C3879C Capstone Project

Yelper Assistant

Date of Submission: 31-JUL-2019

Submitted By:

17060167 LIM YUAN HER

School of Infocomm (SOI)

Republic Polytechnic

ACKNOWLEDGEMENTS

I would like to take this opportunity to thank all our instructors, especially Mr. Andy Lee and Mr. Tan Poh Keam for their excellent tutelege and guidance for the Recommender Systems and Virtual Assistants part of the course. This has provided me with a strong foundation and understanding of these two specialist topics necessary for the execution and successful completion of this project.

TABLE OF CONTENTS

TABLE OF FIGURES

LIST OF TABLES

ABSTRACT

This report documents the design of a virtual assistant application powered with a recommendation engine for the Yelp search service . This system supplements the existing Yelp website and expands user reach and exposure through the use of social media platforms and customized recommendations. With the successful proof-of-concept implementation of this system, it is expected that the expanded interactivity will attract more users to this search service.

1 Introduction

Yelp is a search service using crowd-sourced reviews about local businesses. Besides reviews, it also facilitates searching for events, lists and communication between Yelp users.

Being a single-platform (web-based), the user interface has not kept pace with the advancement of modern technologies e.g. social media etc. and this has limited the scope of reach to potential users.

In addition, there are 2 major isuses with the user interface design that needs to be addressed by this system, and they are:

1. Currently, the reviews display functionality on the Yelp website provides keyword-based search results, and thus, the user need to manually sift through the results individually to find the one closest to their intended search, thus making it a tedious and time-consuming process.

2. All information is currently scattered throughout different subsections of the website, thus making it difficult for the user to search for the information required.

In view of the above problems identified, the objective of this project is twofold: The scope of this project is twofold:

1. To develop a personalized recommender based on reviews data to reduce time spent on searching for relevant information.

2. To provide a virtual assistant interface to allow users to query for the personalized recommendations and to facilitate searching for required information

2 Project Specification and Plan

2.1 Project Overview

2.1.1 Objective

The main objective of this project is to develop a virtual assistant and recommendation system for the Yelp search website.

2.1.2 Scope

In order to ensure the timeliness of the project deliverables due to the short duration of this project (less than 2 months), the focus will be on developing a prototype to evaluate the feasibility of the project objectives identified in order to achieve a minimum viable product. Thus, the emphasis will not be on developing a ready for commercial production system, but on system proofing the viability of the system.

2.1.3 Assumptions

It is assumed that cloud hosting of the system components will be necessary due to the web-based nature of the system. Trial or free accounts will be used for platform hosting and development in order minimize the costs involved.

For the virtual assistant component of the system, although various BOT frameworks e.g. Microsoft Bot Framework are available, Google DialogFlow is used as it is taught during the course.

For the recommender component of the system, Spark MLLib, Scikit Surprise, LightFM open source packages will be evaluated and the most suitable used in this project. Paid auto-recommender services e.g. Amazon Personalize, Google Recommendations AI etc. will not be considered. In addition, to minimize computational resources and costs involved, the original Yelp Open Dataset will be trimmed down to a manageable size.

2.2 Functional Requirements

The diagram below illustrates the overall requirement for the Smart Water Quality Monitoring System:

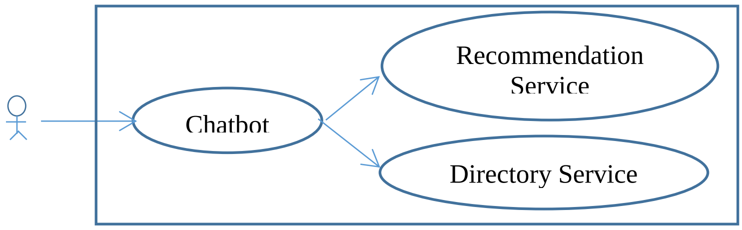


Figure 1 – Functional Requirement Use Caase

The subsequent sections below list the specific functional requirements of the system.

2.2.1 Functional Requirement 1

The. system will have a recommendation engine to provide personalized recommendation

2.2.2 Functional Requirement 2

The system will have a virtual assistant with social media integration e.g. Slack.

2.3 Project Plan

The figure below lists the project tasks and time allocated for each including the project milestone deliverables:

Task	Week No														
	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Data Cleaning/ Exploration															
Recommender algorithm analysis and evaluation															
Recommender Development															
Chatbot Development															
Recommender/ Chatbot Integration															
Testing															
Demo Preparation/ Presentation															

Figure 2 – Project Schedule

3 Business Analysis

3.1 Business Issues

The current reviews-only site at Yelp faced the following challenges that needs to be addressed in the design of the system:

1. **Ease of access** – The site adopts the traditional menu type access whereby the user needs to search for the exact option to fulfill his/her search requirement. Although a search functionality is available to shorten this process, the subsequent search results are numerous and the user has to sift through the results to find the ones relevant to his/her search. These factors result in a hidden barrier between the system and the user and increases the time needed by the user.

2. **Results Differentiation** – The site provides seemingly generic information that may or may not be relevant to the user. This will turn off the user who will search for other sites that can provide customized results.

3. **User Engagement** – The site content layout adopts a traditional approach more suited for a desktop browser experience and lacks the personalized touch associated with modern interfaces.

These are major issues that the system will address in its system design and implementation.

3.2 Market Analysis

The competitive landscape for online local business ratings and reviews platforms is shared by the prominent players such as Google, Facebook, Yelp, Foursquare, and TripAdvisor [1]. All have experienced tremendous growth for the past number of years, with Google strongly outpacing the others.

The behavioural shift observed throughout the last few years is that reviews have migrated from review-only sites like Yelp or TripAdvisor to a much bigger space on social media (Facebook/ Google) [2]. The frictionless access to social media sites like Facebook/ Google on web and mobile plaforms make it a seamless experience for users, thus, helping to generate more reviews traffic on these multi-platform providers. Furthermore, in a joint study by Google/Nielsen, 93% of people who use mobile to research go on to complete a purchase of a product or service [3]. This highlights the importance of mobile as a platform of choice for modern users.

Thus, it is imperative that review-only sites e.g. Yelp need to adapt to the changing landscape by adopting mobile technologies and social media reach in order to retain existing users and to solicit new users to their platforms.

The rapid adoption of chatbots as a interaction mechanism on mobile platforms is prompted by the fact that chatbots are particularly well suited for mobile interfaces, perhaps more so than apps as messaging is at the heart of the mobile experience [9]. In fatct, by outfitting social messaging channels with chatbots, a business creates further growth prospects as prompt feedback to businesses through this channel and the expectation of improved customer care form a virtuous cycle benefitting both businesses and customers [10].

3.3 Business Solutions

To address the issues identified in Section 3.1, the system incorporates several major features that target to eliminate the blocking issues:

1. A virtual assistant is incorporated to provide a natural language interface for interactting with the system. This reduces the time spent searching for the relevant menu/ option in the reviews-only site and also sifting through the long list of search results returned from keyword-based search.

2. A recommender engine is incorporated to provide personalized recommendations based on a user’s prevous rating behaviour and similarity to other users. This provides an avenue for users to look up businesses that they might not have searched for using traditional search and quickly filters out non-relevant businesses that the user most probably will not be interested in.

4 System Design and Implementation

4.1 System Architecture

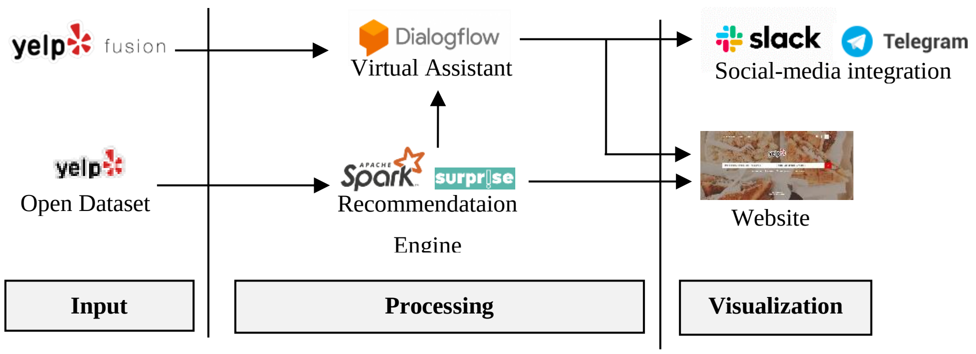


Figure 3 – System Architecture

The figure above illustrates the overall system architecture, which comprises the following components:

1. The yelp open dataset is used to power the recommendation engine used to provide customized recommendations for the user.
2. The yelp fusoin API acts as a fulfillment engine for the business and events directory information portal fot the DialogFlow-based chatbot.
3. Slack and Telegram are used as the social media integration platforms for DialogFlow and is the main user interface for interacting with users.
4. Scikit-Surprise, a Python scikit building and analyzing recommender systems, is used to build the recommendation model used to serve customied recommendations to users.
5. Google DialogFlow, a conversation system development platform with machine learning and natural language processing (NLP) capabilities, is used to power the chatbot.

4.2 Detailed System Design

4.2.1 Introduction

The virutal assistant comprises of 2 main components:

1. A Reviews Recommendation engine to provide personalized recommendation.

2. A virtual assistant using Google DialogFlow with fulfilment provided by the Yelp Fusion REST API and recommendation engine for serving events/services information and personalized recommendations respectively.

Both web-based and social media integration e.g. Slack, Telegram will be developed

4.2.2 Use Cases

The following illustrates the typical use cases fulfilled by the system.

4.2.2.1 Search Business

This use case allows the user to search for businesses using the following criteria:

1. **Category** – category filters e.g. Bike Rental, Bakeries, breweries etc.
2. **Name** – business name identifier e.g. Starbucks
3. **Phone** – business contact number, including country/area code e.g. +19147137865
4. **Price** - Price Level indicator e.g. 1 for $, 4 for $$$$, 2,3 for $$ and $$$ etc.
5. **Business ID** –business identifier e.g. MpF9j5-fBH0H6L9AzzyArA

4.2.2.2 Search Businesses with Food Delivery services

This use case allows the user to search for a business served by food delivery services.

4.2.2.3 Search for Businesses with specific characteristics

This use case allows the user to search for businesses with specific characteristics

1. **Hot and New -** popular businesses which recently joined Yelp
2. **Cashback -** businesses offering Yelp Cash Back to in-house customers
3. **Deals -** businesses offering Yelp Deals on their profile page
4. **Wheelchair Accessible -** businesses which are Wheelchair Accessible
5. **Reservation** - businesses with Yelp Reservations bookings enabled on their profile page
6. **Waitlist Reservation -** businesses with Yelp Waitlist bookings enabled on their profile screen (iOS/Android)

4.2.2.4 Get Recommendations

This use case allows the user to search for customized recommendation based on his/her Yelp user ID.

4.2.2.5 Featured Events

This use case allows the user to search for events as chosen by Yelp's community managers.

4.2.3 Implementation

The diagram below illustrates the hosting platforms used for the deployment of the various components of the sysem:

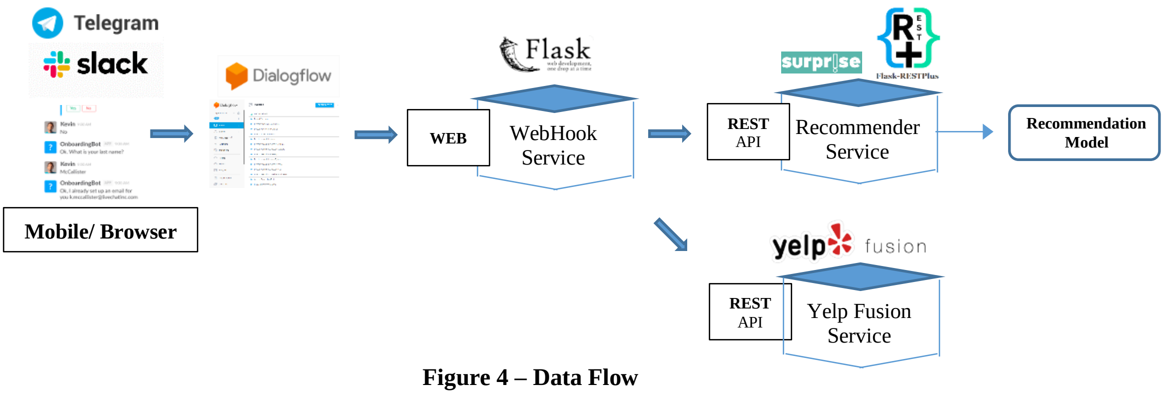


Figure 4 – Data Flow

The system uses a microservice architecture with 3 services:

1. **WebHook Service** – Fulfilment engine for DialogFlow webhook
2. **Recommender Service** – Recommendation engine using Scikit-Surprise for serving customized recommendations
3. **Yelp Fusion API Service** – Directory service using Yelp Fusion API for accessing Yelp directory information

The interaction entry point is via the slack workspace created at https://yelperassistant.slack.com, which is accessible via a web browser or via the slack mobile app on mobile devices. The slack workspace is integrated with a DialogFlow chatbot, which is used for developing the chatbot interface.

4.2.4 Deployment

The WebHook service is hosted on the Heroku platform at https://yelperassistant-wh.herokuapp.com,

The Yelp Fusion API Service is hosted by Yelp and is available via https://api.yelp.com

The Recommender service is hosted on the Heroku platform at https://yelperassistant-rec.herokuapp.com/api/v1.

4.2.5 Recommendation Engine Design

4.2.5.1 Dataset

The Yelp open dataset [4] was used as input data to build the model for generation of recommendations for users. For the purpose of this project, 3 of the data files were used as detailed in the table below:

s/n	Name	Description	Size
1	business	Contains business information e.g. address, attributes, operating hours etc.	192K
2	review	Contains review ratings for businesses by users	6.7M
3	user	Contains user information e.g. name, # of reviews, average rating etc.	1.6M

Table 1 – Yelp Open Dataset

The json to csv converter from the examples website [7] was used to convert the .json files to csv format to facilitate data manipulation and analysis.

As the purpose of this project is to develop a prototype to evaluate the feasibility of the project objectives identified and to minimize the computational costs involved, **20K reviews** were extracted from the original review dataset to facilitate development using a CPU-powered machine.

4.2.5.2 Algorithm Evaluation

2 open source recommender packages were evaluated, namely Spark MLLib [5] and Scikit-Surprise [6].

ALS (alternating least squares) algorithm from the Spark MLLib package is a matrix factorization algorithm that uses Alternating Least Squares with Weighted-Lamda-Regularization (ALS-WR) to reduce user-to-item ratings matrix into a user-to-feature and item-to-feature matrices to uncover the latent factors that explain the observed user to item ratings and tries to find optimal factor weights to minimize the least squares between predicted and actual ratings.

Scikit Surprise is a easy-to-use Python scikit for recommender systems that implements many of the recommender algorithms e.g. collaborative filtering, matrix factorization, kNN (k-Nearest Neighbours)-based etc.

The table below summarizes the recommender algorithms evaluated and the resulting RMSE (Root Mean-Squared Error) scores:

s/n	Algorithm	Description	RMSE
1	KNNBasic	Basic collaborative filtering algorithm	1.4678
2	KNNWithMeans	Basic collaborative filtering algorithm, taking into account the mean ratings of each user	1.4906
3	KNNWithZScore	Basic collaborative filtering algorithm, taking into account the z-score normalization of each user	1.4906
4	SVD	SVD algorithm popularized by Simon Funk during the Netflix Prize	1.4344
5	SVDpp	SVD++ algorithm that is an extension of SVD but taking into account implicit ratings	1.4309
6	NMF	Collaborative filtering algorithm based on Non-negative Matrix Factorization	1.5043
7	ALS	Matrix factorization algorithm that uses Alternating Least Squares with Weighted-Lamda-Regularization (ALS-WR)	3.9320

Table 2 – Recommender Algorithm Evaluation Results

Based on the tabulated results above, the SVDpp algorithm from the Scikit-Surprise library is selected as the recommender algorithm used to build the recommendation model. Refer to Appendix 1 for the Jupyter Notebook analysis.

4.2.5.3 Recommender Implementation

The recommender is designed to cater for 3 types of users:

1. New users who have not given any business ratings (cold-start problem)
2. Users with few given ratings (less than 5)
3. Active users who have given more ratings (more than 5)

Thus, for each type of user identified above, a specific recommender model is implemented as described below.

4.2.5.3.1 Popularity-Based Model

This provides generic non-personalized recommendations to all users assuming no information about the user is available. The top 5 businesses with the highest number of user ratings and average rating value is recommended to the user.

4.2.5.3.2 Content-Based Model

This uses the "categories" attribute e.g. for each business (see below illustration) available in the business dataset to calculate the similarity scores between businesses using similarity measures e.g. cosine-similarity, correlation etc.

	business_id	categories
0	7340	Golf, Active Life
1	82973	Specialty Food, Restaurants, Dim Sum, Imported...
2	134808	Sushi Bars, Restaurants, Japanese
3	186365	Insurance, Financial Services
4	56458	Plumbing, Shopping, Local Services, Home Servi...
5	21350	Shipping Centers, Couriers & Delivery Services...
6	18832	Beauty & Spas, Hair Salons
7	134267	Hair Salons, Hair Stylists, Barbers, Men's Hai...
8	105796	Nail Salons, Beauty & Spas, Day Spas
9	15664	Beauty & Spas, Nail Salons, Day Spas, Massage
10	131422	Local Services, Professional Services, Compute...

The similarity scores are used to find the top 5 similar businesses for each of the business that were rated by the user of interest. The combined list of businesses is then sorted by the aggregated count of each business in the list, and the top 5 business (by count) is recommended to the user.

4.2.5.3.3 Matrix Factorization-Based Model

This uses the SVD++ matrix factorization algorithm and leverages a latent factor model to capture the relationship between users and businesses. The *SVDpp* class from the *Scikit-Surprise* package is used to derive 2 matrices (user-factor and business-factor) that maps the users/businesses to latent factors that can be used to predict ratings for businesses that the user has not rated yet. However, it is incapable of modeling new users unless the whole model is retrained, thus rendering it not suitable for new users in which case, the popularity-based model is used instead.

4.2.5.3.4 Recommender Workflow

The diagram below illustrates the process that the system follows when a user makes a request for recommendations:

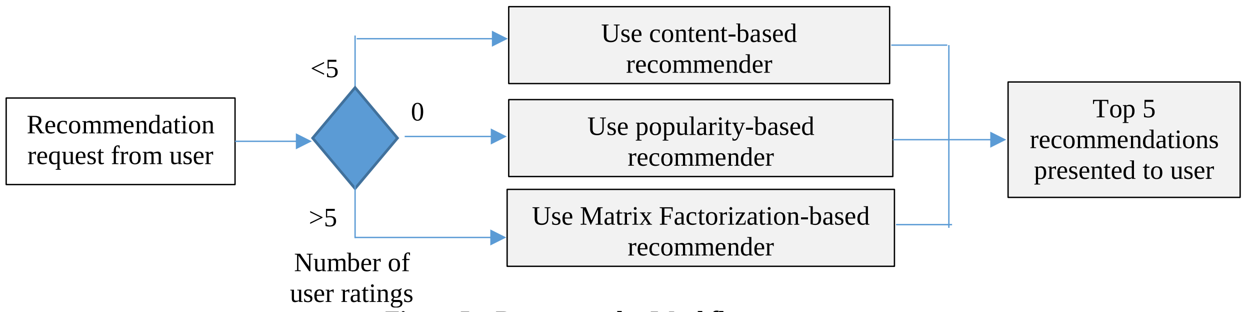


Figure 5 – Recommender Workflow

4.2.5.4 Recommender Model Persistence

The recommender models are saved to disk when manual update requests are made. The purpose of saving the model outputs is for subsequent quick loading when requests for user recommendations are made instead of having to rebuild the model again for each request. The table below summarizes the model saved filename and stored information:

s/n	Recommender Type	Model Filename	Model Info
1	Popularity-based	PopularityModel.pkl	For new users
2	Content-based	TfidfRecommender.pkl	For users with <5 ratings
3	Matrix Factorization-based	SurpriseSVDppRecommender.pkl	For users with >5 ratings

Table 3 – Recommender Model Information

4.2.5.5 Recommender Update

The recommender models need to be re-trained and updated to incorporate new user/ratings information. To facilitate this operation, the recommender micro-service includes a updateModels() API method that can be invoked via a scheduled cron job (batch) or manual request (on-demand).

4.2.6 Chatbot Design

4.2.6.1 Introduction

In this project, Google DialogFlow [8] is chosen as the natural language processing engine which incorporates machine learning to provide engaging and human-centred conversational interfaces with the user.

4.2.6.2 Interface Design

The chatbot is designed to be Q&A type in order to provide quick digestible search results with navigable links to more detailed information if required:

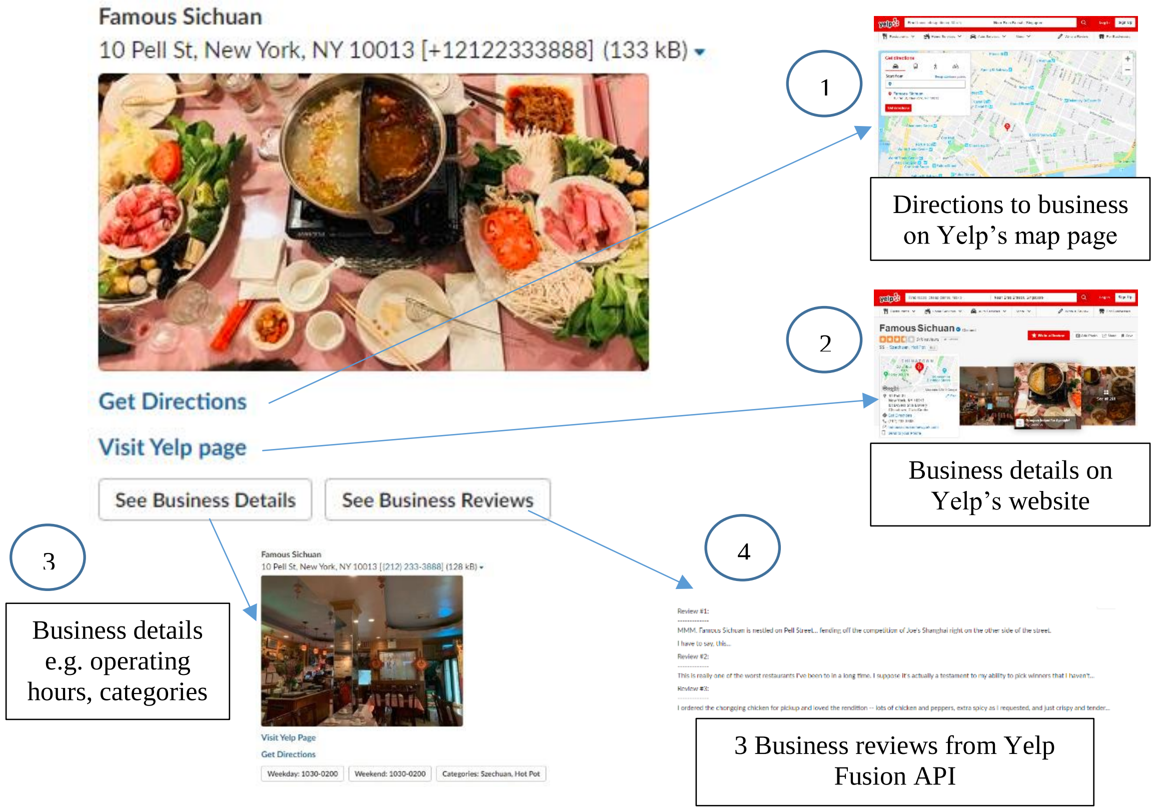


Figure 6 – Display format for typical search result

4.2.6.3 Conversation Design

The conversation flow of the chatbot is divided into both main and alternate flow paths to cater for all encountered situations.

4.2.6.3.1 Main

The main conversation flow is designed such that the user is presented with 5 options corresponding to each use case as detailed in section 4.2.2. Buttons are used where appropriate to minimize input error.

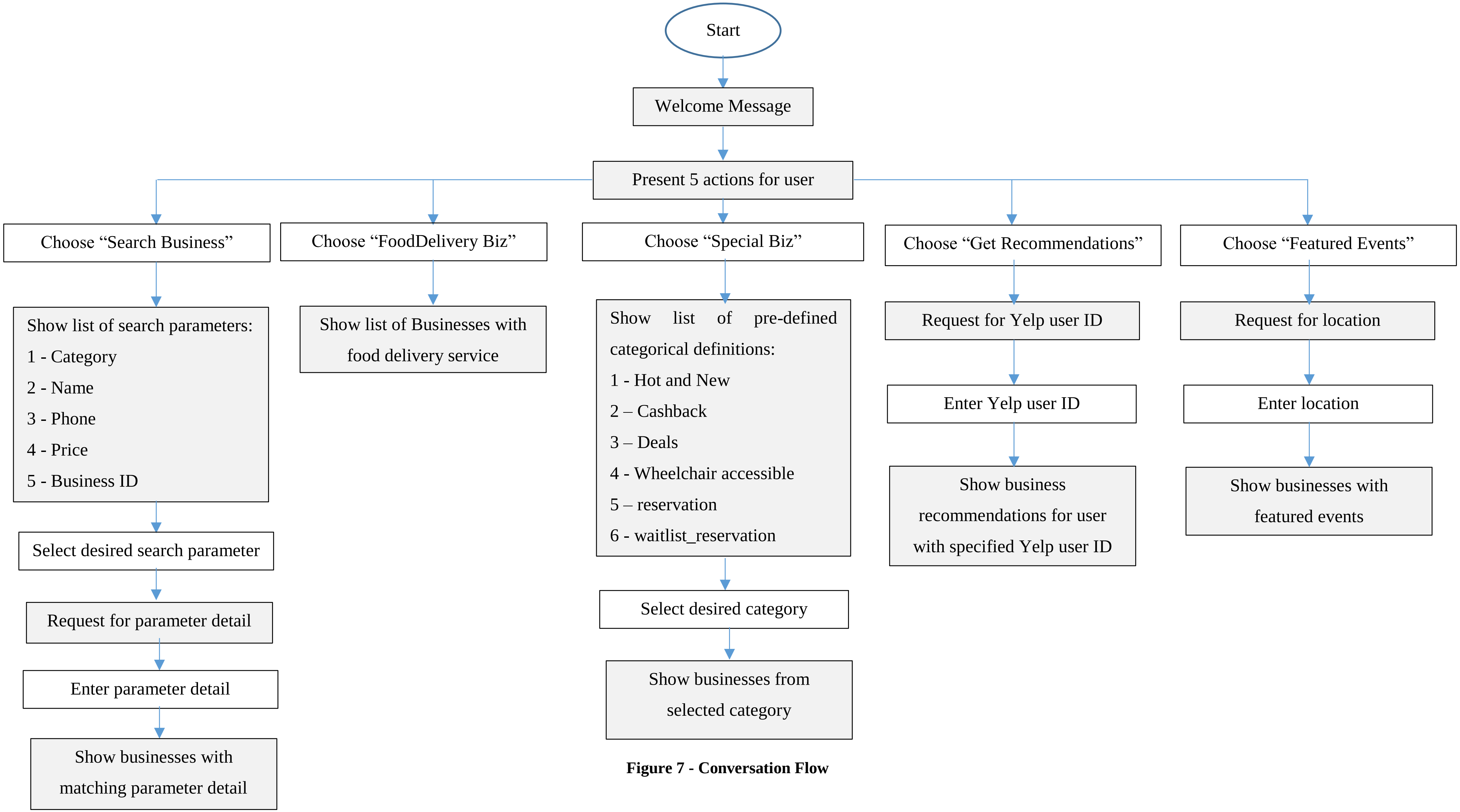


Figure 7 - Conversation Flow

4.2.6.3.2 Alternate

In case the user deviate from the main conversation flow path e.g. type in invalid selection etc., the chatbot will inform the user to re-select the service required through the Default.Fallback intent.:

4.2.6.4 Implementation

4.2.6.4.1 Intents

For each action as shown in the Conversation Flow diagram (Figure 5), an intent is created to service the request from the user as listed below:

s/n	Action	Intent	Purpose	Parameters
1	-	Default.Welcome	Greet user and offer services	-
2	-	Default.Fallback	Handles unexpected users' responses	-
3	-	General.ConnectionTest	Tests connection to webhook service	-
4	getBusinessReviews	Directory.getBusinessReviews	Gets 3 reviews for specified business	businessID
5	getFeaturedEvents	Directory.getFeaturedEvents	Gets 3 featured events for specified location	latitude, longitude, location
6	matchBusiness	Directory.matchBusiness	Finds matching businesses with search parameters	See (1)
7		Directory.searchBusiness	"Search Business" parameter listing	
8	searchBusiness	Directory.searchBusinessByAttribute	Gets businesses with matching attribute	See (1)
9	searchBusiness	Directory.searchBusinessByCategory	Gets businesses in specified category	See (1)
10	getBusiness	Directory.searchBusinessByID	Gets businesses with specified ID	businessID
11	searchBusiness	Directory.searchBusinessByName	Get business with matching name	See (1)

s/n	Action	Intent	Purpose	Parameters
12	searchBusinessByPhone	Directory.searchBusinessByPhone	Gets business with matching phone #	phone
13	searchBusiness	Directory.searchBusinessByPrice	Gets business with matching price ranges	See (1)
14	searchFoodDeliveryBusinesses	Directory.searchFoodDeliveryBusinesses	Gets buseinsses with food delivery services	latitude, longitude, location
15	getRecommendations	Rec.getRecommendations	Gets recommendations for specified user	user_id

Table 4 - Action-Intent Mapping

(1) term, latitude, longitude, radius, locale, offset, sort_by, price, open_now, open_at, attributes, limit, location, categories

4.2.6.5 Entities

The list of entities created for are as defined below:

s/n	Entity	Description	Examples
1	categories	List of categories for "Directory.searchBusinessByCategory" intent	bikerentals, dentists, chiropractors, beaches, bbq

Table 5 – Entity List

4.2.6.6 Context

The "Directory.searchBusinessByCategory", "Directory.searchBusinessByName", "Directory.searchBusinessByPhone" and "Directory.searchBusinessByPrice" intents are assigned one input context "searchBusiness" to control the conversation flow to start from the "Directory.searchBusiness" intent.

4.2.6.7 Fulfilment

Webhook for fulfilment of Yelp directory information search and recommendations is developed as a Flask web application and hosted on https://yelperassistant-wh.herokuapp.com.

4.2.6.8 Integration

This chatbot is enabled for Slack and Telegram integration. The Slack messaging workspace is at https://yelperassistant.slack.com whereas for Telegram is at https://web.telegram.org/#/im?p=@YelperAssistantbot

5 System Testing

5.1 Functional Testing

The following lists the functional tests conducted to verify the functional requirements of the system as identified in section 2.2.

Test Specification ID : TS1
Name of Tester : Lim Yuan Her
Use Case ID : UC1
Date of Test : 29 July 2019
Description of Test : Verify system functional requirements

S/No	Test Case	Expected Result	Pass/ Fail	Remarks
Functional Requirement 1 (Section 2.2.1)				
1.	Check \recommendation availability by selecting 'Get Recommendations" from the virutal assistant default display	5 recommendations are provided	Pass	
Functional Requirement 2 (Section 2.2.2)				
1.	Check virtual assistant is accessible from social media platforms with user greeting and services available displayed.	Slack Messaging Workspace https://yelperassistant.slack.com	Pass	
		Telegram Messaging https://web.telegram.org/#/im?p=@YelperAssistantbot	Pass	
2.	Check the functionality of each use case detailed in section 4.2.2 is available from the virutal assistant	Search Business	Pass	
		Search Businesses with Food Delivery services	Pass	
		Search for Businesses with specific characteristics	Pass	
		Get Recommendations	Pass	
		Featured Events	Pass	

Table 6 – Functional Test Cases

6 User and Technical Documentations

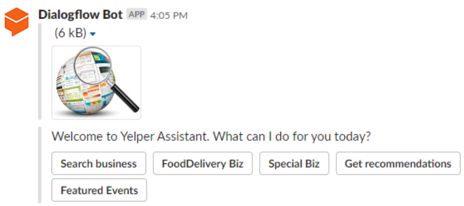
6.1 User Documentation/Guide/Manual

6.1.1 Introduction

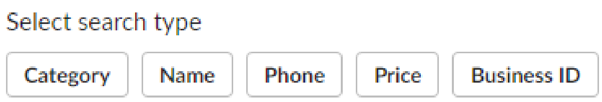
This section details the steps to access the various functions of the virtual assistant.

6.1.2 Search Business

1. Type “Hi” in the virtual assistnat interface. The virtual assistant default greeting and services will be displayed:



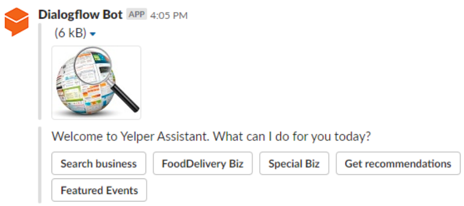
2. Click on “Search Business”. The search type menu will be displayed:



3. Click on the desired search parameter type e.g. Category.A prompt will be displayed requesting for the parameter details/
4. Enter the parameter details e.g. “bikerentals” for category etc.
5. The matched businesses details will be displayed with navigable links to further information as specified in section 4.2.6.2.

6.1.3 Search Businesses with Food Delivery services

1. Type “Hi” in the virtual assistnat interface. The virtual assistant default greeting and services will be displayed:

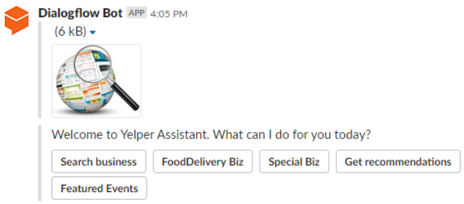


1. Click on “FoodDelivery Biz”. A prompt requesting for the location will be displayed.
2. Enter the location information e.g. NYC.

2. The matched businesses details will be displayed with navigable links to further information as specified in section 4.2.6.2:

6.1.4 Search for Businesses with specific characteristics

1. Type "Hi" in the virtual assistnat interface. The virtual assistant default greeting and services will be displayed:



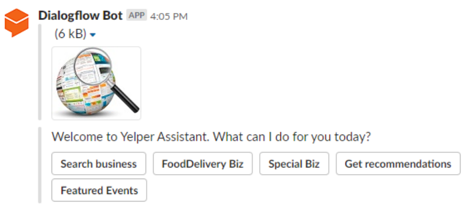
2. Click on "Special Biz". The attributes menu will be displayed:

1 - Hot and New, 2 - Cashback, 3 - Deals, 4 - Wheelchair accessible, 5 - reservation, 6 - waitlist_reservation

3. Enter the number corresponding to the desired search attribute e.g. Hot and New. A prompt will be displayed requesting for the parameter details/
4. Enter the parameter details e.g. "NYC" for location etc.
5. The matched businesses details will be displayed with navigable links to further information as specified in section 4.2.6.2.

6.1.5 Get Recommendations

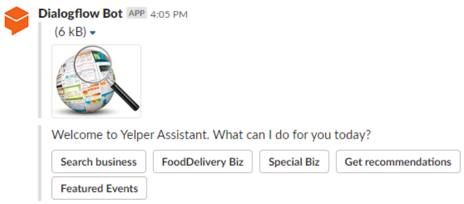
3. Type "Hi" in the virtual assistnat interface. The virtual assistant default greeting and services will be displayed:



4. Click on "Get recommendations". A prompt requesting for the Yelp user ID will be displayed.
5. Enter the Yelp user ID corresponding to the desired user.
6. The recommended businesses details will be displayed with navigable links to further information as specified in section 4.2.6.2:

6.1.6 Featured Events

3. Type "Hi" in the virtual assistnat interface. The virtual assistant default greeting and services will be displayed:



7. Click on “FoodDelivery Biz”. A prompt requesting for the location will be displayed.
8. Enter the location information e.g. NYC.
4. The matched businesses details will be displayed with navigable links to further information as specified in section 4.2.6.2:

6.2 Technical Documentation (Installation guide/Manual)

6.2.1 Introduction

This section details the installation steps to install/ access the virtual assistant.

6.2.2 Desktop/Tablet Access

Basic web browser e.g. Internet Explorer is installed by default in desktop/ tablet. No additional installation is required. Navigate to https://yelperassistant.slack.com (for Slack) and https://web.telegram.org/#/im?p=@YelperAssistantbot (for Telegram) to access the virtual assistant.

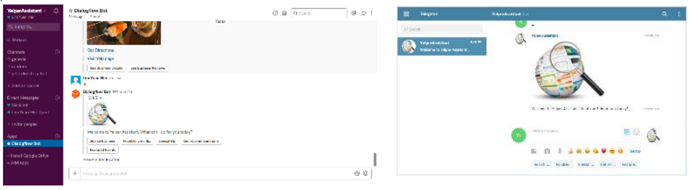


Figure 8 – YelperAssistant Browser Interface

6.2.3 Mobile App Installation

1. Install the Slack mobile App from the Google Playstore.
2. Add “yelp-directory-bot” channel to the Slack mobile App.
3. Add the “DialogFlow Bot” app to the “yelp-directory-bot” channel.

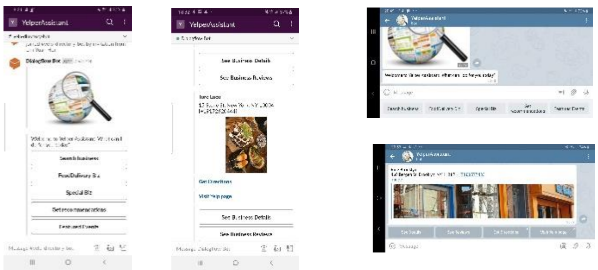


Figure 9 – YelperAssistant Mobile Interface

7 Conclusions

7.1 Introduction

This report detailed the design and implementation results of a virtual assistant for Yelp using Google DialogFlow using Slack messaging platform as the conversational interface. The Yelp directory information is provided by Yelp Fusion API and customized recommendations are powered by a recommendation engine using Scikt-Surprise open source recommendation library implemented as a REST API based microservice.

7.2 Further Enhancements

Some further system enhancements that could be considered for future work are as listed below:

- Integrating with other social messaging platforms e.g. Facebook Messenger, LINE etc. and physical virtual assistant devices e.g. Google Assistant, Amazon Alexa etc.

- Perform sentiment analysis based on business textual reviews for ratings verification

- Use Baynesian ranking for popularity-based recommender model to take into account number of ratings and average rating given for specific business instead of current simple aggregated count sorting method

- Perform anomaly detection for detecting shilling attacks (manipulation of recommendation rankings)

- Integrate Yelp's security framework to facilitate business rating/review directly from chatbot

8 References

[1] R. Murphy, *Comparison of Local Review Sites: Which Platform is Growing the Fastest?*, https://www.brightlocal.com/research/comparison-of-local-review-sites.

[2] *2018 ReviewTrackers Online Reviews Survey*, https://www.reviewtrackers.com/reports/online-reviews-survey.

[3] Diana Kaemingk, *20 online review stats to know in 2019*, https://www.qualtrics.com/blog/online-review-stats.

[4] Yelp, *Yelp Open Dataset*, https://www.yelp.com/dataset

[5] Apache Spark, *pyspark.mllib package*, https://spark.apache.org/docs/2.4.2/api/python/pyspark.mllib.html#pyspark.mllib.recommendation.ALS

[6] Scikit-Surprise, http://surpriselib.com/

[7] Samples for users of the Yelp Academic Dataset, https://github.com/Yelp/dataset-examples

[8] Google DialogFlow, https://dialogflow.com/

[9] Business Insider Intelligence, *80% of businesses want chatbots by 2020*, https://www.businessinsider.com/80-of-businesses-want-chatbots-by-2020-2016-12?IR=T

[10] Chatbots Magazine, *Chatbot Report 2019: Global Trends and Analysis*, https://chatbotsmagazine.com/chatbot-report-2019-global-trends-and-analysis-a487afec05b

9 Appendices

9.1 Appendix 1

Install Surpriselib/ Environment Preparation

In [0]:
```
!apt-get install openjdk-8-jdk-headless -qq > /dev/null
!wget -q https://archive.apache.org/dist/spark/spark-2.4.2/spark-2.4.2-bin-hadoop2.7.tgz
!tar xf spark-2.4.2-bin-hadoop2.7.tgz
!pip install -q findspark

!pip install scikit-surprise

!pip install lightfm

!pip install PyDrive
!pip install msgpack
```

```
Requirement already satisfied: scikit-surprise in /usr/local/lib/python3.6/dist-packages (1.0.6)
Requirement already satisfied: scipy>=1.0.0 in /usr/local/lib/python3.6/dist-packages (from scikit-surprise) (1.3.0)
Requirement already satisfied: numpy>=1.11.2 in /usr/local/lib/python3.6/dist-packages (from scikit-surprise) (1.16.4)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist-packages (from scikit-surprise) (0.13.2)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from scikit-surprise) (1.12.0)
Requirement already satisfied: lightfm in /usr/local/lib/python3.6/dist-packages (1.15)
Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from lightfm) (1.16.4)
Requirement already satisfied: scipy>=0.17.0 in /usr/local/lib/python3.6/dist-packages (from lightfm) (1.3.0)
Requirement already satisfied: requests in /usr/local/lib/python3.6/dist-packages (from lightfm) (2.21.0)
Requirement already satisfied: chardet<3.1.0,>=3.0.2 in /usr/local/lib/python3.6/dist-packages (from requests->lightfm) (3.0.4)
Requirement already satisfied: idna<2.9,>=2.5 in /usr/local/lib/python3.6/dist-packages (from requests->lightfm) (2.8)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.6/dist-packages (from requests->lightfm) (2019.6.1
6)
Requirement already satisfied: urllib3<1.25,>=1.21.1 in /usr/local/lib/python3.6/dist-packages (from requests->lightfm) (1.24.
3)
Requirement already satisfied: PyDrive in /usr/local/lib/python3.6/dist-packages (1.3.1)
Requirement already satisfied: google-api-python-client>=1.2 in /usr/local/lib/python3.6/dist-packages (from PyDrive) (1.7.9)
Requirement already satisfied: PyYAML>=3.0 in /usr/local/lib/python3.6/dist-packages (from PyDrive) (3.13)
Requirement already satisfied: oauth2client>=4.0.0 in /usr/local/lib/python3.6/dist-packages (from PyDrive) (4.1.3)
Requirement already satisfied: six<2dev,>=1.6.1 in /usr/local/lib/python3.6/dist-packages (from google-api-python-client>=1.2->
PyDrive) (1.12.0)
Requirement already satisfied: google-auth>=1.4.1 in /usr/local/lib/python3.6/dist-packages (from google-api-python-client>=1.2
->PyDrive) (1.4.2)
Requirement already satisfied: uritemplate<4dev,>=3.0.0 in /usr/local/lib/python3.6/dist-packages (from google-api-python-clien
t>=1.2->PyDrive) (3.0.0)
Requirement already satisfied: google-auth-httplib2>=0.0.3 in /usr/local/lib/python3.6/dist-packages (from google-api-python-cl
ient>=1.2->PyDrive) (0.0.3)
Requirement already satisfied: httplib2<1dev,>=0.9.2 in /usr/local/lib/python3.6/dist-packages (from google-api-python-client>=
1.2->PyDrive) (0.11.3)
Requirement already satisfied: pyasn1-modules>=0.0.5 in /usr/local/lib/python3.6/dist-packages (from oauth2client>=4.0.0->PyDri
ve) (0.2.5)
Requirement already satisfied: pyasn1>=0.1.7 in /usr/local/lib/python3.6/dist-packages (from oauth2client>=4.0.0->PyDrive) (0.
4.5)
Requirement already satisfied: rsa>=3.1.4 in /usr/local/lib/python3.6/dist-packages (from oauth2client>=4.0.0->PyDrive) (4.0)
Requirement already satisfied: cachetools>=2.0.0 in /usr/local/lib/python3.6/dist-packages (from google-auth>=1.4.1->google-api
-python-client>=1.2->PyDrive) (3.1.1)
Requirement already satisfied: msgpack in /usr/local/lib/python3.6/dist-packages (0.5.6)
```

In [0]:
```
import os
os.environ["JAVA_HOME"] = "/usr/lib/jvm/java-8-openjdk-amd64"
os.environ["SPARK_HOME"] = "/content/spark-2.4.2-bin-hadoop2.7"
```

In [0]:
```
import pandas as pd
import numpy as np
import math
import matplotlib.pyplot as plt
import urllib.request
import zipfile

from collections import defaultdict

import msgpack

import os
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials

from sklearn.preprocessing import LabelEncoder
from sklearn.model_selection import train_test_split as sk_tts

import findspark
findspark.init()
from pyspark.sql import SparkSession
from pyspark.sql.functions import isnan
from pyspark.sql import functions as F
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, IndexToString
from pyspark.ml.evaluation import RegressionEvaluator
from pyspark import SparkContext
from pyspark.sql.functions import lit
from pyspark.ml.recommendation import ALS

from surprise import Reader, Dataset
from surprise import SVD, SVDpp, NMF, KNNBasic, KNNWithMeans, KNNWithZScore, BaselineOnly
from surprise import accuracy
from surprise.model_selection import train_test_split as surprise_tts
from surprise.model_selection import GridSearchCV

from lightfm import LightFM
from lightfm.evaluation import precision_at_k
from lightfm.evaluation import auc_score
from scipy.sparse.coo import coo_matrix

pd.set_option('display.max_rows', 500)
pd.set_option('display.max_columns', 500)
pd.set_option('display.width', 1000)
```

Datasets download

In [0]:
```
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)
```

In [0]:
```
file_list = drive.ListFile({'q': "'root' in parents and trashed=false"}).GetList()
for file1 in file_list:
  print('title: %s, id: %s' % (file1['title'], file1['id']))
```

```
title: LU08.ipynb, id: 1DQoWWBwFeaXI6CumEPKT9CpLQ7pCdLIz
title: SPAAI, id: 15xHXty3TvHwIS6hSAgt1lq-fXg6H-XaO
title: BookCrossing, id: 1UBntfrt6niZGad3p_BbysJGpmKVg7fkq
title: Yelp2, id: 1t7_Z9fEp207WlAPIcw8iLChcW71DtINT
title: Colab Notebooks, id: 1D1Kjnr4ZJkGlIzTqHV3A7swMZCZdUvn8
title: Yelp, id: 1G0-MKdJMIzHCcEnP_8_6Gk3aQKyl_435
title: DLSD_Project, id: 1_aiIGyPkP5ojOrZLhMWb2eP-J_ayI_dB
title: Getting started, id: 0BzLIx_JXOsMsc3RhcnRlcl9maWxl
```

In [0]:
```
file_list = drive.ListFile({'q': "'1G0-MKdJMIzHCcEnP_8_6Gk3aQKyl_435' in parents and trashed=false"}).GetList()
for file1 in file_list:
  print('title: %s, id: %s' % (file1['title'], file1['id']))
```

```
title: user_mapping.pickle, id: 1ZVPjsW6wK0qXg3Wh5e2SnQYstk6tFSfp
title: business_mapping.pickle, id: 1Oj46ti8KLY4QFH3x8_PORvwOmsXBRPME
title: review_enc_30K.csv, id: 1mhSipRAxg3_Nt2bDBqUB1irHbdiqXG9F
title: review_enc_20K.csv, id: 1mTdoiOiV4zr8M7PDnOzW42JHtvPy5klF
title: review_enc_10K.csv, id: 1XfKApQYeMl4Gl1OOQ-8qfyaC2L9PyXnQ
title: review_enc_3M.csv, id: 1_YubYVgKsbpAVhsVEfyORzGWSf_th0EC
title: review_enc_1M.csv, id: 1JSzg4xaooCd14JzOWuwybSDXl8fqsRct
title: cftest_enc.csv, id: 1aGVcnTfEjdSwIH9ZU5MBveiSjKwrn2BK
title: review_enc_100K.csv, id: 17snwxn2nlMba2P6MGoQg5my5yIw9WB-B
title: user_enc.csv, id: 1nEtiOwwp_ym08duyybaDrEK9DM4SEem1
title: business_enc.csv, id: 1BAHxH3p5NI6PWqI8NGI_GieXfM5fc02F
title: review_enc.csv, id: 112uBfXYPtgkS3dIIOkVwxWGj_m6V-UZL
title: review_50000.csv, id: 11k1Xl9ijq2qqM-xP-y1qvC-fVMOslZF9
title: review_15000.csv, id: 1j_urXqXqw2FA_WlPx-6P55-zlufmzEjg
title: review_10000.csv, id: 1g6h0a0LMV-Jkc93z7YnqMF6Id8CShnyG
title: review_25000.csv, id: 1TnUcOitaUX_fn8-N0oW8l6L3Wdtb3Kh9
title: review_11000.csv, id: 1GM6Fk-tlN8DnDRN8kfaMXx3brE8cgfVx
title: review_1000.csv, id: 1tkyp6kkVh_INy7KDsEDeHT2jhilaGrh3
title: review_100000.csv, id: 1_geILDVSLU1s7b4GHX1hto7A79P-W5GL
title: business.csv, id: 1l3QkzlE-EwBX8UE288ALRcy3ceJnUkyE
```

In [0]:
```
for file1 in file_list:
  if('_enc') in file1['title']:
    id = file1['id']
    fileId = drive.CreateFile({'id': id})

    filename = file1['title']
    print(filename)
    fileId.GetContentFile(filename)  # Save Drive file as a local file
```

```
review_enc_30K.csv
review_enc_20K.csv
review_enc_10K.csv
review_enc_3M.csv
review_enc_1M.csv
cftest_enc.csv
review_enc_100K.csv
user_enc.csv
business_enc.csv
review_enc.csv
```

Loading and parsing datasets

In [0]:
```
ratings_raw_data = pd.read_csv('review_enc_10K.csv')
ratings_raw_data_header = ratings_raw_data.columns

ratings_data = ratings_raw_data[['user_id', 'business_id', 'stars']]
ratings_data.columns = ['user_id', 'business_id', 'stars']

ratings_data.head(3)
```

Out[0]:

	user_id	business_id	stars
0	1158189	176750	1.0
1	1599679	74082	5.0
2	1307908	100927	5.0

In [0]:
```
ratings_data.shape
```

Out[0]: (10000, 3)

In [0]:
```
business_raw_data = pd.read_csv('business_enc.csv', encoding = "latin-1")
business_raw_data_header = business_raw_data.columns

business_data = business_raw_data.replace(to_replace='None', value=np.nan)

business_raw_data.head()
```

```
/usr/local/lib/python3.6/dist-packages/IPython/core/interactiveshell.py:2718: DtypeWarning: Columns (6,28,31) have mixed types.
Specify dtype option on import or set low_memory=False.
  interactivity=interactivity, compiler=compiler, result=result)
```

Out[0]:

	address	attributes	attributes_AcceptsInsurance	attributes_AgesAllowed	attributes_Alcohol	attributes_Ambience	attributes_BYOB	attribu
0	2818 E Camino Acequia Drive	NaN	NaN	NaN	NaN	NaN	NaN	NaN
1	30 Eglinton Avenue W	NaN	NaN	NaN	u'full_bar'	{'romantic': False, 'intimate': False, 'classy...	NaN	NaN
2	10110 Johnston Rd, Ste 15	NaN	NaN	NaN	u'beer_and_wine'	{'romantic': False, 'intimate': False, 'touris...	NaN	NaN
3	15655 W Roosevelt St, Ste 237	NaN	NaN	NaN	NaN	NaN	NaN	NaN
4	4209 Stuart Andrew Blvd, Ste F	NaN	NaN	NaN	NaN	NaN	NaN	NaN

In [0]:
```
business_data.shape
```
Out[0]:
```
(192609, 60)
```

In [0]:
```
user_raw_data = pd.read_csv('user_enc.csv')
user_raw_data_header = user_raw_data.columns

user_data = user_raw_data[['user_id', 'name']]

user_raw_data.head()
```

Out[0]:

	user_id	name	review_count	yelping_since	useful	fans	average_stars
0	1256813	Rashmi	95	2013-10-08 23:11:33	84	5	4.03
1	141378	Jenna	33	2013-02-21 22:29:06	48	4	3.63
2	1013852	David	16	2013-10-04 00:16:10	28	0	3.71
3	1054713	Angela	17	2014-05-22 15:57:30	30	5	4.85
4	596783	Nancy	361	2013-10-23 07:02:50	1114	39	4.08

In [0]:
```
user_data.shape
```
Out[0]:
```
(1637138, 2)
```

Data Cleaning

In [0]:
```
ratings_data_nodup = ratings_data.drop_duplicates()

print("Before:", ratings_data.shape)
print("After:", ratings_data_nodup.shape)
```
```
Before: (10000, 3)
After: (9991, 3)
```

In [0]:
```
idx_dup = np.where(ratings_data_nodup.index.duplicated())
print(idx_dup)
```
```
(array([], dtype=int64),)
```

In [0]:
```
#Filter records for users who have rated at least 5 businesses
user_rating_counts = ratings_data_nodup['user_id'].value_counts()
print(user_rating_counts.shape[0])
```
```
9366
```

In [0]:
```
#Filter records for businesses who have rated at least 5 ratings
business_rating_counts = ratings_data_nodup['business_id'].value_counts()
print(business_rating_counts.shape[0])
```
```
4618
```

In [0]:
```
rating_counts = ratings_data_nodup['stars'].value_counts()
rating_counts
```
Out[0]:
```
5.0    4418
4.0    2177
1.0    1527
3.0    1069
2.0     800
Name: stars, dtype: int64
```

Dataset Presentation

In [0]:
```
ratings_dict = {'itemID': list(ratings_data_nodup.business_id),
                'userID': list(ratings_data_nodup.user_id),
                'rating': list(ratings_data_nodup.stars)}
df = pd.DataFrame(ratings_dict)
```

Scikit-SurpriseLib

Train/Test Split

In [0]:
```
reader = Reader(rating_scale=(0.5, 5.0))

data = Dataset.load_from_df(df[['userID', 'itemID', 'rating']], reader)

trainset = data.build_full_trainset()
```

In [0]:
```
print("Number of Businesses:", trainset.n_items)
print("Number of Users:", trainset.n_users)
```

```
Number of Businesses: 4618
Number of Users: 9366
```

In [0]:
```
training, test = surprise_tts(data, test_size=.2)
test_for_predict = test
```

Collaborative Filtering

In [0]:
```
for Algo in ["KNNBasic", 'KNNWithMeans', 'KNNWithZScore']:

  for sim_metric in [ 'cosine', 'pearson', 'msd', 'pearson_baseline']:

    sim_options = {'name': sim_metric,
                   'user_based': True  # compute  similarities between users
                   }

    if(Algo == "KNNBasic"):
      model_user = KNNBasic(sim_options=sim_options, verbose=False)
    elif(Algo == "KNNWithMeans"):
      model_user = KNNWithMeans(sim_options=sim_options, verbose=False)
    elif(Algo == "KNNWithZScore"):
      model_user = KNNWithZScore(sim_options=sim_options, verbose=False)

    model_user.fit(training)

    predictions_user = model_user.test(test_for_predict)

    # Then compute RMSE
    error_user = accuracy.rmse(predictions_user)

    # best RMSE score
    print('Algo: {}, similarity metric: {}, RMSE: {}'.format(Algo, sim_metric, error_user))
```

```
RMSE: 1.4733
Algo: KNNBasic, similarity metric: cosine, RMSE: 1.473314935577257
RMSE: 1.4739
Algo: KNNBasic, similarity metric: pearson, RMSE: 1.4738796557934557
RMSE: 1.4733
Algo: KNNBasic, similarity metric: msd, RMSE: 1.473314935577257
RMSE: 1.4739
Algo: KNNBasic, similarity metric: pearson_baseline, RMSE: 1.4738796557934557
RMSE: 1.4932
Algo: KNNWithMeans, similarity metric: cosine, RMSE: 1.4932031581304874
RMSE: 1.4936
Algo: KNNWithMeans, similarity metric: pearson, RMSE: 1.4935800059696343
RMSE: 1.4932
Algo: KNNWithMeans, similarity metric: msd, RMSE: 1.4932031581304874
RMSE: 1.4936
Algo: KNNWithMeans, similarity metric: pearson_baseline, RMSE: 1.4935800059696343
RMSE: 1.4931
Algo: KNNWithZScore, similarity metric: cosine, RMSE: 1.4930825798422611
RMSE: 1.4936
Algo: KNNWithZScore, similarity metric: pearson, RMSE: 1.4935800059696343
RMSE: 1.4931
Algo: KNNWithZScore, similarity metric: msd, RMSE: 1.4930825798422611
RMSE: 1.4936
Algo: KNNWithZScore, similarity metric: pearson_baseline, RMSE: 1.4935800059696343
```

Matrix Factorization

In [0]:
```
for Algo in ["SVD", 'SVDpp', 'NMF']:

    if(Algo == "SVD"):
      model_MF = SVD()
    elif(Algo == "SVDpp"):
      model_MF = SVDpp()
    elif(Algo == "NMF"):
      model_MF = NMF()
    elif(Algo == "ALS"):
      bsl_options = {'method': 'als',
                     'n_epochs': 5,
                     'reg_u': 12,
                     'reg_i': 5 }
      model_MF = BaselineOnly(bsl_options=bsl_options)

    model = model_MF
    model.fit(training)
    predictions = model.test(test_for_predict)

    # Then compute RMSE
    error = accuracy.rmse(predictions)

    # best RMSE score
    print('Algo: {}, RMSE: {}'.format(Algo, error))
```

```
RMSE: 1.4300
Algo: SVD, RMSE: 1.430005490798664
RMSE: 1.4251
Algo: SVDpp, RMSE: 1.4251121726087237
RMSE: 1.4995
Algo: NMF, RMSE: 1.4995359743418488
```

In [0]:
```
param_grid = {'n_epochs': [10, 20, 40], 'lr_all': [0.002, 0.005, 0.01],
              'reg_all': [0.1, 0.2, 0.4]}

model = SVDpp
gs = GridSearchCV(model, param_grid, measures=['rmse'], cv=3)

gs.fit(data)

print('The best model was trained with %s' % gs.best_params['rmse'])

model = gs.best_estimator['rmse']
model.fit(training)
predictions = model.test(test_for_predict)

# Then compute RMSE
error = accuracy.rmse(predictions)

# best RMSE score
print('For testing data the RMSE is %s' % (error))
```

```
The best model was trained with {'n_epochs': 40, 'lr_all': 0.01, 'reg_all': 0.2}
RMSE: 1.4182
For testing data the RMSE is 1.4181911761433839
```

Spark ML

Train/Test split

In [0]:
```
from pyspark.sql import SparkSession

spark = SparkSession \
     .builder \
     .master("local[*]") \
     .config("spark.executor.memory", "4g") \
     .config("spark.driver.memory", "4g") \
     .config("spark.memory.offHeap.enabled",True) \
     .config("spark.memory.offHeap.size","4g") \
     .appName("rec") \
     .getOrCreate()
```

In [0]:
```
ratings_raw_data = spark.read.csv('review_enc_10K.csv',inferSchema=True,header=True)
ratings_raw_data_header = ratings_raw_data.columns

ratings_data = ratings_raw_data[['user_id', 'business_id', 'stars']]
```

In [0]:
```
training, validation, test = ratings_data.randomSplit([0.6, 0.2, 0.2], seed=0)
```

Matrix Factorization

In [0]:
```
seed = 5
iterations = 25
regularization_parameter = 5.0
ranks = [4, 8, 10, 12]
errors = [0, 0, 0, 0]
err = 0
tolerance = 0.02

min_error = float('inf')
best_rank = -1
best_iteration = -1
for rank in ranks:
    als = ALS(maxIter=iterations, rank=rank, regParam=regularization_parameter, seed=seed, userCol="user_id",
          itemCol="business_id", ratingCol="stars", coldStartStrategy="drop")
    model = als.fit(training)
    predictions = model.transform(validation)
    evaluator = RegressionEvaluator(metricName="rmse",
                                labelCol="stars",
                                predictionCol="prediction")
    error = evaluator.evaluate(predictions)
    errors[err] = error
    err += 1
    print('For rank %s the RMSE is %s' % (rank, error))
    if error < min_error:
        min_error = error
        best_rank = rank

print('The best model was trained with rank %s' % best_rank)
```

```
For rank 4 the RMSE is 4.21255973560481
For rank 8 the RMSE is 4.212133766200419
For rank 10 the RMSE is 4.211716778894937
For rank 12 the RMSE is 4.2117295975393905
The best model was trained with rank 10
```

In [0]:
```
predictions.head(3)
```

Out[0]:
```
[Row(user_id=48898, business_id=173382, stars=5.0, prediction=7.0597027572796840e-13),
 Row(user_id=1352785, business_id=186574, stars=4.0, prediction=0.0003806327877100557),
 Row(user_id=1354006, business_id=3834, stars=5.0, prediction=-3.2748345111270975e-41)]
```

In [0]:
```
als = ALS(maxIter=iterations, rank=best_rank, regParam=regularization_parameter, seed=seed, userCol="user_id",
          itemCol="business_id", ratingCol="stars", coldStartStrategy="drop")
model = als.fit(training)
predictions = model.transform(test)
evaluator = RegressionEvaluator(metricName="rmse",
                                labelCol="stars",
                                predictionCol="prediction")
error = evaluator.evaluate(predictions)

print('For testing data the RMSE is %s' % (error))
```

```
For testing data the RMSE is 3.932013566624354
```

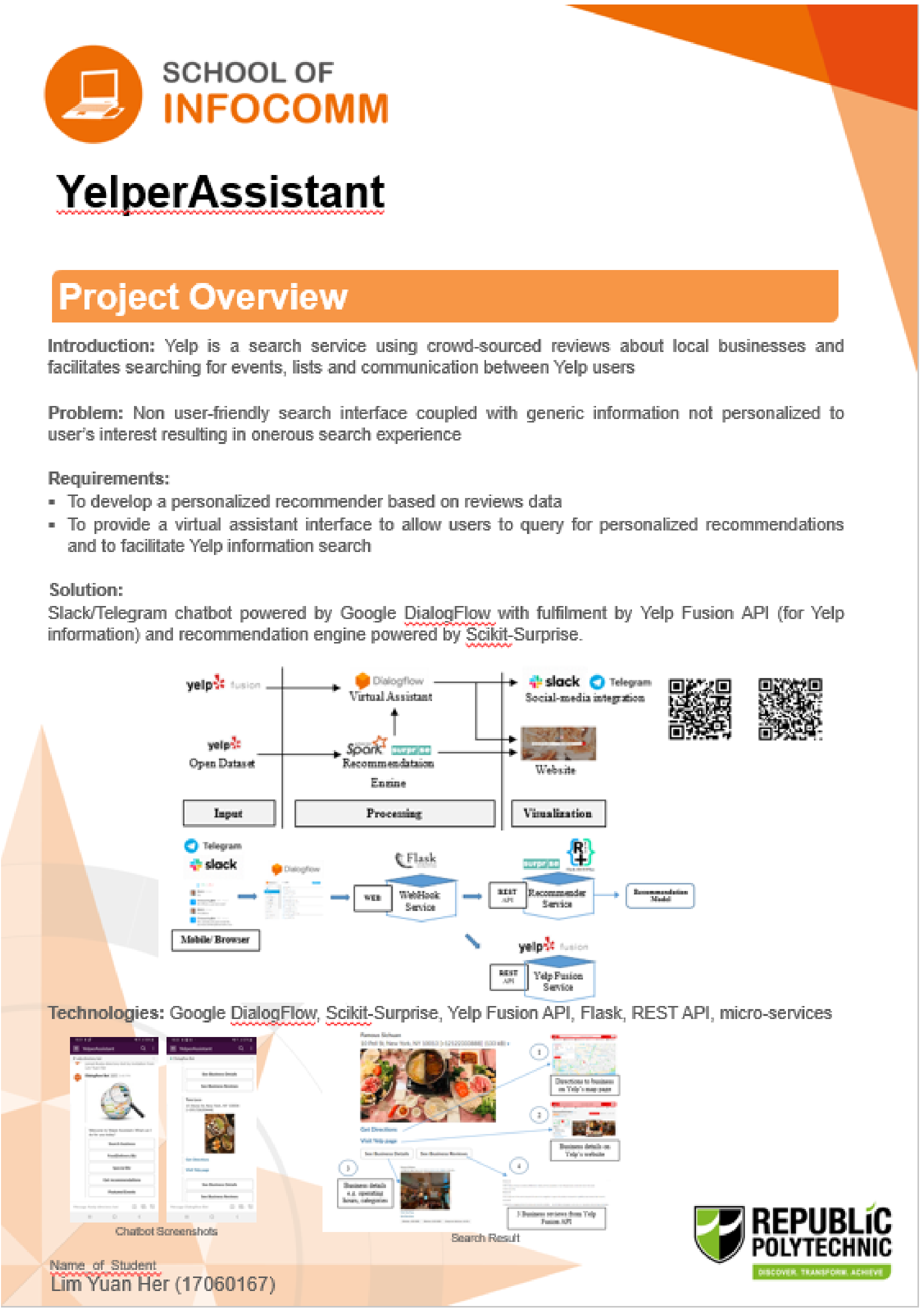
Recommendation

In [0]:
```
testSubject = 956232

print("\nBuilding recommendation model...")

trainSet = data.build_full_trainset()

model = SVDpp(n_epochs=40, lr_all=0.01, reg_all=0.2)

model.fit(trainSet)

print("Computing recommendations...")
trainset = data.build_full_trainset()
fill = trainset.global_mean
anti_testset = []
u = trainset.to_inner_uid(testSubject)
user_items = set([j for (j, _) in trainset.ur[u]])

anti_testset += [(trainset.to_raw_uid(u), trainset.to_raw_iid(i), fill) for
                                 i in trainset.all_items() if
                                 i not in user_items]

testSet = anti_testset

predictions = model.test(testSet)

recommendations = []

print ("\nWe recommend:")
for user_id, business_id, stars, estimatedRating, _ in predictions:
  recommendations.append((business_id, estimatedRating))

recommendations.sort(key=lambda x: x[1], reverse=True)

for ratings in recommendations[:10]:
  print(business_data[business_data.business_id == ratings[0]]["name"].values[0], ratings[1])
```

```
Building recommendation model...
Computing recommendations...

We recommend:
Animal Kindness Veterinary Hospital 4.542446337259187
Gio Rana's Really Really Nice Restaurant 4.528699723489563
Biaggio's Pizzeria 4.51662373084823
Nikki's Akropolis Pizza 4.481412511521263
Clear View Home Inspections 4.459661964887028
Fountains of Bellagio 4.4584252409045915
PokÃ© Catcher 4.454446491709405
Las Enchiladas Demama 4.454090502414389
Gun Garage 4.445375169969016
Pink Cherry Wax 4.4436391083879005
```

10 Project Poster

10.1 Poster



10.2 Source File

C3879C_Capstone_Project_Poster_1706016