Your Inner Foodie

Implementation Project

Project Description:

An algorithm that outputs the shortest path (MST via Prim's or such similar algorithms) to all the top 10 (or less if no sufficient data is found) most popular and highly rated restaurants, based on the number of tweets and sentiment in the tweets. Popularity would be measured by the number of tweets that mention that restaurant and rating would be measured by performing a rule based sentiment analysis. The outputs will be in console format, listing the shortest route to visit all restaurants sequentially.

Please note that Twitter's API has an upper limit of accessing 450 tweets/15 mins. In other words, the program will only run well if used *once* every 15 minutes. This is because we have the free "non-elevated" access to Twitter's API. This issue would be resolved if we had keys that had more elevated privileges.

Important: It is likely that Maven and the Twitter library have not been imported properly. In this case, please import the following dependency into `pom.xml` as described in this link: https://www.jetbrains.com/help/idea/work-with-maven-dependencies.html#generate_maven_dependency (*please use IntelliJ for ease of use)* :

```
<dependency>
     <groupId>com.twitter</groupId>
     <artifactId>twitter-api-java-sdk</artifactId>
     <version>1.1.4</version>
 </dependency>
```

Implemented Categories:

1) Graph and Graph Algorithms

Given a weighted, undirected graph that represents the distances between top trending restaurants, we used Prim's Minimum Spanning Tree Algorithm to determine the most efficient way to visit all parts of the restaurants curated on the bucket list.

2) Information Networks (Twitter and Google Maps)

- *Google Maps*
  We use the following APIs:

  Geocoding (Base: https://maps.googleapis.com/maps/api/geocode/json?) - To get information about the city (ex. City bounds, location, place id) and to check if the inputted city is valid. We do this by calling the api url and iterating over all address components to check if any of the components have locality type, which means that it's a valid city.

Places (Base: https://maps.googleapis.com/maps/api/place/nearbysearch/json?) - To get all restaurants in a city. We do this by iterating over all pages after fetching results from the api url and get each restaurant name.

Distance (Base: https://maps.googleapis.com/maps/api/distancematrix/json?) - To get distance from one restaurant to another. This is done through storing the location of the restaurant names in a HashMap and then calling the api url with the correct destination and origin locations. We then find the distance JSONObject to get the distance between the restaurants.

- *Twitter*

  Recent tweets (Base: https://developer.twitter.com/en/docs/twitter-api/tweets/search/api-reference/get-tweets-search-recent) We search tweets that contain at least one of the restaurant names compiled from Google Maps. We also request the tweet's author's user ID.

  User lookup (Base: https://developer.twitter.com/en/docs/twitter-api/v1/accounts-and-users/follow-search-get-users/api-reference/get-users-lookup) Because tweets themselves are rarely geotagged, we also lookup the user's general location. We assume that tweets were made in the same location as listed in the user's profile.

3) Document Search / Advanced Topics:

We used NLP to perform sentiment analysis on the tweets. We used a rule-based model to compare the frequency of positive and negative words (after parsing and stemming the words) on each of the tweets and give a score based on that. Each restaurant would have a score based on the average sentiment on its tweets. Positive and Negative words were determined based on external lists of negative and positive documents.

Work Breakdown:

Task Chart	
Task	People
[Twitter] Scraping tweets from the city	Ethan
[Twitter] finding most popular (most tweeted) restaurants	
[Twitter] Analyzing the sentiments of the tweets	Mohamed
[Twitter] Suggesting 10 restaurants based on sentiment and popularity.	

[Google] Checking if city is a valid input & finding all restaurants in the city for Twitter to check against	Andy
[Google] Scraping distance information about the restaurants	
Implementing an MST algorithm and using it to generate the shortest path to the top 10 restaurants.	Jeff
Console input/output	Mohamed & Andy

Task Chart			
Task	People	Input Variables	Output Variables
[Google Maps] Checking if city is a valid input	Andy	- String cityName	- Double[][] geolocation - Element at index 0 is one corner of the city, index 1 is the other corner - Assume all cities have a rectangular region - Edit: return boolean -
[Google] Finding all restaurants in the city for Twitter to check against	Andy	- String cityName	- String[] restaurantNames
[Twitter] Scraping tweets from the city that also contain the restaurant name	Ethan	- Double[][] geolocation - Element at index 0 is one corner of the city, index 1 is the other corner - String[] restaurantNames	- String[] tweetedContent
[Twitter] finding most popular (most tweeted) restaurants	Ethan	- String[] tweetedContent - String[] restaurantNames	- Map<String, List<String>> Restaurant to all tweets that mention the restaurant
[Twitter] Analyzing the sentiments of the tweets	Mohamed	- Map<String, List<String>> Restaurant to all	- Map<String, Integer> Restaurant to score

		tweets that mention the restaurant	
[Twitter] Suggesting 10 restaurants based on sentiment and popularity.	Mohamed	- Map<String, Integer> Restaurant to score	- String[] restaurantNames
[Google] Scraping distance information about the restaurants	Andy	- String[] restaurantNames	- Adjacency Matrix - Weighted edges (distance) - Index 0 corresponds to #1 restaurant
Implementing an MST algorithm and using it to generate the shortest path to the top 10 restaurants.	Jeff	- Adjacency Matrix - Weighted edges (distance)	- String[] restaurantsToTour - In order of which to visit, first to last
Console input/output	Jeff	- String cityName	- String[] restaurantsToTour - In order of which to visit, first to last