# CMPT 353 Project: OSM, Photos, and Tours

Reece McGowan, Trung Hieu Le, Yangyang Liu

# Introduction

Our group decided to tackle the project topic related to the OpenStreetMap data set. OpenStreetMap [1] is a project intended to create a free database of the world’s geography. For this topic, we are interested in finding amenities, that is, a desirable or useful feature in the Vancouver area. Our group aimed to solve the following problems. Given a collection of geotagged photos, can we create a model that attempts to find the most interesting thing in the photograph? If someone were planning a tour of the city, where should they go? Are there paths that are more interesting than others? If someone were to choose a hotel to stay in, where should it be?

The planet.osm dataset is large, with the most recent upload being 110GB. Provided to us to use was code which takes the entire dataset and transforms it into Spark-friendly JSON entries (.\provided\disassemble-osm.py), gets only the entries defined as an amenity (.\provided\osm-amenities.py), and then extracts the amenities that are in the Greater Vancouver area (.\provided\just-vancouver.py). This provided a much more manageable dataset.

# First problem: Given a collection of geotagged photos, tell the user the things they should have seen. Reece McGowan

## Initial data filtering

The OSM data includes GPS coordinates, so it seems trivial to tell the user what they should have seen. We can simply show them a list of all the things near them in the picture. The problem is the dataset is filled with things that we aren’t interested in. For example, many of the amenities in the dataset are infrastructure or utility amenities. So, some entries that are obviously not interesting to the user need to be dropped manually. To do this, the file get\_unique\_amenities.py was created. This script takes as input a list of amenities and exports the unique amenity types as CSV. In our Vancouver subset, there are 138 unique amenities. I then viewed the list of amenities and started partitioning the list into two files, vancouver-amenities-to-drop.csv and vancouver-interesting-amenities.csv. At first, I simply took the name of the amenity at face value and decided whether to keep it. But I soon realized the amenity name was deceptive. For example, the Gastown Steam Clock, a notable attraction for anyone in Gastown, appears as the amenity type “clock.” Without checking the entries for clock, one might assume this entry is purely infrastructure. So, for each amenity I wanted to drop, I manually searched the original dataset to look at the relevant entries and decide if it is in fact boring. This process produced the vancouver-amenities-to-drop.csv.

This CSV is then passed as an argument to load\_extracted\_amenities.py. This file takes the entire amenities dataset and then drops any rows that have an amenity in the CSV of amenities to drop. This process takes our Vancouver subset of 17,718 entries and reduces it down to 4673 entries. This is still a lot but much more manageable.

## Applying heuristics

The next part of the problem is to apply some heuristics to find which entries are the most “interesting.” Each entry contains a dictionary of tags. Many entries contain a tag which specifies a Wikidata ID. Wikidata is a user-contributed public repository containing information about anything. I thought relative popularity compared with other Wikidata entries would be a good measure of how interesting something is. My initial idea was to use page views for each entry but that turned out to be non-trivial, so instead I went with the total number of sitelinks in each page. Wikidata sitelinks are links that go from individual Wikidata pages to the entry’s equivalent entry in other wiki sites such as Wikipedia, Wikisource, and Wikivoyage [2]. The total number of these sitelinks can be used to measure popularity across various Wikis. For example, the Wikidata page for Earth [3] has 307 sitelinks, while the Wikidata page for Starbucks [4] has 78 sitelinks.

So, I wrote a file called append\_wikidata\_info.py which depends on a class I created called WikidataAPI. This class queries the Wikidata API with the Wikidata ID provided in the tags of the amenity and returns the number of sitelinks for that entry. This data is then appended to the data frame. After this is done, I also added a column which counts the number of times that Wikidata ID appears in the Vancouver dataset. The purpose of this is to identify chains. If a Wikidata ID appears with 200 amenities, it probably isn’t that interesting. A good example of this is Starbucks. It has a high number of sitelinks but isn’t that interesting. We can identify it is a chain by looking at its number of Wikidata occurrences and scaling its interest accordingly.

Now we have all the necessary information to determine interesting amenities in one data frame. So now it’s time to assign each amenity an “interest score.” My basic strategy was to assign each entry a number. There are certain properties I think signal interest for an amenity, which means the score should be increased, and some properties I think signal disinterest for an amenity, which means the score should be decreased. So, I came up with the following formula which seems to work well.

score = score\_multiplier \* (min\_score + wikidata\_sitelinks\_boost + tags\_length\_boost + tags\_entry\_boost)

The components of this formula are derived as follows.

### score\_multiplier

This value attempts to boost or suppress results based on the type of amenity. There are a lot of restaurants and cafes in the Vancouver subset. I personally wouldn’t consider any specific one of these tourist attractions, so they are suppressed with the value set to 0.5. If the amenity is something else, then it is boosted by setting this value to 2.

### min\_score

By default, this value is 0. If the entry has a Wikidata entry, we want to give it a boost. Entries that someone has made the effort to create a Wikidata entry for are likely more interesting than an entry without. If that Wikidata ID appears only once in our dataset, min\_score is set to 4. If the Wikidata ID appears more than once, this is likely part of a chain of identical amenities and so isn’t as interesting and we set min\_score to 2.

### wikidata\_sitelinks\_boost

This value adds a boost based on the number of sitelinks in an entry with a Wikidata ID to boost based on popularity. First, I computed a relative sitelinks score for each entry in the Vancouver dataset. This was done by measuring the average number of sitelinks for all entries with a Wikidata ID. The relative sitelinks score for a particular entry is its own sitelinks count divided by the average sitelinks count for all entries. The wikidata\_sitelinks\_boost is then computed by normalizing this value against the difference between the maximum and minimum sitelinks score for the entire dataset.

wikidata\_sitelinks\_boost = sitelinks\_score / (max\_sitelinks\_score – min\_sitelinks\_score)

### tags\_length\_boost

This is a boost based on the length of the tags dictionary of the entry. This value is computed in similarly to the wikidata\_sitelinks\_boost.

### tags\_entry\_boost

This is a value intended to boost amenities with entries in the tags dictionary which I think indicate a more interesting amenity.

Once the above values have been calculated, the final score is added as a column to the data frame and exported.

## Analyze photos

The final data frame is then taken as input to interesting\_things\_in\_photos.py. This file allows the user to select a collection of photos in Windows File explorer. The model will then get the coordinates from the file using a class I created called ImageParser. Much of the code for this was adapted from a Medium article [5]. Essentially, the model looks for all amenities near the coordinates of the picture taken from the pipelined dataset. The model then sorts by the calculated interest score and saves the top hit. This data is then exported for any number of pictures to a CSV for reference.

## Sample run

We are going to run the model on the following pictures



The latitude and longitude in the metadata of this picture is 49; 17; 2.6520 and 123;6;25.9070 in the DMS coordinates system which is 49.28407 and -123.1072, respectively. Obviously, we expect the model to guess that the user should have seen the Gastown Steam Clock while they were in the area.

A picture containing text, building, outdoor, road

Description automatically generated

We will also pass this photo of the Commodore Ballroom to the model in the same run. The GPS coordinates for this photo were injected manually, as I have found it difficult to find photos online which are geotagged. The coordinates of the image then are 49; 16; 50.4243 and 123; 7; 15.0611 in the DMS coordinates system which is 49.2807 and -123.1209, respectively. We would expect the model to guess that the user should have seen the Commodore Ballroom while they were in the area.

## Results

Running the model with these two images as input provides the following CSV, which is what we expected. This file is included as nearby\_interesting.csv.

Table

Description automatically generated

## Limitations/Retrospective

One problem I encountered was gathering test images to test the model. Ideally, I would have liked hundreds of images of interesting things in Vancouver that are geotagged. Then I would see how many of these the model could correctly identify to give an idea of how good the model is. I unfortunately found it difficult to find images that are already geotagged and taking hundreds of photos around Vancouver wasn’t feasible.

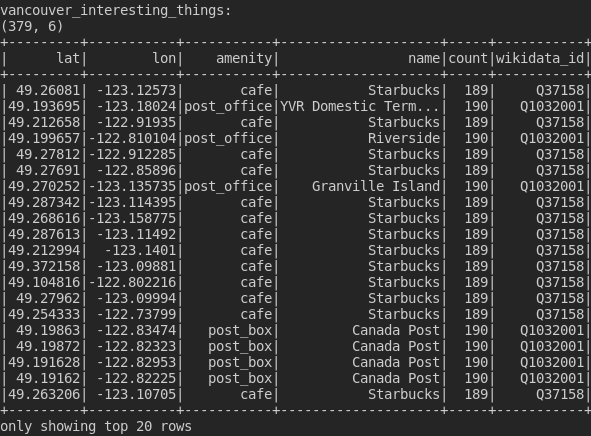
I also felt like the OSM data was missing a lot of key amenities. For example, some interesting things that I know of in Vancouver are the Vancouver Aquarium and the Capilano Suspension Bridge. These are certainly things people take pictures of or in. Unfortunately, neither of these amenities are found in the Vancouver dataset, and there are many more examples. These entities do appear in the OpenStreetMap database, but clearly, they are not tagged as amenities. So given more time I would have come up with a different way of parsing the full OSM data dump for things that could potentially be interesting so entities such as these are included.

# Second problem: Provide paths to take the user to a variety of interesting places. Trung Hieu Le

## Initial data filtering

The initial given dataset contains information about latitude, longitude, type of amenity, name, timestamp in which the data was recorded, and the tags of the location. The provided data is relatively raw so the first thing that needs to be done is exclude any data entries that do not have the **Wikipedia** tag. This includes the process of removing any tuples that contain null as its value on either amenity or name column. By doing so, the monstrous size of our dataset reduces drastically from 17718 data points to only 1639.

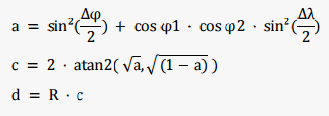
To filter out which location is deemed as “interesting”, I added a column which counts the number of times that Wikidata ID appears in the dataset. The threshold is set at 150, meaning that a random place with a number of Wikidata ID occurrences above 150 can be viewed as “interesting”. This helps further reduce the size of our dataset to just only 379 data points.



# Applying heuristics

After having our dataset cleaned and filtered out the only data that we need, the next step is to find a recommended sequence of locations for a user to visit. Due to the fact that all the data we have at this point are considered to be “interesting”, it would be unjustified to just prefer a location over another due to its slightly lower or higher interest score; hence, I believe that grouping nearby “interesting” places together is the most effective way to create the most “efficient” path. To make it simpler, I used python instead of PySpark. We do not have to use PySpark in this case because the remaining of our data after getting thoroughly filtered are not that large.

My general assumption is that a person cannot visit more than 3 locations in 1 day because it would be way too overwhelming and can possibly take a toll on their body. The metric I am using in this case is haversine distance. The formula is as follow:



# I used the formula to calculate the distance of a location to each of the other interesting places and then try to get the nearest ones. A list of indexes is used to make sure that no places get repeated.

The total travel distance is provided at the end to help compare the actual distance in km a person has to go in they follow that path.

# Limitations/Retrospective

It is possibly better if I opted to use a combination of nearest interesting places and interest score to provide a better result. Excluding out interest score may have a huge impact on our conclusion. Furthermore, in my output file of recommended paths, there might be occurrences where a path could lead the user to Starbucks and then the next location is Tim Horton. So, filtering out any locations that share the same name or amenity as the previous visited places can further enhance the accuracy or efficiency of the overall outcome.

# Sources

[1] <https://www.openstreetmap.org/>

[2] <https://www.wikidata.org/wiki/Help:Sitelinks>

[3] <https://www.wikidata.org/wiki/Q2>

[4] <https://www.wikidata.org/wiki/Q37158>

[5] <https://medium.com/spatial-data-science/how-to-extract-gps-coordinates-from-images-in-python-e66e542af354>

[6] <https://en.wikipedia.org/wiki/Haversine_formula>

[7] <https://stackoverflow.com/questions/59406045/convert-pandas-series-into-a-row>

**Reece McGowan Project Experience Summary**

Created a model using PySpark to find interesting entities in a collection of photographs in the Vancouver area. This was done by loading over 17,000 amenity entries from the OpenStreetMap project, and then applying a custom heuristic analysis to modify the data to give each a ranking. This involved looking at the data to boost or suppress certain entries as well as interacting with the Wikidata REST API to append useful information. This results in a tool which takes as input geotagged photos of attractions in Vancouver and returns the most likely thing of interest in the picture.

**Trung Hieu Le Project Experience Summary**

Provided a list of recommended paths for a typical user to visit 3 different interesting places that are in nearest to each other. This was done by manipulating the given dataset to exclude any “boring” data points. Haversine Distance was used to measure the proximity of a location to others, and then from that only the 3 nearest places were selected. The final result is a list of sequences (paths) of interesting places with their latitude, longitude, and the total distance a person has to travel in order to visit all of them.