DDest

Travel Destination Recommender System for iOS

By,
Caio Novelini, 100985497
Supervised by Prof. Louis D. Nel
School of Computer Science

Honours Project (COMP 4905) - Carleton University
November 28th, 2017
Abstract

Traveling is part of most people vacations plans and sometimes finding a perfect destination can be a hard task. This paper discusses the implementation of an iOS mobile application that can be used to help users find a good matching destination by simply answering a few questions. Using technologies like NodeJS and Swift to create a recommender system, the application is able to find a suitable destination for the user as well as a handful of extra information about the place with text, images and links.

This report will explain the implementation of this application as well as the analysis of the results when used by real travelers.
Acknowledgments

I would like to thank Prof. Louis D.Nel for all the support and valuable advices throughout the development of this project. I would also like to thank my fiancé Mariah Tremblay who has given me support and helped me to build this database of destinations. Finally, I would like to thank everyone who used the application and helped me to make a result analysis by giving the proper feedback.
# TABLE OF CONTENT

1. INTRODUCTION 6
   1.1 CONTEXT 6
   1.2. GOALS 6
   1.3. OBJECTIVES 7
   1.4. OUTLINE 8

2. BACKGROUND INFORMATION 8
   2.1. RECOMMENDER SYSTEM 8
   2.2. COSINE SIMILARITY 9
   2.3. REST WEB SERVICE 10

3. APPLICATION OVERVIEW 11

4. IMPLEMENTATION 15
   4.1. SYSTEM DESIGN 15
   4.2. SYSTEM SETUP 16
   4.3. DATABASE DESIGN 17
   4.4. DATA DESIGN 19
   4.5. ALGORITHM 20
   4.6. REST API 23
   4.7. EXTERNAL API’S 25
   4.8. IOS IMPLEMENTATION 26
   4.9 DEPLOYING THE APPLICATION USING HEROKU 28

5. EVALUATION 30
   5.1 ACCURACY STUDY 30

6 CONCLUSION 31
   6.1 SUMMARY 31
   6.2 FUTURE WORK 32

7. REFERENCES 33
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Figure 1</td>
<td>Cosine Similarity Formula</td>
<td>10</td>
</tr>
<tr>
<td>Figure 2</td>
<td>REST Web Service Architecture</td>
<td>11</td>
</tr>
<tr>
<td>Figure 3</td>
<td>First question presented to the user</td>
<td>12</td>
</tr>
<tr>
<td>Figure 4</td>
<td>Questions presented to the user</td>
<td>13</td>
</tr>
<tr>
<td>Figure 5</td>
<td>Recommended Destination View</td>
<td>14</td>
</tr>
<tr>
<td>Figure 6</td>
<td>System Design</td>
<td>15</td>
</tr>
<tr>
<td>Figure 7</td>
<td>Database ER Diagram</td>
<td>18</td>
</tr>
<tr>
<td>Figure 8</td>
<td>Example of a JSON response from the server</td>
<td>19</td>
</tr>
<tr>
<td>Figure 9</td>
<td>Swift classes</td>
<td>20</td>
</tr>
<tr>
<td>Figure 10</td>
<td>Similarity Classification</td>
<td>23</td>
</tr>
<tr>
<td>Figure 11</td>
<td>Example of hotel information provided by Google’s API.</td>
<td>25</td>
</tr>
<tr>
<td>Figure 12</td>
<td>Swift code that ask permission to use location</td>
<td>26</td>
</tr>
<tr>
<td>Figure 13</td>
<td>&quot;Stepper button&quot; example</td>
<td>27</td>
</tr>
<tr>
<td>Figure 14</td>
<td>UIKit button example</td>
<td>27</td>
</tr>
<tr>
<td>Figure 15</td>
<td>POST request using Alamofire</td>
<td>28</td>
</tr>
<tr>
<td>Figure 16</td>
<td>Heroku setup</td>
<td>29</td>
</tr>
</tbody>
</table>
1. Introduction

1.1 Context

Traditionally, when the period of vacations approaches, people start looking for a destination to travel. This process can be long and frustrating as you start looking at thousands of options in your favorite travel website or getting overwhelmed by the huge amount of information given to you when simply typing “Where to go summer vacations” at Google.

Nowadays a vast amount of information is available to everyone in the simple click of a mouse, and being able to filter this information to capture only what interest you is a key to both save time and be more assertive.

With this in mind, the idea of a recommender system, in this case, is to pick the right destination specifically for you as if the system somehow knew you and your preferences at that particular moment, excluding all the non-interesting information that would make the user lose time. DDest not only finds the user a suitable destination, it also provides a vast amount of information about it, such as description, images, activities, places to stay, ratings and Instagram images.

In a more fast and modern world portability and easy access is key to productivity, considering this fact, the main client-side of the application is a mobile application in which the user experience was carefully crafted to provide the best experience.

1.2. Goals

The goal of this project is to provide a mobile application in which the user can give an input by answering some questions and the system will recommend the best fitting destination. Once the destination is given the application should provide a wide range of information about the destination, such as images, descriptions, activities, places to stay and ratings. The ultimate goal of this project is to create an application that will not only recommend with a high degree of accuracy a destination, but will also to gather and display the information that user may need when going to that destination.
1.3. Objectives

The approach used to achieve the goals is to create a NodeJS server that will handle all the intelligence of the application. That is: working as a recommender system, providing an API so that the client-side of the project can reach out that server with the user information, and get back the formatted result. These are the functional requirements of the project:

- The mobile application should get the current location of the user.
- The mobile application should automatically get the UDID of the device and send it to the server.
- The mobile application should display and get the answer of 5 questions to the user.
- The mobile application should reach out to the server and request for a suitable destination for the user.

- The mobile application should display:
  - Image of the destination.
  - Distance of the destination for the user in Km.
  - Description of the information.
  - Top 3 activities in the destination.
  - Top 3 places to stay.
  - An external link providing more information about the place.
  - An external link to book accommodation in that place.
  - Instagram Images provided by a partner.

- The NodeJS server should work as a recommender system.
- The NodeJS server should work as REST Web Service
- The NodeJS server should have an API.
- The NodeJS server should be able to connect with another API’s, such as Google and Instagram.
- The NodeJS server should communicate with the MySQL database

When all objectives are met, the user should be able to download the application answer five questions and receive the recommended destination with all the information.
1.4. Outline

The rest of this report is structured as follows:

Section 2 – Present the background knowledge needed for the project.

Section 3 – Present an overview of the application.

Section 4 – Describe in details the implementation of the project.

Section 5 – Evaluation of the result.

Section 6 – Conclusions.

2. Background Information

2.1. Recommender System

A recommender system, as the name itself reveals, is a system that produces a list of recommendations especially crafted for the user. There are two basic architectures for a recommender system:

1. A Content-Based Architecture System makes use of discrete characteristics of items to recommend another item with similar properties.
2. A Collaborative-Filtering Architecture System makes use of the relationship between users and items. Based on past rates, the system can determine if users are similar and can provide a recommendation based on the ratings of these similar users.

This report will focus on the Content-Based recommender system because it is the recommender system architecture chosen to execute this project.

As previously mentioned the Content-Based architecture is based on comparing properties to determine if one item is similar to the other. For example, consider the properties of a song:

- The genre of the song.
- The artist of the song.
- The year of the song.
These are all properties that can be previously known for a song. Now let’s say the user listened to ten songs using this particular system, and all of them from one specific singer/band, when recommending the next song for your user the system will have more chances to recommend a song that is performed by this specific singer/band.

But how can we compare and check if an item is suitable for a user? First, we need to represent both, the item and the user in a mathematical way using a vector or a matrix. In this simple example, I will use a user that is looking for a destination to travel and the possible matching destination. To represent the user, I will create a 3-dimensional vector that represents some characteristics of the user: [Budget, Adventurous, PossibleByCar].

- Budget: A rating from 0-5 on 0 being very tight and 5 being very loose.
- Adventurous: A rating from how the user like adventure on 0 being do not like at all and 5 love it.
- PossibleByCar: A Boolean value indicating if it the user is more likely to travel to destinations where is possible to go by car.

Now let’s assume user A = [3, 5, 1] and we have the destinations D1 = [1,1,0] and D2 = [4,5,1] how can the system decide which destination to recommend? To achieve this, we can use the distance between the vectors by calculating the cosine similarity. This will be better discussed in the following section.

2.2. Cosine Similarity

Cosine similarity is a measure that is used to predict how similar two non-zero vectors are. The use of it is particularly interesting because it can be applied to vectors of any dimensions and its evaluation is very efficient.

Given two non-zero vectors, A and B we can measure the cosine similarity by applying the following formula:
The result of this calculation will range between -1 and 1 or 0 and 1 if none of the vectors contains a negative number. In this case 0 means that the vectors are the complete opposite and 1 means that they are exactly the same. As an example, let’s consider the vector A = [0.2, 0.1, 0.3, 0.1, 0.2] and a vector B = [0.8, 0.7, 0.8, 0.5, 0.9] and compare both of them with vector D = [0.7, 0.9, 0.7, 0.6, 0.9] to know if A or B is more similar to D.

With the vectors in hand we can apply the formula above to determine the similarity:

\[
\text{Similarity A-D} = \frac{A \cdot D}{\|A\|_2 \|D\|_2} = 0.906747
\]

\[
\text{Similarity B-D} = \frac{B \cdot D}{\|B\|_2 \|D\|_2} = 0.988159
\]

By analysing the result, we can determine that the vector B is more similar to the vector D when compared with vector A.

### 2.3. REST Web Service

REST stands for Representational State Transfer and it is an architecture that is designed for a better web performance of the service. It typically uses HTTP and HTTPS to exchange resources between clients and servers using a standardized interface and protocol.

Endpoints are URIs used to access specific information’s in the server. These accesses are made through a uniform interface designed to create, read, update and delete (CRUDE). GET is used to retrieve some information in the server. POST is used to transfer a new state into the server. PUT create a new resource in the server and DELETE can be used to delete a resource.

The REST Web Service architecture can be represented in the following structure:
3. Application Overview

As previously mentioned the goal of the application is to recommend a destination to the user after answering five questions related to his travel habits and tastes. In this section, we will walk through the idea so it can lead to a better understanding of the global process.

As soon as the application is opened the user is presented with the first question, no login process is needed because the application is intended to work fluidly and fast.
The first question is the only one where the user selects the intensity by clicking in the “+” sign and in the “−” sign to increase or decrease the intensity of the particular property. Once the levels are set, the user presses on the “next” button and go to the next question.

The questions were elaborated in a way to extract what is the profile of the traveler which, in this case, is the user. Four main components of the user’s profile were analyzed in order to obtain an accurate recommendation in the end of the process. That is: the user’s interest in the five main activities (Adventure, Romantic Relaxing, Exploring, Sports), The user’s disposal of resources to travel, how far the user is willing to go and what is the time availability of the user during the trip. Considering these components five questions were elaborated to present to the user as can be seen in the next image.
From the second to the fifth question the user must pick one of the options as the answer. Once all the answers are provided the application will return the recommended destination.
Welcome to Toronto, the most multicultural ³ diverse city on the planet; over 140 languages are spoken. It’s estimated that over half of Toronto’s residents were born outside Canada, and despite its complex makeup, Torontonians generally get along.

Top Things to Do: ⁴

<table>
<thead>
<tr>
<th>Destination</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ripley’s Aquarium Of Canada</td>
<td>4.1</td>
</tr>
<tr>
<td>Toronto Islands</td>
<td>4.2</td>
</tr>
<tr>
<td>Royal Ontario Museum</td>
<td>4.3</td>
</tr>
</tbody>
</table>

Top places to stay: ⁵

<table>
<thead>
<tr>
<th>Place</th>
<th>Rating</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lily Home Hostel</td>
<td>5.1</td>
</tr>
<tr>
<td>EGI Suites - York St Condos</td>
<td>5.3</td>
</tr>
</tbody>
</table>

Figure 5 - Recommended Destination View

1. Name of the recommended destination.
2. Current distance from the destination.
3. Description of the destination.
4. List of activities of the destination.
4.1 Name of the activity.
4.2 Type of the activity.
4.3 Rating of the activity.
5. List of places to stay in the destination.
5.1 Name of the place.
5.2 Type of the place.
5.3 Rating of the place.
6. Link to a website giving more information about the destination.
7. Link to a website where you can book accommodations.
8. Images of the place from a specific Instagram account.
9. Recommend another destination based on the same settings.
The user is able to interact with the destination by accessing the website of the recommended accommodations, clicking on the partner’s images to be redirected to their Instagram account or pressing the “Not quite what I want. Give me another destination” button to be presented with another matching destination. In the next section, we will walk through the details of the implementation.

4. Implementation

4.1. System Design

The DDest system is composed of a client-server architecture organized in the following structure:

![Figure 6 - System Design](image-url)
NodeJS Server

The server works as the brain of the system. It communicates with the clients through a REST service and is able to receive the client input, process the data and to return the desired output. When processing the user data, the NodeJS server can communicate with the system database, fetch and update data when needed as well as reach out to external web services to add extra information to the system.

MySQL Database

The database is responsible for holding permanent information about destinations and users. It can only be accessed and edited by the server. More details about the database design will be discussed in the section 4.2.

iOS Client

The iOS client is the application living inside of the user’s phone, more specifically in this case iPhone. Through the iOS application the user will insert the input by answering five questions and receive all the output from the system. The application is also responsible for reading important data from the user such as up-to-date latitude and longitude coordinates of the device.

External API’s

External API’s are services provided by other companies that can be accessed by a REST web service. In this project the external API’s used are Google API and Instagram API. More details on the external API’s will be provided in section 4.6.

4.2. System Setup

In order to make the entire system to work a setup process was needed. On the server side, we had to install NodeJS v6.10.0 to be able to run the back-end in the machine. Dependencies were used to simplify some tasks in the server, we managed them using npm, a JavaScript package manager with the following configuration:

```json
{
   "name": "ddest",
   "version": "1.0.0",
   "description": "",
   "main": "server.js",
   "scripts": {
```
For the database setup, we used MySQL Workbench to execute the design that will be discussed in the next section. To setup the iOS client side we used XCode 9.0 and Swift 4.0.

4.3. Database Design

The database is very simple and consists in 4 tables:

- Destinations: Hold all the destination information such as description, images, locations and score for all the categories.
- Hotels: Hold all the hotels of all destinations.
- Activities: Hold all the activities from all destinations
- UsersSearch: Hold the history of inputs from every user.

The system does not hold any personal information about the user. The id of the user is the UDID of its device.
The database was populated by an extensive research hand picking 40 destinations in Canada along with many accommodations and activities options. Some information’s such as ratings are fetch in real time using the Google’s Place API, where by querying the service using the place name, the information needed returns.
4.4. Data Design

The system uses JSON format to transmit data from the server to the client and from the external API’s to the server. Inside of the server, using JavaScript, the data is also handled using the JSON object format making it easier to create an API that will return JSON as data type.

Once in the iOS client, the JSON response data is stored, organized and handled in instances of swift classes.
4.5. Algorithm

The algorithm was designed to promote the interaction between the user’s input and the destinations in the database. Implementing the recommender system, the algorithm creates the structure by transforming input in vectors and comparing them when needed.

In a higher-level, the algorithm looks like this:

\[
\text{Recommend Destination}\left(\text{input}\right):
\]

\textbf{Input:} User’s 5 answers

\textbf{Output:} Recommended destination

\[
\text{userVector} = \text{inputToVector}\left(\text{input}\right)
\]

\[
\text{destinations} = \text{Fetch all destinations from the database}
\]

\textbf{for} destination in destinations \textbf{do}:

\[
\text{getDistanceFromUser}\left(\text{destination}\right)
\]

\[
\text{if distance } < \text{userInput maximum accepted distance}:
\]

\[
\text{destinationVector} = \text{destinationToVector}\left(\text{destination}\right)
\]

\[
\text{calculateSimilarityWithUser}\left(\text{destinationVector}, \text{userVector}\right)
\]
Sort destinations by similarity.

Pick the 3 most similar destinations.

Shuffle in a random order.

Return the 3 destinations.

With a better idea on how the algorithm works we can now discuss the functions within the algorithm.

**inputToVector (input)**

This function receives the raw user’s answers and transforms into the standard vector that will later be compared with the destination.

The user vector $V = \{\text{adventure, romantic, relaxing, sportive, budget, time}\}$ have the following parameters:

**Adventure**: User’s input on question 1.adventures ranging from 0 – 5.

**Romantic**: User’s input on question 1.romantic ranging from 0 – 5.

**Relaxing**: User’s input on question 1.relaxing ranging from 0 – 5.

**Exploring**: User’s input on question 1.exploring ranging from 0 – 5.

**Sportive**: User’s input on question 1.sportive ranging from 0 – 5.

**Budget**: User’s input on question 3 ranging from 0 – 4.

**Time**: User’s input on question 4 ranging from 0 – 4.

This function uses question number 5 “What do you like the most” answers as an enhancer of one or more parameters following these rules:

- If answer is “Food/Wine”: Exploring and Romantic are incremented by 1.
- If answer is “Wildlife/Nature”: Adventure is incremented by 1.
- If answer is “Hiking”: Sportive and Adventure are incremented by 1.
- If answer is “Surfing”: Sportive is incremented by 1.
- If answer is “Cultural Exploration”: Exploring is incremented by 1.
**getDistanceFromUser** *(destinations)*:

This function makes use of the Google API to calculate the distance between 2 pairs of coordinates. The user’s location is provided by the user’s device and the destination coordinates are provided by the application database.

**destinationToVector** *(destination)*:

This function transforms the destination information coming from the application database to the standard vector \( V = [\text{adventure, romantic, relaxing, sportive, budget, time}] \). In this case, we have 14 destination parameters that combined will result in a normalized vector \( V \).

**Determining a parameter on the destination vector**:

To determine the parameter on the destination vector, the application gives weights for the relevant fields in the database when related to that parameter. As an example, the destination fields relevant to adventure are [Adventurous, Sportive, Surfing, Trail].

The weights were assigned considering the level of relevancy of the parameter with the field. Adventurous (10), Sportive (5), Surfing (8), Trail (10). After applying the weights, the parameter is normalized with the maximum size of the same parameter in the user’s vector (5) so it can be properly compared later. As an example, the parameter adventure of the destination vector is represented by \( \text{Adventure} = (((\text{adventurous} \times 10 + \text{sportive} \times 5 + \text{surfing} \times 8 + \text{trail} \times 10) \times 5)/93) \). The same concept is applied to all the other five parameters so that the destination vector is complete.

**calculateSimilarityWithUser** *(destination)*:

In possession of both vectors, User’s and Destination, it is possible to calculate the cosine similarity between them by applying the process discussed in the section 2.2. In a log of the server we can see the calculations:
4.6. REST API

The system provides a public API so that another application can use the DDest database and services. The access to the API is limited and is given by private access-tokens that are generated by the system and handled to the authorized user.
The endpoints of the API are:

**GET**

/destinations

Input parameters: none.

Output parameters: List of all the destinations in the database.

Data format: JSON.

/destination/:id

Input parameters: id of the destination.

Output parameters: Information about the destination.

Data format: JSON.

/destination/name/:name

Input parameters: destination name

Output parameters: Information about the destination.

Data format: JSON

/recommendDestination

Input parameters: adventure – INT from 0-5,
romantic – INT from 0-5,
relaxing – INT from 0-5,
exploring – INT from 0-5,
sportive – INT from 0-4,
budget – INT from 0-4,
time – INT from 0-4,
distance – INT from 0-4,
lat – Float,
long – Float,
likeMost – INT from 0-5

Output parameters: Top 3 destinations

Data format: JSON

4.7. External API’s

The system makes use of two external API’s to fetch important data from other servers. The Google API provides the distances between the user and the destinations and the extra information about accommodations, such as up to date ratings, website and more, as can be seen in the image below.

```
{
  "long_name": "K1R 7Y5",
  "short_name": "K1R 7Y5",
  "types": [
    "postal_code"
  ]
},
"adr_address": 
  <span class="street-address">101 Lyon St N</span>,
  <span class="locality">Ottawa</span>,
  <span class="country-name">Canada</span>,
"formatted_address": "101 Lyon St N, Ottawa, ON K1R 7Y5, Canada",
"formatted_phone_number": "+613-257-3586",
"geometry": {
  "location": {
    "lat": 45.4187422,
    "lng": -75.7945078
  },
  "viewport": {
    "neast": {
      "lat": 45.4280229302915,
      "lng": -75.78316697085
    },
    "southwest": {
      "lat": 45.417324867985,
      "lng": -75.7660149302915
    }
  }
}
```

*Figure 11 - Example of hotel information provided by Google's API.*

The Instagram API provides images that are linked to the destination. For the up to 20 users, the Instagram API return the images with the correspondent tag, making this useful to link Instagram accounts to the application when relevant.
4.8. iOS Implementation

The client-side of the application was implemented using XCode and Swift 4.0. The client-side role on the application is to deliver a nice user experience while extracting all the information needed.

The first information that is captured when the user opens the application is the UDID, that is the unique id from that specific device. Once the UDID is captured it POST a request to the server informing the parameter, the server then checks if the device is already in the database and adds it in case of first occurrence.

The second information captured by the iOS application is the location of the user. Once the app is opened, the user is prompted to give access to the location of the device while using the app, once the permission is given the location is stored as a variable to be submitted together with the question’s answers.

![Swift code that ask permission to use location](image)
These first two information’s are collected in the background every time the user opens the application and in the case of the coordinates, these are updated every second when the application is in foreground.

To receive the input of the user when asking the questions, the application uses 2 types of interaction, the regular button provided by XCode in the UIKit and the “stepper-button” that was created specifically for that application so users can give non-binary answers for the questions:

![Stepper button example](image13.png)

![UIKit button example](image14.png)

Figure 13 - "Stepper button" example

Figure 14 - UIKit button example
Once all the questions are answered the application send the information to the server using the REST API. This is done with the help of a framework called Alamofire that help Swift to parse JSON and organize the response.

```
let parameters: Parameters = {
    "adventure": Global.adventure,
    "romantic": Global.romantic,
    "relaxing": Global.relaxing,
    "exploring": Global.exploring,
    "sportive": Global.sportive,
    "budget": Global.budget,
    "time": Global.time,
    "distance": Global.distance,
    "lat": Global.lat,
    "long": Global.long,
    "likeMost": Global.likeMost
}

    response in
    //print(response.value as? Dictionary<String,AnyObject>)
    do{
        if let json = response.result.value as? Dictionary<String,AnyObject>
        for i in json.resultArray{
            let activities = ["activities"] as! [AnyObject]
            let hotels = ["hotels"] as! [AnyObject]
            let destination = DestinationClass()
            for a in activities{
                let ac = ActivityClass()
                ac.name = (a["name"] as? String)
                ac.tipo = (a["tipo"] as? String)
                ac.id = (a["id"]) as? Int
                ac.id_place = (a["id_place"] as? Int)
                ac.id = (a["id"] as? Int)
                ac.ranking = (a["ranking"] as? Int)
                destination.activities.append(ac)
            }
            for h in hotels{
                let ht = HotelClass()
                ht.name = (h["name"] as? String)
            }
        }
    }
```

Figure 15 - POST request using Alamofire

4.9  Deploying the application using Heroku

To make the application available to the general public the NodeJS server was setup in a hosting platform called Heroku, this can be done with these two steps:

1. **Account Setup:** The process of uploading your server is rather simple. First you need to create a free account in [http://www.heroku.com/](http://www.heroku.com/). You will receive a confirmation email to finish up the registering process. Once this is done, after the first login, you will be presented with the following screen:
2. **Create and deploy the app:** The only information needed to create an application is the name and where you want to host. After creating the app you are ready to deploy. A few deployment options are offered in Heroku, for this project a Heroku Git using the CLI was chosen. In order to use the CLI you have to download it and install it in your machine, details are provided by Heroku in this link [https://devcenter.heroku.com/articles/heroku-cli](https://devcenter.heroku.com/articles/heroku-cli). Once the CLI is installed you are now able to access it via Command Line. Using the CL, navigate to the root folder of your application. Once there, type `heroku login` and fill in your credentials. If it is your first time deploying the application you will need to create a Git repository, this can be done by typing the following commands:

```bash
$ git init
$ heroku git:remote -a name_of_your_app
```

With the Git repository ready, you can now deploy your application by typing:

```bash
$ git add .
$ git commit -am "any relevant comment to this commit"
$ git push heroku master
```

The `add .` command performs the indexation of the files in the project, the `commit` command makes the changes in your local repository and the `push` command pushes these changes to the remote repository, in this case, to Heroku.

Heroku provides free hosting for the application with some limitations, which is, per the company, ‘Ideal for experimenting with cloud applications’. To host persistent data, the company offers a wide range of add-ons. For this project ClearDB MySQL add on was used to host the MySQL database.
5. Evaluation

5.1 Accuracy Study

The goal of this application was to provide an easy-to-use and fast recommender system for travelers that are looking to find new destinations to go. In order to evaluate the accuracy of this application we created an accuracy study.

In the study, we presented the application for 18 different volunteers in 2 different countries. After letting the users use the application, we asked them to rate the quality of the destination proposed by the system based on the answers that they gave to the questions.

We asked them to rate 1 if the proposed destination matches their expectation, 0 if the proposed destination does not match their expectation or N/A if they are not confident enough that the presented destination matches their expectation. From this study, we obtained the following data set:

<table>
<thead>
<tr>
<th>User</th>
<th>Quality of First Recommendation</th>
<th>Quality of Second Recommendation</th>
<th>Quality of Third Recommendation</th>
</tr>
</thead>
<tbody>
<tr>
<td>User 1</td>
<td>1</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>User 2</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>User 3</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>User 4</td>
<td>1</td>
<td>N/A</td>
<td>1</td>
</tr>
<tr>
<td>User 5</td>
<td>1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>User 6</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>User 7</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
</tbody>
</table>
In order to assess the accuracy of the system, we excluded the N/A results from the statistic. Given the resulting table we had 43 positive feedbacks and 6 negative feedbacks resulting in an accuracy of approximately **87 percent**.

### 6 Conclusion

#### 6.1 Summary

The application provides a fast, easy and friendly way to find new destinations to travel, and gives easy access to all the information needed to start planning a trip. Combining NodeJS and Swift technology we were able to create a recommender system that is truly based on the user’s feelings to deliver with 87 percent of confidence his next travel destination. When developing this project, we were able to achieve this goal using a portable solution that can be
easily exported for other devices maintaining or improving the structure and accuracy achieved in the original project.

### 6.2 Future Work

The goal is to make this application available in the app store and as a web application covering the countries of Canada and Brazil. The project provides a wide range of possible improvements that ought to be implemented in the near future: Add collaborative filtering, Android and Web Clients, on-app booking, increase the number of destinations available.

**Add Collaborative Filtering**

One possible improvement is to add collaborative filtering to the recommender system. In order to add this feature, a rating feature would have to be implemented in the destination view. Having users rating destinations allows the system to establish a relationship between users and destinations and to improve the accuracy of the recommendation.

**Android and Web Clients**

Android owns approximately 82% of the mobile phone market share so creating an android version of the application is a need if the goal is to make the application widely available. To create a responsive web client is another interesting solution because it can be accessed by any device connected to the internet. None of these improvements are extremely complicated to implement once all the intelligence of the system lies on the server side.

**On-App booking**

With the idea of making the life of the traveler easier, an on-app booking system can be implemented. This would allow users to book reservations on hotels without leaving the application. A possible approach is to implement a connection with an external API that offers the booking services.
Increase the number of destinations available

Probably the most important improvement to be made is to increase the numbers of destinations in the database. The project started with 40 destinations in Canada and 200 in Brazil. By increasing this number the chances of a repeated destination thus decrease and the accuracy of the system will automatically increase because there is a bigger chance of the top destinations to have a higher score in the system.

7. References

Jeffrey D. Ullman – “Mining Massive Datasets”, Chapter 9:

Faisal Rahutomo, Teruaki Kitasuka, Semantic Cosine Similarity:
https://pdfs.semanticscholar.org/41ff/3934f40c32ac8643270822de1c763e16c71b.pdf


https://docs.oracle.com/javaee/6/tutorial/doc/gijqy.html