing geospatial constraints to allow their algorithms (a form of heuristic search based travel route planning) to converge to an optimal solution. Individual candidate POIs are selected based on: user feature vectors built from their LBSN profiles, location popularity, user location and time of day, which accounts for a fair amount of the user's context.

The main takeaways from the conducted research that I plan on applying in my work are:

- Recommender systems should be a hybrid of the three main recommender categories: content, collaborative, and knowledge based, in order to account for cold start and other challenges.
- Both collaborative and content based location recommenders can benefit from integrations with social streams from LBSNs like Yelp, Foursquare, or Facebook, which can provide user activity and POI rankings and ratings to address the cold start problem.
- User context and constraints are core to making personalized recommendations, as is dynamically incorporating user feedback.
- Soft sensing opens new avenues to inferring user context, but it comes with concerning privacy issues that need to be addressed.
- Recommenders designed to address group recommendations need to incorporate a form of aggregation or build group preference models.
- Most approaches for effectively computing itineraries are variations of a shortest path problem.
- In addition to recommendation accuracy, metrics such as coverage, serendipity and novelty are fundamental in evaluating a recommender's performance.

Chapter 3: Approach

This chapter details the proposed approach to building a group focused recommender system and a complex itinerary generator, with immediate application to a group travel planning mobile application. I will talk about the general methodology used to gather user and POI (Point of Interest data), build user and POI profiles, make recommendations, and interpret feedback from the users.

The available datasets will be described and an approach to selecting relevant features from the datasets will be proposed. Next, I will discuss the algorithms that were evaluated for performing recommendations and itinerary generation. I will talk about how the approach was evaluated and what metrics were used.

Finally, the technologies employed will be discussed and some of the challenges identified will be addressed.

3.1 Methodology

The approach chosen to address the group travel recommendation problem identified is to build a collaborative travel planning mobile application that allows users to collaborate in the travel planning process while making personalized group recommendations for trip artifacts and travel itineraries. For the sake of simplicity, the app will:

- Allow users to create, join, and invite their friends from their phonebook to trips.
- Allow users to add artifacts to trips, such as: flight, hotel, and package reservations from Expedia, car rentals and points of interest. The reason for choosing Expedia is that they offer their REST API free of charge.

- Be able to make group recommendations for three types of artifacts: hotels, food venues, and landmarks.
- Be able to recommend flight plus hotel packages from Expedia based on group preferences and constraints such as trip dates and price.
- Compute walking itineraries at a certain location based on preferences and constraints.

The architecture of the mobile application and the technologies that were used to build the full software stack are described in sections 3.7 and 3.8.

The POI data that was used to make recommendations was meant to be an aggregate of multiple social media and LBSN streams, such as TripAdvisor, Yelp, Facebook, and Instagram. However, the aggregation of multiple streams in realtime and generation of consistent views of the POI dataset is a complex research topic in itself, that is outside the scope of this paper. Therefore, I chose to use a static dataset from a LBSN that is publicly available, has a big enough POI and user set such that the cold start will not be an issue, but small enough such that I did not have to spend a great amount of time optimizing queries and data access. The description of the dataset and the criteria for selecting it is described in section 3.3 of this chapter.

For making recommendations I chose to build a hybrid recommender system that incorporates all the three major recommender systems: knowledge, content, and collaborative-based, augmented with three dimensions: group preferences, user feedback, and context. The recommender uses multiple levels of filtering to sieve the original set the POIs and identify only the artifacts that have a high probability to match the group preferences. The architecture of the recommender is described in detail in section 3.2 below.

In order to measure the performance of the recommender, I evaluated it against two other recommenders and computed the following metrics: Mean Absolute Error with respect to the test samples, coverage and evaluate

serendipity. The first system for comparison is a random POI generator on the chosen dataset and the second one is the Yelp recommender. Since none of them are focused on group recommendations, I generate predictions for multiple users and then merged them.

For the user profile model generator, I evaluate it by splitting the user profile dataset into a training and test set and incrementally improving the model by evaluating the model's predictions against the test dataset.

3.2 Recommender Architecture

As most of the literature discussing recommender systems suggests, the proposed recommender architecture is a hybrid one, combining elements from knowledge, collaborative, content and context based recommenders.

A recommendation agent is dispatched for each user, each time they perform a direct query to find a POI or they join a trip. The agent takes the initial set of POIs, apply the factual constraints, such as POI category and location and filter out all the POIs that do not conform to the constraints. Next, the remaining POIs are filtered twice: once through a collaborative filter that surfaces POIs that are popular with similar users and once through a filter that surfaces POIs that are recommended by the User profile model and past feedback. The union of these two sets of POIs constitutes the individual user's recommendations and are precomputed and stored in her trip profile. The individual recommendations from each trip participant are aggregated and the resulting set is filtered at the time of the ad-hoc queries based on the user's context, inferred from soft sensing (day of week, time of day, weather), or given by the users (hours and money to spend).

A visual representation of the Recommender Architecture and the data flow through the system is depicted in Figure 1.

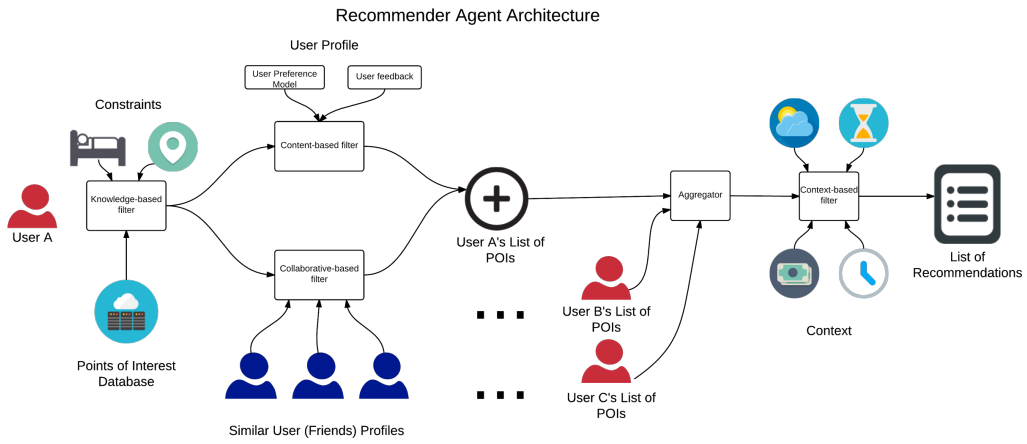

Figure 1: Recommender Architecture

3.2.1 User profiles

The user profile is based on all the recommendations the user has made and is built dynamically when recommendations are requested by the app. The reason for building the profile dynamically is that it is contingent on realtime user data and needs to be updated as the user makes new recommendations.

The agent that is built to generate the user profiles relies on supervised learning techniques. All the user sentiment data is gathered and then split into two sets: one containing 80% of the samples that are used for training and another containing 20% of the samples that are used for test purposes. The training data is then fed to a model builder that attempts to build a model using Linear Regression(LR) and Multiple Linear Regression(MLR) using individual features and feature sets from the yelp dataset. The proper model is then chosen during the implementation phase with accuracy and performance in mind. The 20% test set is then used to make predictions and evaluate the accuracy of the model. I used common accuracy metrics such as RMSLE (Root Mean Squared Logarithmic Error) to measure accuracy. The predictions are made on the ratings column, while other features are used for building simple and multiple linear regression

models. Then, depending on the accuracy, I will build a feedback loop and improve the prediction model. Finally, this model will be used to make predictions on new POIs.

A conceptual diagram of the user profile model builder is presented below.

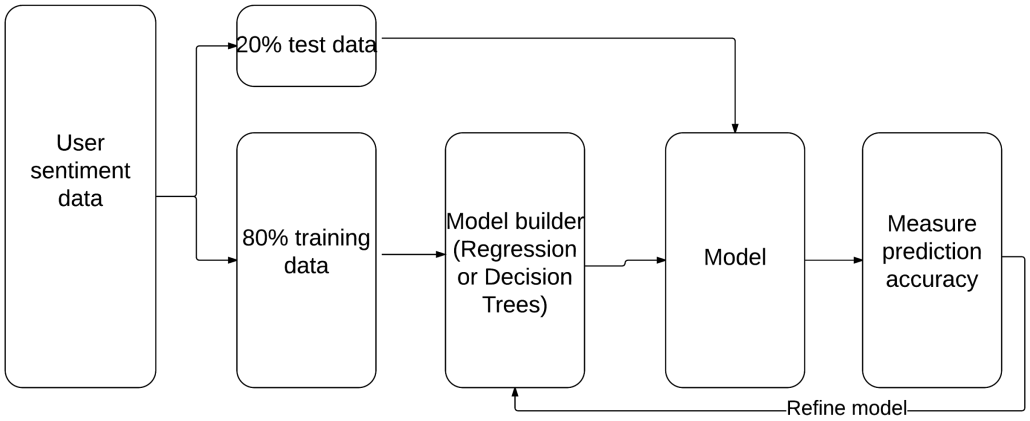

Figure 2: User Profile Builder

3.2.2 Recommender Filters

As mentioned in the previous sections, the recommender is essentially a set of cascading filters that fulfills different roles and addresses the common recommender issues that were identified in Chapter 2.

The four recommender filters and their functionalities are:

- **The Knowledge-based filter** is responsible for filtering based on constraints that the user either manually inputs as part of an in-app query or that the app infers from the trip details. The main constraint types are the POI category and the POI location. Since this is the first filter in the chain, I need to ensure that it will not constitute a bottleneck; therefore I experimented with different clustering and indexing techniques on the POI database, such that I optimize it for the knowledge-based filter queries. This filter is meant to address the cold start issue that recommenders have when there are not enough user preferences in the recommender system.

Applying this filter guarantees that at least the POIs at the trip location will be recommended to the user.

- **The Collaborative-based filter** makes recommendations based on the similarity with other users. Since the POI dataset chosen comes from a LBSN that contains info about a user's friends network, I used that to determine affinity between users. I propose a fuzzy clustering technique to extract the list of recommended POIs. Each friend in the user's network represents a cluster, while POIs have a probability of belonging to a cluster, based on the user's past preferences (their profile model). Based on that probability, I select a number of POIs that have a high degree of belonging to more than one cluster. Another strategy I tested for this filter is majority voting, where the most frequent score in the user's friends sentiments is chosen as the POI score. The collaborative-based filter addresses issues such as novelty since it will expand the recommendation pool, going outside the user's past experiences and preferences. In combination with the content-based filter, it also addresses the POI coverage issue, because the two filters are applied in parallel and their results combined.
- **The Content-based filter** also takes the list of POIs generated by the first filter and applies the user's model to them in order to determine which POIs to recommend based on the user's preferences. The user model is built first for the existing users in the dataset by splitting the dataset into two sets: training and test. I attempted building a few models, such as regression and multiple linear regressions and evaluated their accuracy before settling on one model. The resulting set from this filter is merged with the set from the collaborative-based filter to form the user's individual recommendation set, that will be kept in the user's trip profile such that it can be retrieved at a later time without having to recompute

everything from scratch, which will improve user experience and app performance.

- **The Context-based filter** is used as an ad-hoc filter for location aware queries and infers user context from the mobile soft sensors or directly get context from the user's input. What I mean by this is that the context-based filter will be activated only when the user does a nearby-type of search. At that point, the app will sample the day of week, time of day and weather and filter out POIs that are closed or not appropriate for the weather or time of day. Additional context can come directly from the user, such as cost and time available. Some of the intelligence behind this filter is contingent on the POI having attributes that allow us to perform said filtering. The attributes could be features like: outdoors, good for groups, price ranges or indicators of affordability, time to spend, etc.

A detailed implementation of each filter will be given as part of the implementation in Chapter 4.

3.2.3 Aggregator

The aggregator takes the recommendations generated for each trip participant and build a common list of recommendations, ranking higher the POIs that are popular with most participants and the items that have overall higher ratings, computed by averaging the individual user scores. Another alternative that I explored is majority voting, where a POI will only be surfaced in the recommendations if it has a majority vote from trip participants, meaning that the POI ranking will have the most frequent rating among the individual POI results. This ranking is based on a computed score, that I call the SmartTrip score, and POIs will be surfaced in descending order of this score.

3.3 Dataset Description

While researching potential data sets that I can use to mimic LBSN generated data, I came across a series of potentially good freely available ones provided by Expedia and Yelp or gathered by third parties from LBSNs. The Expedia dataset fell through because it was limited to hotels only and I wanted a dataset that incorporated other types of POIs. Moreover, Expedia does not provide access to anonymized user profiles like the Yelp dataset does, which will be useful in testing different scenarios like: user with many and few reviews, user with many and few friends. Finally it came down to the Yelp public dataset and a dataset gathered by crawling TripAdvisor pages. Both had multiple types of POIs, rating and reviews data correlated to user profiles and a fairly big set, but the Yelp dataset was superior for reasons that I will cover in the following subsections.

3.3.1 Yelp Public Dataset

The Yelp public dataset is published by Yelp as part of their data mining challenge. The current set is 3GB large without photos and contains 2.7M reviews for 86K businesses in 10 cities across 4 countries and contains data from 687K users. In addition to the reviews, the dataset contains also user profiles, tips, business descriptions, social check-ins, and photos.

The dataset contains the following object types, which are expressed as JSON objects:

- Business, containing the business' name, address, coordinates, categories, hours of operations, ratings, number of reviews. This factual data is core to the knowledge-based filter.
- Review, containing the business reviewed, reviewer ID, text of the review, rating, date and votes. Since I am not focusing on natural language processing to get user sentiment for reviews, I rely on ratings alone, combined with number of votes, to determine usefulness of the review. I

use the ratings information in the user's feedback loop and in the collaborative-based filter. However, I might want to retrieve the review text for display purposes alone.

- User, containing their names, their Yelp stats and their list of friends. Having their list of friends readily available makes this dataset valuable for building the collaborative recommendation filter.
- Check-in, containing the time and number of checkins for a certain business. I could easily use this information to make educated guesses when building the context-based recommendation filter.
- Tip, containing a textual tip from a user for a certain business, together with the number of likes for that tip. Tip information could be parsed to identify features like: appropriate for groups, families or couples.
- Photo, containing the id of the photo, associated business and some metadata fields. For this project, I am not be using the photo data provided, which is over 5GB. Rather I used a coordinates-based photo API, such as Panoramio to retrieve photos for a certain POI or location.

Each object type is stored in a separate file, so in order to be able to make complex queries across the whole dataset, such as retrieving all the preferred locations of a certain group of users, I exported the relevant data into a relational database and create appropriate indexes depending on which database operations are most frequent.

A complete description of the dataset can be found in Appendix A.

3.3.2 Choosing the appropriate dataset and designing the datastore

While the TripAdvisor dataset is fairly large and offers detailed reviews of POIs, it lacks a few features that the Yelp dataset contains. Things like: user profile, list of friends, and social check-ins made the Yelp dataset more attractive and easier to use out of the box. Moreover, the Yelp dataset is official, newer and

already cleaned up, while the TripAdvisor dataset has been created by crawling TripAdvisor pages that are already stale.

Having said that, I split the dataset into a training set and a test set, with 80% of the records going into building the individual recommendation model and 20% into testing out the derived model.

The relational database I propose is represented in Figure 3 below:

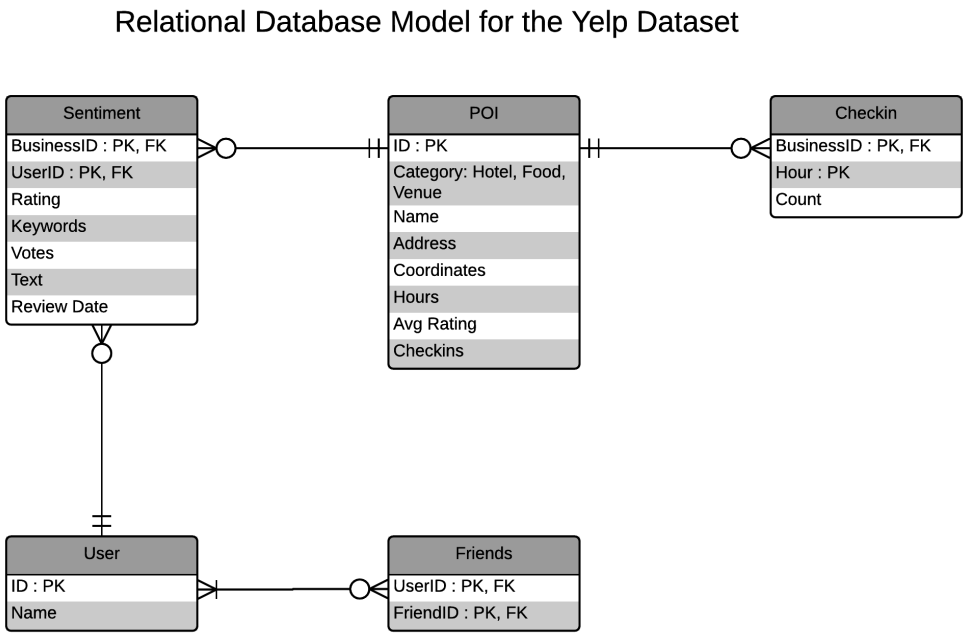

Figure 3: Relational database for the Yelp Dataset

In order to optimize the queries I experimented with indexes and clustering. Some of the optimizations I considered are:

- Clustering the POI table on location or category.
- Clustering the Sentiment table on Business or User.
- Creating secondary indexes on fields and composite fields that will most commonly be used in queries, like location and category.

3.4 Algorithms for generating itineraries

There will be two types of itineraries generated by the app: flight plus hotel packages and POI sightseeing packages. While the first type of itinerary is sourced from Expedia, based on the user's time and monetary cost constraints, the second type of itinerary is generated based on the user's current or selected location, the group recommendations that were pre-computed, and the time and monetary cost specified.

For generating trip itineraries I looked into shortest path algorithms, like variations of Dijkstra's algorithm. I built a network of POI nodes for a given area, where the nodes have a symbolic cost associated to them (combination of time to spend there and monetary cost), and each edge has a cost associated to it based on physical distance and time to traverse it. The origin node is the user's current or selected location, and the end node will be selected by the user. I am keeping not only the shortest path value but also the intermediary nodes. The algorithm only stores paths that have a lower cost than the maximum cost inferred from user constraints.

3.5 Evaluation Metrics

With respect to evaluating the approach proposed, there were multiple criteria and levels at which I evaluated performance and accuracy:

- Overall accuracy was measured by comparing the recommendations list with another recommender system, like Yelp. Since Yelp does not have group recommendations, I looked into recommendations made for different individual profiles in the same context.
- The user profile model generator accuracy was evaluated using the RMSLE metric on the test data predictions. I took the predictions ratings vector and compare it to the actual ratings vector of the test dataset.

RMSLE or Root Mean Square Logarithmic Error, which is a popular metric for Kaggle data mining competitions, is computed as:

$$\epsilon = \sqrt{\frac{1}{n}\sum_{i=1}^{n}(\log(p_i + 1) - \log(a_i + 1))^2}$$

Where:

ϵ is the RMSLE value (score) n is the total number of observations in the (public/private) data set, p_i is your prediction, and a_i is the actual response for i. $\log(x)$ is the natural logarithm of x

Notes:

- RMSLE penalizes an under-predicted estimate greater than an over-predicted estimate

Figure 4: RMSLE metric description from Kaggle[5]

- Recommender performance was measured by computing mean and average times for producing group recommendations. The goal is to be able to produce the recommendations in under 5 seconds, which is half the time the user can stay engaged without losing interest, according to UX Research[6].
- Mobile application performance was evaluated by recording application metrics, which were collected via Android Monitor.
    - Memory usage over time
    - CPU usage over time
    - Battery usage over time

3.6 Technologies

In this section I will list the technologies chosen with a brief motivation of the rationale behind choosing each of them. Each subsection is centered around one of the three tiers of the proof of concept application: infrastructure, backend and frontend.

[5] https://www.kaggle.com/wiki/RootMeanSquaredLogarithmicError
[6] https://www.nngroup.com/articles/powers-of-10-time-scales-in-ux/

3.6.1. Infrastructure

In order to minimize infrastructure concerns and not have to deal with concerns such as auto scaling, load balancing and server maintenance, I chose Google App Engine to handle the server side infrastructure to host the web application and REST API that implements the business application logic in the backend. Google App Engine also offers a generous free tier that resets its quotas daily, such that at the scale of the proof of concept application I will not have to pay for usage.

The recommender system is implemented as a set of Amazon Web Service Lambdas, again in order to avoid using a paid service tier and not worry about implementation details. The rationale for using Lambdas is that I plan on using the Python Scientific Kit (SciKit Learn), which is not supported on Google App Engine, but can be deployed on AWS lambdas.

3.6.2. Backend Technologies

For the backend application, I chose as the main language Python 2.7.3 since it is a language that allows for fast prototyping and has powerful web api and data mining frameworks.

I used webapp2 for the REST API routing layer, which is the framework that Google App Engine works with. I use Pandas and SciKit Learn for Yelp data pre-processing and building the linear regression models and making the predictions since the library has built-in adjustable models and I have previous experience using the kit.

3.6.3. Frontend Technologies

Our application has a single frontend, an Android mobile app, developed for SDK version 24 (Nougat) and backwards compatible with the oldest supported

Android version, which as of today is Lollipop (SDK versions 21-22). The language of choice for the frontend is Java 8.

3.7 Challenges Identified

While defining the approach there were several challenges identified, ranging from recommender typical problems, such as those defined in Chapter 2, to performance problems related to web api round trip times and battery usage to privacy concerns.

Two of the challenges I considered during implementation are:

- The cold start problem that I defined above. I made an attempt to get around that at first by associating a Yelp pre-existing profile to the first users of the app. I also simulated making recommendations for new users without any pre-existing sentiment data and introducing POIs without any sentiment data associated to them.
- Privacy concerns arising from the fact that the app has access to the user's location, address book and LBSN data. Our backend will never store PII (Personal Identifiable Information) such as email/password combination, but rely on social authentication such as Google and only store hashed emails for user profile matching. With respect to accessing user location and address books, Android SDK 24 already requires you to implement privacy mechanisms in your app by default.

Chapter 4: Implementation

In this chapter I detail the main implementation concerns for the main different pieces of our group recommendation mobile application proof of concept:

1. The recommender system implementation is detailed, specifying the techniques and libraries used and the tradeoffs considered during implementation.
2. The mobile application software architecture is presented, with details about the application backend REST API, the activity screens and the third party APIs interfaces.

   This chapter also covers two adjacent topics:

- The data pre-processing I performed on the yelp dataset, explaining what data I extracted and how I modeled it such that it fits our applications' needs.
- The technical limitations I have run into while developing certain software components or trying to integrate software components across different realms.

4.1 Yelp Dataset preprocessing

The Yelp dataset was first loaded into a relational database with the schema presented in Figure 3. This allowed for complex querying of users, POIs and user sentiments and for eliminating any attributes in either dataset that are not useful for the recommender. A few statistics were extracted in order to establish the basis for the test cases, in order to simulate cold start problems and other corner cases, and measure the performance of the recommender. The statistics computed are:

- Top 10 most reviewed and bottom 10 least reviewed cities.

- Top 10 most reviewed and bottom 10 least reviewed POIs.
- Top 10 most and bottom 10 users with respect to number of reviews.

For the content-based filter, since I only needed the business (POI), but needed all the POI features in order to choose the best Linear Regression (LR) or Multiple Linear Regression (MLR) model, I transformed the business dataset into a csv file, that was then imported in pandas dataframes, a data structure used frequently with Python SciKit learn.

4.2 Recommender Agent Implementation

The recommender agent was logistically split into a server side recommender and a client side recommender. The server side recommender contains three out of four filters: knowledge, content and collaborative, as well as the aggregator. The context-based filter was implemented on the client side, in this case in the mobile application. The rationale for splitting the recommender like this is the fact that the context-based filter needs user context that can only be retrieved from soft sensors on the user's device. On the other hand, the other filters and aggregator are more computational-intensive so they need more computing power. Moreover, considering the performance numbers, it is very likely that the results of the server side recommender will have to be computed asynchronously and cached.

4.2.1 Knowledge-based Filter Implementation

The knowledge-based filter simply performs a query on the POI database and retrieves the POIs that fulfill the query constraints for a given area. The area is computed as a radial area computed using a point specified by a set of coordinates and a distance relative to the point. The POI list is sorted based on proximity to the point and all POIs in the list receive the same score, 1.0 out of a

maximum of 5.0, because their only merit is being close to the coordinates specified.

4.2.1 Content-based Filter Implementation

The content-based filter takes the list of POIs generated by the knowledge-based filter and uses that to generate the pandas POI test dataframe whose ratings will be predicted by the regression model computed for a particular user. The content-based filter uses a machine learning library to perform LR and MLR, namely the LinearRegression methods in Python SciKit Learn. The model is generated and evaluated using the user's previous ratings. Since the model is computed on the fly, the model always takes into account the latest user feedback on POIs. The POIs are scores are generated by the model's prediction operation applied to the test dataframe.

4.2.3 Collaborative-based Filter Implementation

While building the collaborative filter, I approached two strategies that I tested with extensively: majority voting and clustering. In the first approach, the score of the POI is determined by performing a majority vote, meaning that the most frequent score from friends' sentiments is selected as the POI final score in the collaborative filter. In case of a tie, an optimistic approach is implemented and the highest score is selected. In case of the clustering approach, the score is computed by computing the cumulative probability that a POI belongs to a cluster and multiplying by a coefficient that translates the probability back to a score. Majority voting was the winning strategies, for reasons explained in *Chapter 5, Section 5.1.7.*

4.3 Mobile Application Frontend and Backend Implementations

The mobile application software stack presented in Figure 5 is composed of 4 tiers:

- The Web Server is implemented using Google App Engine and the NDB Datastore. This component provides the services backend and data storage for the application.
- The Web API is implemented in Python using the webapp2 framework included in Google App Engine. This component exposes the REST API described below, which is used to transfer data between the mobile app and the GAE Datastore and interface with third party libraries and APIs. All data is JSON serialized.
- Third party Services and APIs are used for authentication, location awareness, image retrieval and POI and itinerary search services.
- The Mobile Application Frontend provides the UI, interfaces with the Web API and communicates with a subset of the third party APIs like Google+ and Facebook for authentication, Panoramio for location-based image retrieval and Expedia for travel package research. The Mobile App is the container for the Recommender Agent proxy as described above and is responsible for triggering the lambda jobs that compute group recommendations on trip creation or ad-hoc.

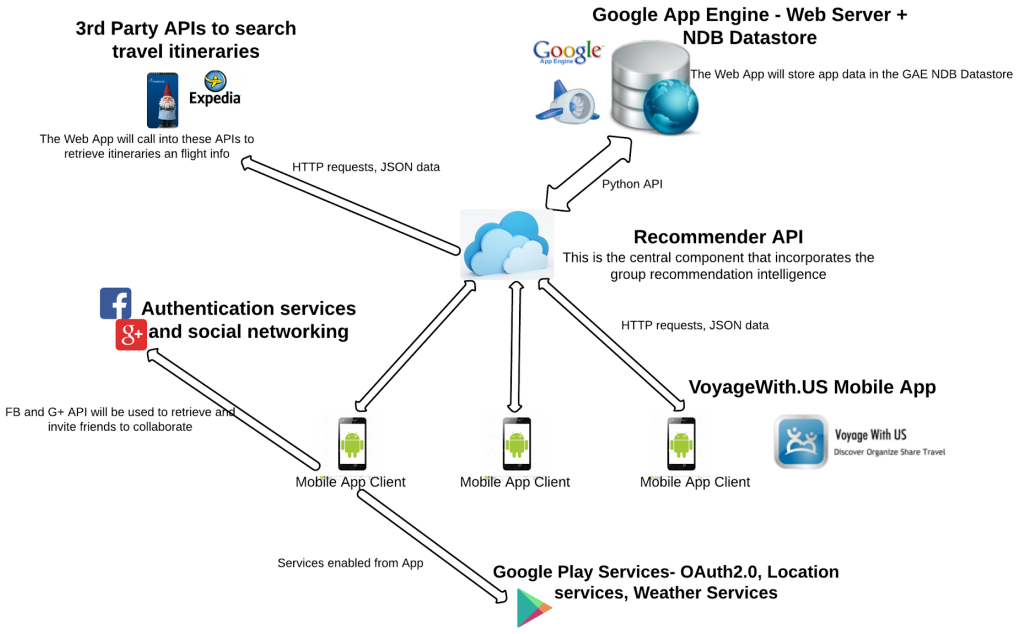

Figure 5: Proof of concept mobile application architecture diagram

4.3.1 REST API Endpoints

The REST API exposes the following endpoints, grouped into functions:

- Trip creation and management endpoints, implemented via the CRUD operations detailed in *Appendix C, Table 1*.
- Trip artifact creation and management, where artifacts can be: flights, hotels, cars, trains, cruises. These are covered in *Appendix C, Table 2*.
- User profile creation and management operations. User creation is self-managed, with user profiles being created upon first application login and user deletion and user updates performed from the backend app only. The endpoints exposed can be found in Appendix *C, Table 3*.
- Sentiment creation and retrieval operations that are used to update the sentiment database with user ratings as the user rates them in the app. For authenticity purposes, sentiments cannot be edited or deleted. A sentiment

object contains a review and/or rating, the POI ID and User ID. Sentiment endpoints are enumerated in Appendix *C, Table 4*.

- POI creation and management operations. POIs are created by calling into the API directly. Since the proof of concept app assumes that the POIs already exist and are sourced from LBSN streams, I do not allow POI creation from the app, but I expose the endpoint, such that I can add POIs at a later time to simulate some cold start problem scenarios. These endpoints can be found in *Appendix C, Table 5*.
- Recommender endpoints are used to trigger the recommender lambda jobs and compute recommendations based on given filters. Some examples of the recommender endpoints are given in *Appendix C, Table 6*. The full list of filters for the recommender endpoints can be found in *Appendix C, Table 7*.
- Trip itinerary generator endpoints, described in *Appendix C, Table 8*. The same filters that apply to recommendation endpoints also apply to trip itinerary endpoints.

4.3.2 Datastore Entities

In addition to the database storing the POIs, user checkins and sentiment data, our application also requires us to store the following type of entities in the datastore:

- Trips object that represent user created trips.
- Trip Artifacts that represent the different artifacts in the trip itineraries.
- User profiles that are associated with trip histories.

The relationships and fields stored for each type of entity are presented below in Figure 6.

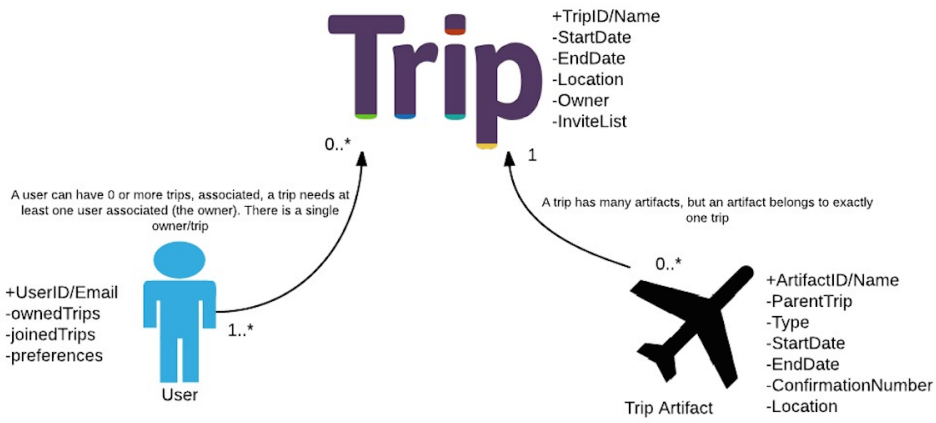

Figure 6: Relationships between Trip, User and Artifact entities

4.3.3 Mobile Application Frontend Design

The mobile application, called VoyageWithUs, was built for Android devices, using Android Studio. I targeted the Android SDK version 24 and made it backwards compatible down to version 21 of the Android SDK. The application consists of the following activities:

- Authentication activity, which allows users to login via social login, using either the Facebook or Google+ authentication API.
- Main Activity for trip management, containing multiple fragments for different views: past trips, current trips, all trips and trip invites.
- Create trip activity, used to create a trip in a specific location in a certain time interval.
- Trip view activity with fragments for different views: timeline view, artifact view, members view and recommendations view.
- Add/Modify/Rate trip artifact activity, which allows for adding artifacts such as hotels, flights, cars and cruises to trips. This activity interacts

behind the scenes with the recommender to make relevant artifact recommendations.

- Trip itinerary activity, which shows generated travel packages based on cost and time available. This activity interacts with the Expedia API for flight packages.
- Nearby activity which makes recommendations based on user context.

A few screenshots of the Mobile Application's basic functionality are included in Appendix B.

4.3.3 Third Party APIs

The mobile application uses a series of third party APIs and services to accomplish functionality such as:

- **Authentication:** The mobile application relies on Google authentication services to perform user authentication.
- **Social Networking:** The mobile app uses Google+ and the Contacts list to retrieve friends list that will be used for sending trip invites and viewing trip participants.
- **Location services:** The mobile app uses Google Play Services in order to be able to implement location awareness.
- **Panoramio** was used to enhance the mobile UX with pictures from the trip locations.
- **Affiliate Networks Integration -** The web API uses the Expedia Affiliate Network[7] and its associated API to generate trip itineraries based on given user time, location and cost constraints.

[7] http://developer.ean.com/docs/

4.4 Limitations Of The Implementation

While developing the recommender, I ran into limitations from the free infrastructure that I attempted to use, which made the full integration of the different software components challenging at best. First of all, AWS lambda does not allow for stacks larger than 500MB to be deployed in their free tier. Since the recommender was using a fair amount of Python packages, once I needed to add the POI dataframe to the stack for the content-based filter and the full database for the knowledge-based filter, I was no longer able to deploy to the free tier. Therefore, my integration testing was limited to retrieving pre-computed recommendations that I previously uploaded to AWS lambda.

Chapter 5: Experimental Results

This chapter covers the series of experiments conducted to test the results and performance of the recommender and the mobile app. The first section describes the representative test scenarios used, interpreting the results obtained. Next, I compared the results of the recommender against a random recommender and against the Yelp recommender. Finally, performance is analyzed, both for the recommender and the mobile app and some suggestions for improvement are discussed.

5.1 Test Scenarios

In order to assess the performance of the recommender, I put together a series of scenarios that I tested the recommender against. These scenarios were chosen to simulate and compare extreme corner cases, such as:

- The users with the most number of reviews traveling together
- The users with the least number of reviews traveling together
- The users with most friends traveling together
- The users with least friends traveling together

For each test case, I ran the recommender with one, two, three, four and five users, building the travel groups incrementally from the previous user set. The location chosen for all test cases was Las Vegas, and the type of POI chosen was “Hotel”, because this was the location with the most reviews and the most reviewed POI type for this location was “Hotel”. This allowed running all the test case scenarios on the same POI set. In order to reduce the number of recommendations, I restricted the area searched to 0.5km area.

5.1.1 Users With Most Reviews Traveling Together

This use case was chosen to simulate an environment where all trip participants have issued many reviews, therefore the individual profiles will likely

produce different scores and rankings for most of the POIs. Incidentally, users with many reviews also have many friends, which was reflected in the final results that can be seen in Figure 7. One interesting thing to notice from the graph below is how the sentiment changes for a POI as more people join the trip. For instance, when one or two users participate, POI 5 is one of the top rated ones(it is ranked 4 by one user and ranked 6 by a group of 2 users), but as more people join the trip, that have a less favorable score, POI 5 drops to the bottom 5 choices. One other noticeable aspect is that a POI score can never increase as more people join, because the score is computed such as the most negative individual score has the final weight in the overall score. This is due to the policy of the recommender, which is to err on the safe side and recommend POIs that all trip participants have a good sentiment for, rather than suggesting POIs that one participant loves, but another one hates. However, ranking of POIs can increase even if their overall score decreases.

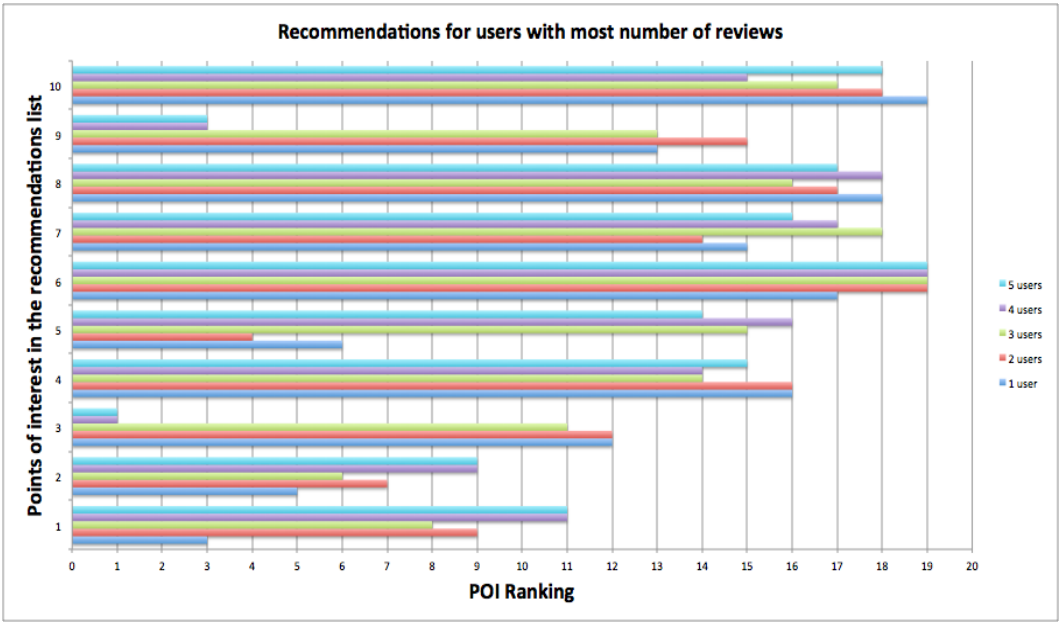

Figure 7: Recommendations made for groups of travelers with most reviews

5.1.2 Users With Least Reviews Traveling Together

This use case was chosen to simulate the cold start problem, where users have reviewed little or no POIs. The users were chosen among the bottom ten users with least reviews. Aside from User 1, which has 0 reviews and 0 friends, all the other users have 1 review and 1 or more friends. As you can see from Figure 8 below, when only 1 user is participating, since he has no reviews, all the POIs are ranked equally, with a score of 1.0, which is the default score that the knowledge-based recommender assigns to the POIs. Once another user joins, the scores of the POIs change, but since the new user has only 1 review, all the POIs will get the same score and be ranked equally. But the second user also has friends with reviews, so POI number 8, gets a lower score based on the friends' profile. Adding more users will adjust the final scores based on the same rationale. One thing to notice in this cold start scenario is that there is little variation in the ranking of the POIs since all participants have so few reviews, so most of the suggestions will be ranked by distance.

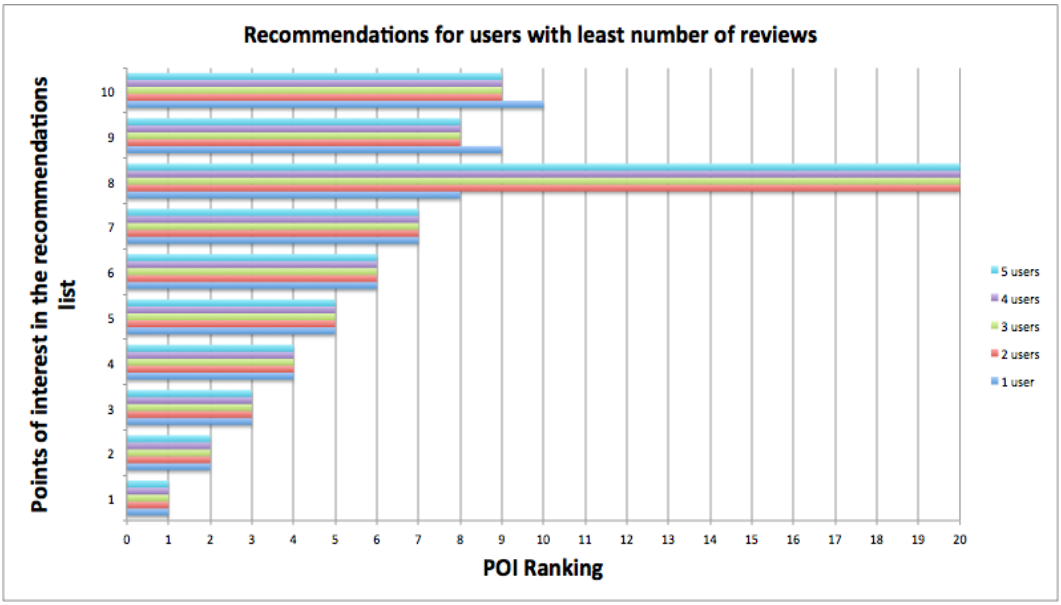

Figure 8: Recommendations made for groups of travelers with least reviews

5.1.3 Users With Least VS Most Reviews Traveling Together

I also compared the POI scores generated by groups of five users with most and least reviews as depicted in Figure 9 below. As mentioned previously, users with least reviews will produce a recommendation set with little variety, where the POIs will be ranked mostly in the order given by the knowledge-based filter, which is the distance from the selected location. 95% of the POIs have the same rank as the default location-based recommendation. By contrast, the group of users with most reviews exhibits much greater variety in POI ranking, with only 15% of the POIs having the same rank as in the location-based ranking.

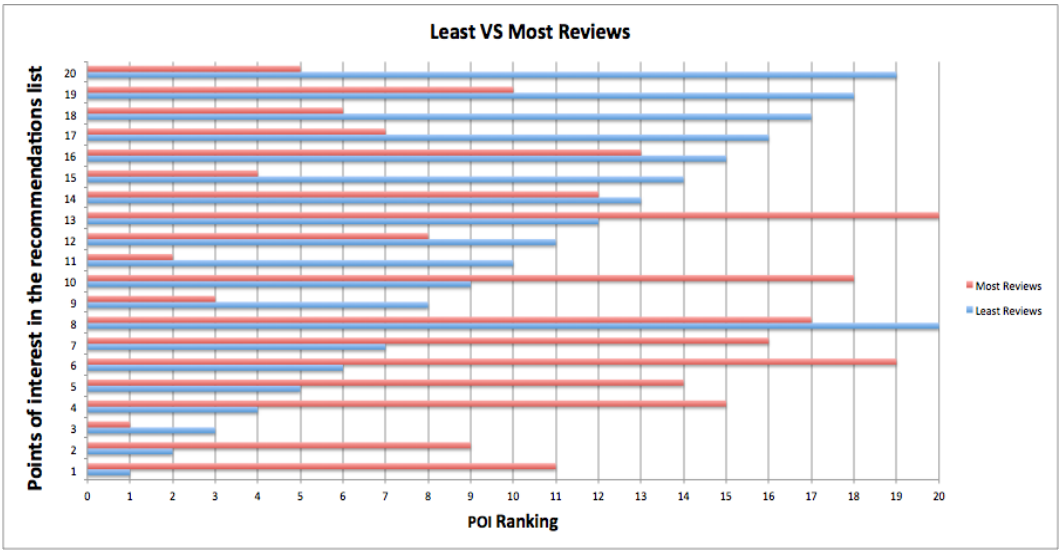

Figure 9: Comparison between users with least and most reviews

5.1.4 Users With Most Friends Traveling Together

Another scenario I explored is choosing the users with most friends traveling together. One interesting observation from this scenario, as seen in Figure 10, is that POI scores averaged in the [3.0, 4.0] range, indicating no strong dislike or like for any of them. Also, the recommendations for some POIs converged to the final score even for a small number of trip participants.

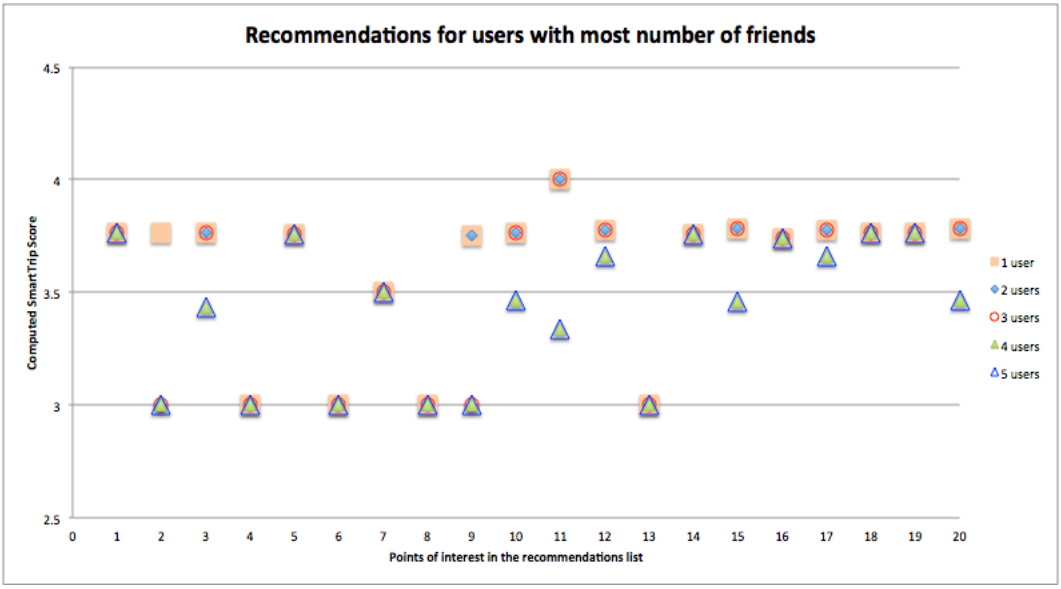

Figure 10: Recommendations made for groups of travelers with most friends

5.1.5 Users With Least Friends Traveling Together

In the case where users with least friends are traveling together, users also had a small number of reviews. Out of the five users with least friends, only one, User 2, had written any reviews. Therefore Figure 11 below only shows the individual recommendations for User 1 and User 2, and the aggregate results for both users. Since User 1 has no review and no friends, the POIs are going to be ranked by distance. However, since User 2 has past reviews, a model could be built that reflects her preferences, which in the end gives the final scores and ratings for the group.

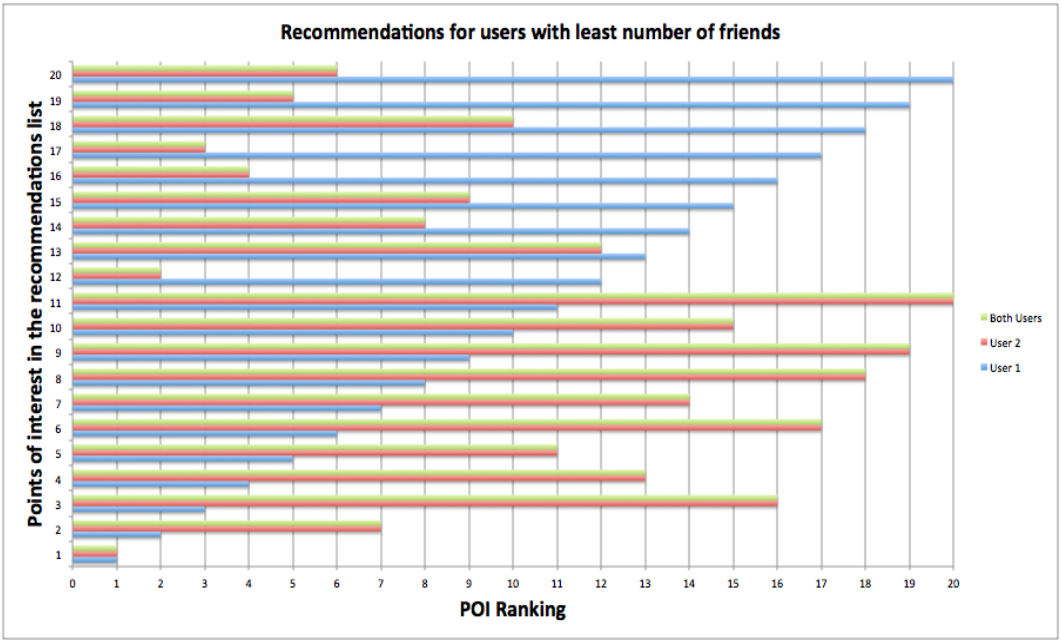

Figure 11: Recommendations made for groups of travelers with least friends

5.1.6 Majority voting versus distance to clusters

In order to determine which strategy to use for the collaborative filter, I conducted multiple experiments with both clustering and majority voting. As it can be seen from Figure 13 below, for the Las Vegas Hotel recommendations, clustering has produced higher scores, since each friend's score has equal weight when computing the score. In the case of majority voting, some of the POIs had visibly lower scores, due to a greater number of negative reviews. This implies that the clustering approach can amortize low scores and obfuscate the fact that some POIs are generally poorly rated, therefore I chose majority voting as the better approach for the collaborative filter.

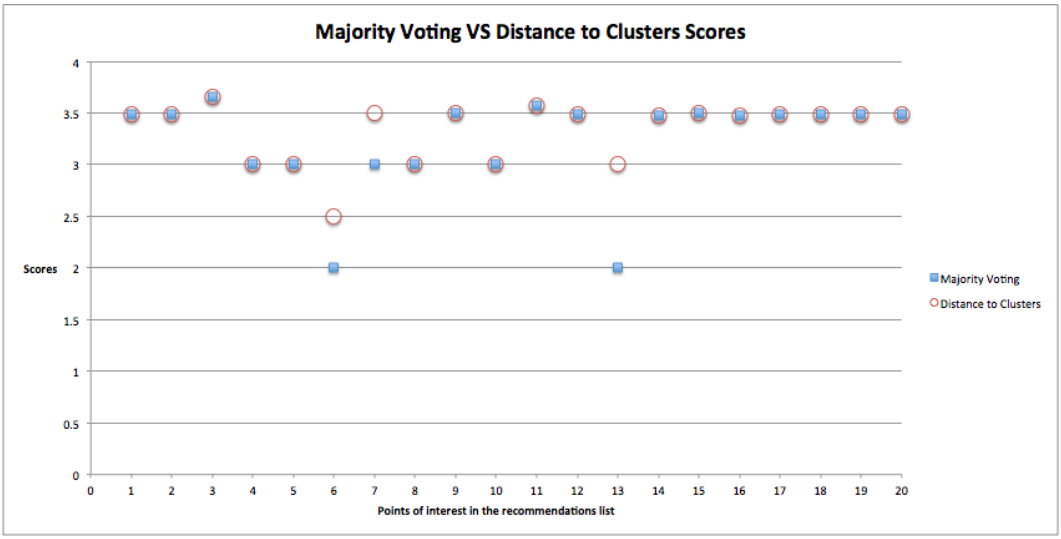

Figure 12: Majority voting VS distance to clusters comparison

5.2 Recommender Results Comparison

Among the experiments I performed to compare the recommender with other recommenders and itself, there were two particular scenarios that prove the VoyageWithUs recommender's two distinctive features:

- Making group recommendations, that accounts for each individual's personal preferences.
- Making personalized recommendations, by taking into account user history, preferences and their friends' influence.

5.2.1 Individual Versus Group Recommendations

I chose to compare the top five users that have generated the most number of reviews and run the recommender for each individually and for the group containing all five users. While the POI scores had little variation, the ranking of the POIs had more variation, which can be seen in *Table 1* below. Looking at the individual rank and then the group rank, one can see that the highest ranked POIs in the group ranking are, as expected, those that have the highest individual scores. Similarly, the lowest rated POIs are the ones that have low individual

scores. One can see that not all users in this scenario can absolutely agree on highest ranked POIs, with User 1 and User 3 ranking the overall favorite, Bellagio, in the bottom half of the twenty POIs in the location selected. However, the delta between those users' top and bottom rated POIs is negligible, all of their scores being in the [3.6, 3.9] range. If that were not the case, it would have been reflected in the final ranking. For instance, Flamingo had two scores: 2.5 and 2.0 which made it drop to the second to last position, despite getting similar scores as Bellagio for users 1 and 3.

Table 1: POI ranking for individuals and group of individuals

POI	User 1 rank	User 2 rank	User 3 rank	User 4 rank	User 5 rank	Group rank
Bellagio Hotel	16	2	15	1	1	**1**
Caesar's Palace	19	16	10	4	19	**16**
The Cosmopolitan	18	1	1	3	14	**2**
Flamingo	14	20	20	20	18	**19**
Paris Las Vegas	13	18	18	17	15	**15**

5.2.2 VoyageWithUs Recommender Compared to Yelp a and Random Recommender

The data that I used to implement the predictor was taken from the Yelp! public dataset. Therefore, I chose to generate recommendations for the same location on Yelp! and compare the rankings with the rankings produced by the VoyageWithUs recommender for the group with most reviews and with the ratings obtained via a random recommender, that scores the POI in a certain area randomly with scores between 1.0 and 5.0.

When generating recommendations in Yelp!, I did a query for “Hotels”, in a specific area in Las Vegas and, as expected, most of the returned results matched what the VoyageWithUs location-based filter produced. I performed the query from an account that has no friends and no reviews, from an account that has some friends and some reviews and while logged out. In all three cases, the results were identical, proving Yelp! does not personalize recommendation results. The default POI ranking in Yelp!, which is branded as **Best Match** seems to be a composite score involving the rating, number of reviews, but also other attributes which are not obvious without knowledge of their ranking algorithm.

As one can see from *Table 2* below, there were some POIs that had similar if not identical ranking in VoyageWithUs and Yelp! (The Cosmopolitan, Aria Sky Suites and Mirage), while two others had significantly different ranking. I believe that both the similar and different rankings are a consequence of the fact the VoyageWithUs makes personalized group recommendations, they just prove two different points. First, looking at the similar recommendations, one can see they have more than a few thousand reviews concentrated around a very similar high score. Since VoyageWithUs uses past recommendations and friends influence to make suggestions, it is only natural that for those highly rated POIs with thousands of reviews, my recommender will produce similar scores and ranking. Second, the dissimilar results prove that a popular POI on Yelp!, like Vdara, can drop significantly below the fold in preferences, depending on the trip participants individual preferences or influencers.

The reason for including the Random recommender is to show that both Yelp! And VoyageWithUs implement recommender algorithms that can generate more relevant results than just randomly scoring the POIs, although in this particular run of the Random recommender, the top results was coincidentally the same as Yelp!.

Table 2: POI Ranking Comparison

POI	VoyageWithUs ranking	Yelp ranking	Random recommender ranking
The Cosmopolitan	2	1	1
Bellagio Hotel	1	4	9
Aria Sky Suites	5	5	20
Mirage	11	9	17
Vdara	18	10	7

5.3 Recommender Performance

In order to measure recommender performance, I ran the recommender through various scenarios for ten to one hundred iterations, depending on the length of each iteration, measured and averaged the execution times. The plots of the scenarios are presented in the following sections.

5.3.1 Impact of number of user reviews on performance

Since the user with most reviews generated just over one thousand reviews, I chose to measure performance for users with 1, 10, 100, 500 and 1000 user reviews and 0 friends, in order to establish a baseline performance. As one can see in Figure 14, the recommender execution time averages to approximately one second for the minimum number of reviews and doubles for the maximum number of reviews, which is an acceptable increment if we wanted to compute the recommendations on demand from the mobile UI.

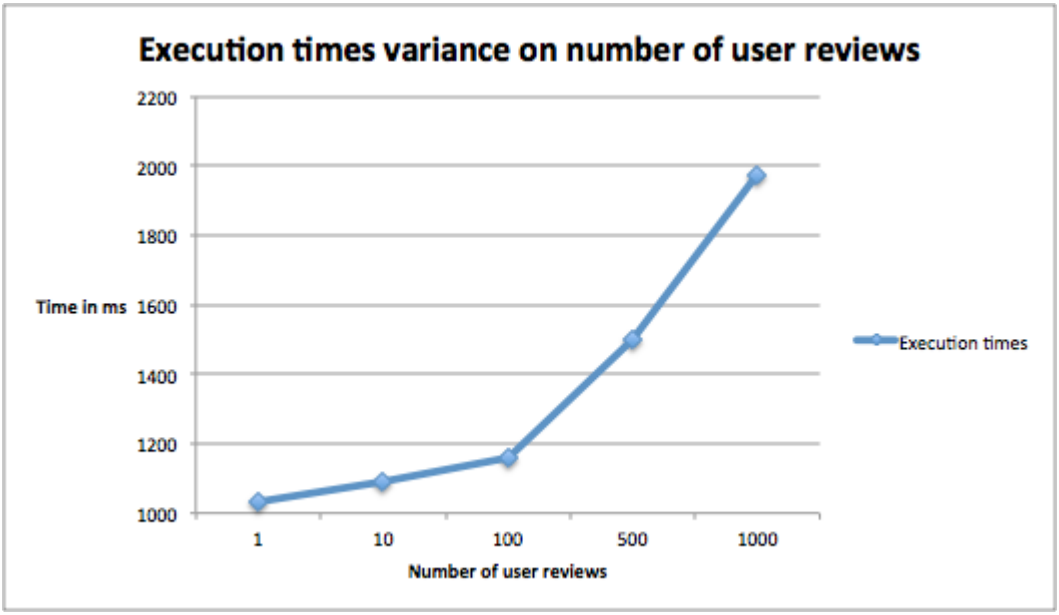

Figure 13: Execution times variance on number of user reviews

5.3.2 Impact of number of friends on performance

In order to determine the impact of the number of friends on performance, used in the collaborative-based filtering, I ran the recommender against users with 1, 10, 50, 100, 500 and 1000 friends, by taking the user with most friends and limiting the number of friends selected in the collaborative-based filter. Since users with most friends have many reviews and in turn, their friends have many reviews, I also made variations in the max number of reviews per user (1, 10, 100 and 1000). As it can be seen in Figure 15 below, number of reviews has a minimal impact compared to number of friends. Increasing the number of reviews can result in a 10% increase in execution time for the same number of user friends. The real performance killer is the increase in number of friends, which determines the execution time to increase from a few seconds when a user has 1 friend, to 30 minutes when the user has 1000 friends.

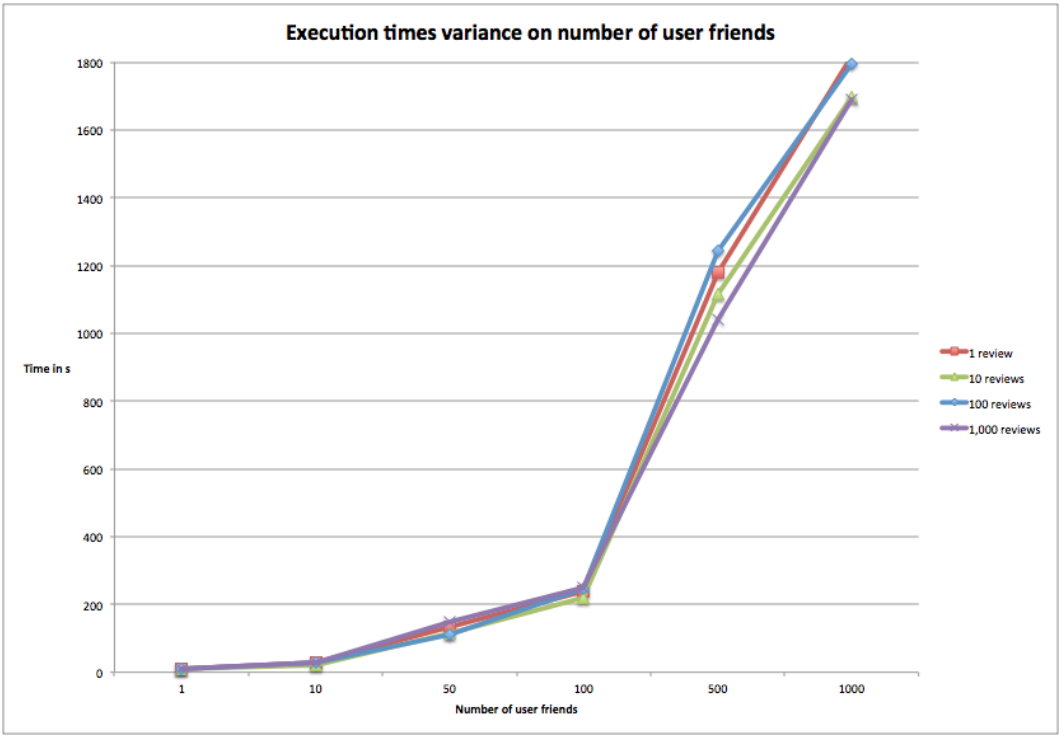

Figure 14: Execution times variance on number of user friends

5.3.3 Impact of number of trip participants on performance

The effect of the number of trip participants on the execution times is presented on Figure 16. The numbers were obtained for the top ten users with most reviews, but I limited the number of friends to one, in order to reduce the execution times. Each travel group was obtained by adding one traveler to the previous travel group, starting with the user with most reviews and ending with a group containing all ten users. The number of reviews was limited to 500, because that was the minimum number of reviews among the ten users. It can be seen that increasing the number of trip participants also increases the execution time, but even with ten users, the maximum time is shy of 45 seconds, so the performance bottleneck is still caused by the performance of the collaborative-based filter when the users have many friends.

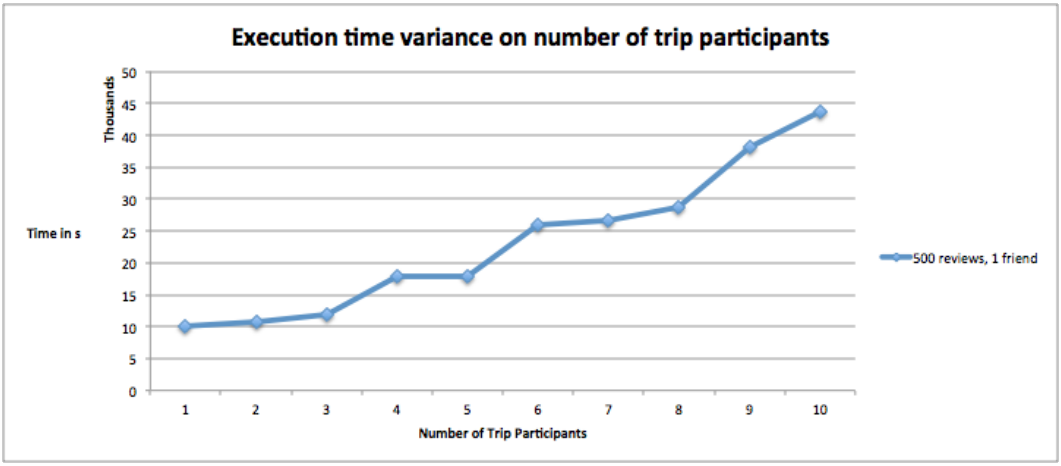

Figure 15: Execution times variance on number of trip participants

5.3.4 Strategies for performance improvement

One strategy to improve performance is pre-computing the collaborative-based filtering asynchronously and storing it server-side instead of computing it on the fly. The main trade off is storage and creating an index that would allow to quickly retrieve the pre-computed results. Another trade off is that the predictions will be made on stale data, so a proper schedule will have to be established to improve the freshness of the data.

Chapter 6: Future Work

In the scope of this report I implemented a recommender architecture and implemented basic integration with a proof of concept application that targets enhancing the group travel planning experience. I believe the system designed has the potential to be extended and productized, if the following aspects are addressed in the future:

- Improving the collaborative-based filter performance by looking into periodically asynchronously precomputing the recommendations generated by this filter.
- Integrating multiple LBSN streams which will provide user sentiments and edges in the friends network, instead of using a static dataset. This integration will require looking into technologies like Apache Spark to aggregate and process social-media streams to perform sentiment analysis.
- Properly integrating all the software components by moving the infrastructure to a paid tier in AWS, which will allow implementing resilience and failover mechanisms via autoscaling.
- Improving the mobile application UX and testing it on focus groups to ensure that the different flows in the application are intuitive.
- Addressing privacy concerns related to sharing of different artifacts such as user location and context, trip itineraries and travel artifacts.

Chapter 7: Conclusions

The gaps in current travel planning systems have created an opportunity to research and develop approaches that better serve groups of travelers, may they be families or backpackers. VoyageWithUs is a mobile and REST API solution that enhances group travel and tries to close some of the shortcomings of group travel planning, The solution proposed by this paper caters to groups of travelers by making personalized group travel recommendations and by allowing groups of travelers to collaborate in the travel planning process. The first feature, implemented via a hybrid recommender system, sets VoyageWithUs apart from other recommender applications by aggregating individual recommendations into a set of recommendations that are suitable for all trip participants. Experimental results have proven the recommender can incrementally adapt as more trip participants are added and the recommendations are generated such that they are suitable to every trip participant. The second distinctive feature is the collaborative aspect of VoyageWithUs, which allows users to share trips and trip artifacts, invite others to trips and allow everyone to contribute to a trip. With proper social streams integration, such as the ability to retrieve user sentiment and create edges between users directly from Facebook and Instagram, VoyageWithUs could be productized into a novel travel planning platform that is targeted at people that travel in groups, whether they are physically co-located or not.

Appendix A: The Yelp Challenge Dataset Format

business

```
{
    'type': 'business',
    'business_id': (encrypted business id),
    'name': (business name),
    'neighborhoods': [(hood names)],
    'full_address': (localized address),
    'city': (city),
    'state': (state),
    'latitude': latitude,
    'longitude': longitude,
    'stars': (star rating, rounded to half-stars),
    'review_count': review count,
    'categories': [(localized category names)]
    'open': True / False (corresponds to closed, not business hours),
    'hours': {
        (day_of_week): {
            'open': (HH:MM),
            'close': (HH:MM)
        }, ...
    },
    'attributes': {
        (attribute_name): (attribute_value), ...
    },
}
```

review

```
{
    'type': 'review',
    'business_id': (encrypted business id),
    'user_id': (encrypted user id),
    'stars': (star rating, rounded to half-stars),
    'text': (review text),
    'date': (date, formatted like '2012-03-14'),
    'votes': {(vote type): (count)},
}
```

user

```
{
    'type': 'user',
    'user_id': (encrypted user id),
    'name': (first name),
    'review_count': (review count),
    'average_stars': (floating point average, like 4.31),
    'votes': {(vote type): (count)},
    'friends': [(friend user_ids)],
    'elite': [(years_elite)],
    'yelping_since': (date, formatted like '2012-03'),
    'compliments': {
        (compliment_type): (num_compliments_of_this_type), ...
    },
```

```
    'fans': (num_fans),
}
```

check-in

```
{
    'type': 'checkin',
    'business_id': (encrypted business id),
    'checkin_info': {
        '0-0': (number of checkins from 00:00 to 01:00 on all Sundays),
        '1-0': (number of checkins from 01:00 to 02:00 on all Sundays),
        ...
        '14-4': (number of checkins from 14:00 to 15:00 on all Thursdays),
        ...
        '23-6': (number of checkins from 23:00 to 00:00 on all Saturdays)
    }, # if there was no checkin for a hour-day block it will not be in the dict
}
```

tip

```
{
    'type': 'tip',
    'text': (tip text),
    'business_id': (encrypted business id),
    'user_id': (encrypted user id),
    'date': (date, formatted like '2012-03-14'),
    'likes': (count),
}
```

photos (from the photos auxiliary file)

This file is formatted as a JSON list of objects.

```
[
    {
        "photo_id": (encrypted photo id),
        "business_id" : (encrypted business id),
        "caption" : (the photo caption, if any),
        "label" : (the category the photo belongs to, if any)
    },
    {...}
]
```

Appendix B: Application Screenshots

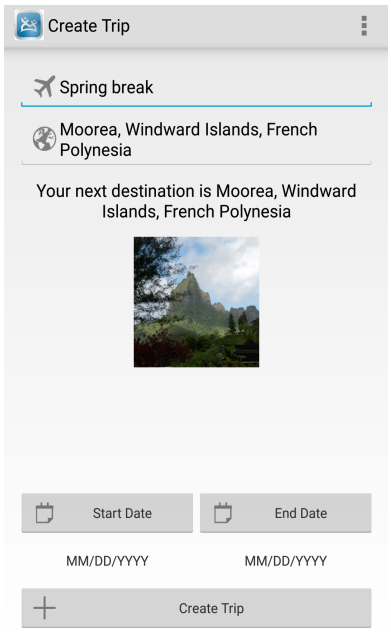

Figure B.1: Create trip Activity

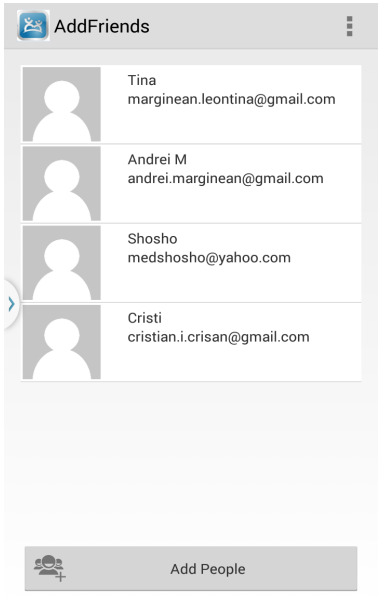

Figure B.2: Adding Friends to a Trip

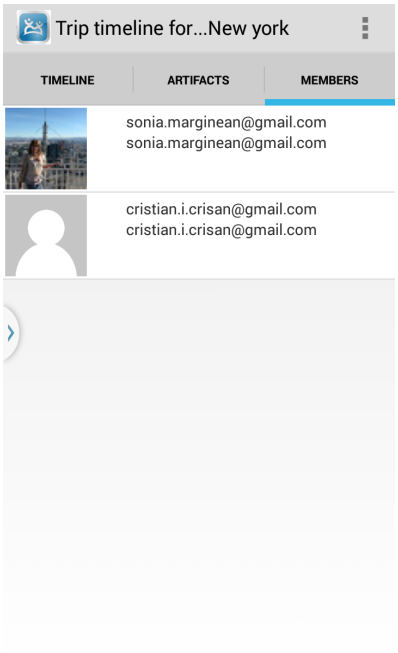

Figure B.3: Viewing Trip Members

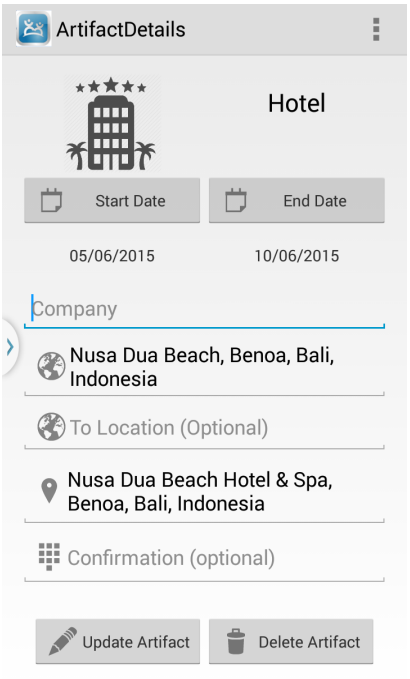

Figure B.4: Adding Trip Artifacts

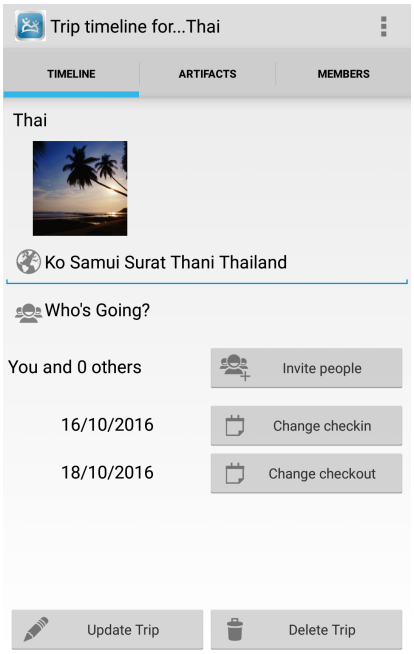

Figure B.5: Trip Summary

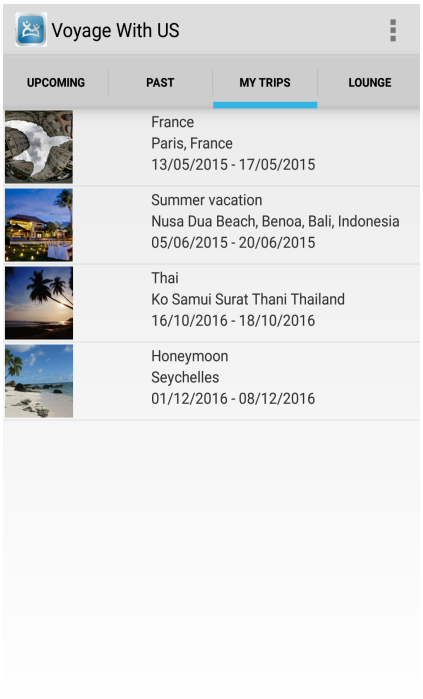

Figure B.6: List of User Trips

Appendix C: REST API Endpoints Description

Table 1: Trip CRUD Operations

Endpoint	**Method**	**Description**
<api_url>/trips	POST	Creates a new trip object, needs a trip object to be passed in the request body.
<api_url>/trips/{id}	DELETE	Deletes a trip. Needs ID of trip to delete passed in as a path parameter.
<api_url>/trips/{id}	PUT	Updates a trip. Needs ID of the trip as a path parameter and updated trip object as body parameter.
<api_url>/trips/{id}	GET	Gets detailed trip object. Needs ID of the trip to view as path parameter.
<api_url>/trips	GET	Returns all trip IDs.
<api_url>/trips/users/{id}	GET	Returns all the trip IDs for user with specified ID.
<api_url>/trips/users/{id}/current	GET	Returns all the current and future trip IDs for user with specified ID.
<api_url>/trips/users/{id}/past	GET	Returns all the past trip IDs for user with specified ID.

Table 2: Trip Artifact CRUD Operations

Endpoint	Method	Description
<api_url>/trips/{id}/ artifacts	POST	Creates a trip artifact for a trip. I pass in an Artifact object to create as a body parameter and trip ID as a path parameter.
<api_url>/artifacts/{id}	DELETE	Deletes a trip artifact with the specified ID.
<api_url>/artifacts/{id}	PUT	Updates the trip artifact that had the ID specified in the path parameter with a trip artifact object specified in the request body.
<api_url>/artifacts/{id}	GET	Gets the trip artifact with the specified ID.
<api_url>/trips/{id}/ artifacts	GET	Gets all artifacts for a trip with the specified ID.

Table 3: User CRUD Operations

Endpoint	Method	Description
<api_url>/users	POST	Creates a user profile object, given a user object in the request body.
<api_url>/users/{id}	GET	Gets a user's profile info with the specified ID.
<api_url>/users/{id}	PUT	Updates the user who has the ID specified in the path param with the user object specified in the body of the request.
<api_url>/users/{id}	DELETE	Deletes the user with the ID specified.

Table 4: Sentiment CRUD Operations

Endpoint	**Method**	**Description**
<api_url>/sentiments	POST	Creates a sentiment object passed in the request body.
<api_url>/sentiments	GET	Gets all sentiment IDs from the datastore.
<api_url>/sentiments /pois/{id}	GET	Gets sentiment IDs for a POI with the specified ID in the path parameter.
<api_url>/sentiments /users/{id}	GET	Gets sentiment IDs for a user with the specified ID in the path parameter.

Table 5: POI CRUD Operations

Endpoint	**Method**	**Description**
<api_url>/pois	POST	Creates a POI object given in the request body.
<api_url>/pois	GET	Gets all POI IDs.
<api_url>/pois/{id}	GET	Gets specific details for a POI with the specified ID.
<api_url>/pois/{id}	DELETE	Deletes the POI with the ID specified in the path parameter.
<api_url>/pois/{id}	PUT	Updates the POI that has the ID specified in the path parameter with the POI object in the request body.

Table 6: Recommender Operations

Endpoint	**Method**	**Description**
<api_url>/recommend/trip/{id}	GET	Gets all the recommendations for a trip with an ID given as path parameter.
<api_url>/recommend/trip/{id}?type=<poi_type>	GET	Gets the recommendations for certain POI types (hotel, restaurant, etc...) for a trip with an ID given as path parameter.
<api_url>/recommend/nearby?location=<location>	GET	Gets nearby recommendations based on the passed in location.

Table 7: Recommender Endpoints Filters

Filter	**Query Parameter**
Location	nearby=<location>
Day of Week	weekday=<day_of_week>
Month	month=<month>
Day	day=<day>
Time of day	time=<hh:mm>
POI type	type=<poi_type>
User	user=<user_id>
Price category	price=<$$$signs>

Table 8: Trip Itinerary Operations

Endpoint	**Method**	**Description**
<api_url>/itinerary/trip/{id}	GET	Triggers the computation of an itinerary for trip with the specified ID.
<api_url>/itinerary/walking/trip/{id}	GET	Triggers the computation of a walking tour for trip with the specified ID.

Bibliography

[1]. Kosmides, P., Remoundou, C., Demestichas, K., Loumiotis, I., Adamopoulou, E., & Theologou, M. (2014). A Location Recommender System for Location-Based Social Networks. *2014 International Conference on Mathematics and Computers in Sciences and in Industry*. doi:10.1109/mcsi.2014.39

[2]. Chen, S., Owusu, S., & Zhou, L. (2013). Social Network Based Recommendation Systems: A Short Survey. *2013 International Conference on Social Computing*. doi:10.1109/socialcom.2013.134

[3]. Esfahani, M. H., & Alhan, F. K. (2013). New hybrid recommendation system based On C-Means clustering method. *The 5th Conference on Information and Knowledge Technology*. doi:10.1109/ikt.2013.6620054

[4]. Toozandehjani, H., Zare-Mirakabad, M.-R., & Derhami, V. (2014). Improvement of recommendation systems based on cellular learning automata. *2014 4th International Conference on Computer and Knowledge Engineering (ICCKE)*. doi:10.1109/iccke.2014.6993443

[5]. Xie, M., Lakshmanan, L. V., & Wood, P. T. (2011). CompRec-Trip: A composite recommendation system for travel planning. *2011 IEEE 27th International Conference on Data Engineering*. doi:10.1109/icde.2011.5767954

[6]. Meehan, K., Lunney, T., Curran, K., & Mccaughey, A. (2013). Context-aware intelligent recommendation system for tourism. *2013 IEEE International Conference on Pervasive Computing and Communications Workshops (PERCOM Workshops)*. doi:10.1109/percomw.2013.6529508

[7]. Rachuri, K. K., Hossmann, T., Mascolo, C., & Holden, S. (2015). Beyond location check-ins: Exploring physical and soft sensing to augment social check-in apps. *2015 IEEE International Conference on Pervasive Computing and Communications (PerCom)*. doi:10.1109/percom.2015.7146518

[8]. Veningston, K., & Shanmugalakshmi, R. (2015). Personalized location aware recommendation system. *2015 International Conference on Advanced Computing and Communication Systems*. doi:10.1109/icaccs.2015.7324140

[9]. Yu, Z., Xu, H., Yang, Z., & Guo, B. (2016). Personalized Travel Package With Multi-Point-of-Interest Recommendation Based on Crowdsourced User Footprints. *IEEE Transactions on Human-Machine Systems IEEE Trans. Human-Mach. Syst.*, *46*(1), 151–158. doi:10.1109/thms.2015.2446953

[10]. Beirigo, B. A., & Santos, A. G. D. (2015). A parallel heuristic for the travel planning problem. *2015 15th International Conference on Intelligent Systems Design and Applications (ISDA)*. doi:10.1109/isda.2015.7489239