MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY

BHOPAL (M.P.)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

MAJOR PROJECT

ON

LOCATION-BASED RECOMMENDER SYSTEMS

Submitted in partial fulfillment for the degree of Bachelors of Technology in

Computer Science and Engineering

(SESSION 2017-2018)

UNDER THE GUIDANCE OF:

Dr. S. K. Saritha

Submitted By:

Ravuri Amritha Vardhini – 141112084, Siva Likitha Valluru – 141112009,

Kanchan Chauhan – 141112285, Suneel Kumar Jatav – 141112264

MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY

BHOPAL (M.P.)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

CERTIFICATE

This is to certify that **Ravuri Amritha Vardhini, Siva Likitha Valluru, Kanchan Chauhan, and Suneel Kumar Jatav,** students of B. Tech, Fourth Year, have successfully completed the paper "**LOCATION-BASED RECOMMENDER SYSTEMS**" in partial fulfillment for the degree of Bachelor of Technology in Computer Science and Engineering.

Dr. S. K. SARITHA (Project Guide)

Dr. NAMITA TIWARI (Project Coordinator)

MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY

BHOPAL (M.P.)

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

DECLARATION

We, hereby, declare that the following report which is being presented in the project documentation entitled **"LOCATION-BASED RECOMMENDER SYSTEMS"** is the partial fulfillment of the requirements in the field of Computer Science and Engineering. It is an authentic documentation of our own original work carried out under the valuable guidance and the dedicated coordination of Dr. S. K. Saritha. The work has been carried out entirely at Maulana Azad National Institute of Technology, Bhopal. We, hereby, declare that the facts mentioned above are true to the best of our ability and knowledge.

Ravuri Amritha Vardhini – 141112084
Siva Likitha Valluru – 141112009
Kanchan Chauhan – 141112285
Suneel Kumar Jatav – 141112264

ACKNOWLEDGEMENT

With due respect, we express our deepest gratitude to our respected guide and coordinator, Dr. S. K. Saritha, for her valuable support, time, patience, and guidance. We are also very thankful for the encouragement that she has given us in completing this project successfully. Her rigorous evaluation and constructive criticism was of great assistance.

We are also grateful to our respected director Dr.N.S.Raghuwanshi for permitting us to utilize all the necessary facilities of the college.

Needless to mention is the additional help and support extended by our respected HOD, Dr. Meenu Chawla, in allowing us to use the departmental laboratories and other services.

We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind co-operation and help.

Last but certainly not the least; we would like to express our deep appreciation towards our family members and batch mates for providing the much-needed support and encouragement.

ABSTRACT

Smartphones have become a primary platform for information access. Every day, an increasing number of people bring smartphones into their lives for various purposes especially communication. To increase the usability and offer leisure access to end users, mobile phones have incorporated recommendation systems (recommender systems or recommendation engines). A recommendation engine is a software that analyzes available data using various algorithms to make suggestions for an item that a user might be interested in among other possibilities. One of the foremost applications of mobile recommender systems is using location to support several services such as a travel system using Global Positioning System (GPS). A typical location-based travel system would need a good recommender system to predict a sequential optimal path for a given set of pick-up points. Conventional statistical tools are simply not enough to analyze real-time and dynamically changing instantaneous data. Therefore, recommendation systems are used to filter an information system's database and retrieve the most probabilistic desired result for the user. In this case, the desired result would be the most profitable (and preferably less time-consuming also) path from a source and destination.

Table of Contents

List of Figures

INTRODUCTION

Mobile phones are becoming a primary platform for information access. Many people rely on these devices as communication and information access tools. The functionalities and the challenges provided by these devices are growing increasingly exponentially. Finding and implementing a way to give optimal solutions is surely one of the primary application areas being currently and rapidly developed for mobile applications.

The area that this project focuses is chiefly on location-based driving routes. An incredible number of services are now offered to support travelers before, during, and after their journey. Hence, it is important to understand the capabilities of this channel and the information access behavior of mobile users.

Moreover, as the amount of information and online services increases, it becomes increasingly difficult for end users to find the right information needed to complete a particular task. This happens often especially for users of e-commerce websites because they find it difficult to locate the best products and services, mostly likely due to the overwhelming number of options to consider and the lack of effective system support in making decisions. Recommender systems (RSs) are information filtering and decision support tools aimed at addressing these problems while providing product and service recommendations personalized to the user’s needs and preferences at each request.

A recommendation system is a software program which attempts to narrow down selections for users based on their expressed preferences, past behavior, or other data which can be mined about either that user or other users with similar interests. It is a subclass of information filtering systems that seeks to predict the "rating" or "preference" that a user would give to an item.

The purpose of this project is to analyze the parameters that a location-based recommender system will use and generate a series of pick-up points, increasing the accuracy of the recommendations. One of the main challenges while implementing a location-based

recommender system is dealing with computation power which exponentially grows due to huge and sparse datasets. Extracting, integrating, and preprocessing this data are prominent issues and should not be ignored or taken lightly. Due to enormous amounts of dynamically changing data, mishandling data could lead to complex calculations. Complex calculations lead to mistakes. And these mistakes lower the accuracy by a significant amount. Even if the algorithms are well-defined, if data takes a long time to process, the system will be of little use and produce results with low accuracy. Therefore, proper steps must be taken to ensure that efficiency is maintained.

This project incorporates location-based information and uses that data to compute a series of pick-up points. The application integrates a recommender system to implement a geographical-based recommender system and suggest an optimal path sequentially for a series of given pick-up points. This application is especially designed for cab drivers who may be new to a place and are aiming to travel through potential pick-up points where the chance of picking up a passenger is high. The optimum sequential route is displayed in a way such that the result yields maximum profit while travelling through less number of pick-up points.

Some drivers prefer to travel long distances in hopes for more number of passengers, increasing the drivers' chances for higher profits. Others favor for travelling less distance in hopes of the same profit. The project gives a choice to the end user who can specify the number of pick-up points he/she wishes to travel. The result is calculated pertaining to the wish of the cab driver and is displayed accordingly, all the while optimizing the code such that the result is computed and displayed faster.

LITERATURE SURVEY AND THEORETICAL ASPECTS

2.1 RECOMMENDER SYSTEMS

Recommender systems have become increasingly popular especially since the last decade and are integrated in various areas such as music, movies, news, medicine, technology, social media, sales, etc. Smartphones, smart TVs, Facebook, Amazon, Twitter, eBay, Flip Kart, and many more commercial websites use recommender systems. The World Wide Web (WWW) is growing each day and so is the number of users on the Internet. Organizations, companies, and entrepreneurs are building intelligent recommendation engines to study a user's behavior over a certain interval of time. It is extremely important for them to provide users with recommendations according to their preferences and choices [1,2,6,7,8]. These systems try to find a user's interest by discovering data patterns based on three approaches [3]:

- Content-based filtering
- Collaborative filtering
- Hybrid-based filtering

2.1.1 Content-based Filtering

Content-based filtering (also known as cognitive filtering) methods output users' probable interests based on their actions on other items in the past [5]. This algorithm checks what a user liked in the past and what he/she likes in the present to determine what he/she might like in the future. For example, checking a user's ratings and reviews (using deeper sentiment analysis) to recommend items that are similar to those is a way. Basically, this model focuses on the user's previous (and current) preferences, user profile, and his/her interaction levels with the system. For example, a user's behavior may consist of his/her browser history, pages he/she visited, links he/she clicked on, items he downloaded, search queries he typed, and so on.

After taking user's information, the focus is shifted towards the items. An item's attributes, traits, characteristics, and properties are gathered to determine the category of a user's interest. The features of the item are weighted accordingly. The stored information in this type of filtering is usually represented in the form of text documents. So, each word in a document is labelled as a term. Similarities between documents are ignored and are not part of the prediction. The differences between the documents are ranked in order are outputted. Simple methods of calculating weights would be computing the average mean of all the weights. Advanced methods such as machine learning algorithms like decision trees, classifiers, clustering methods, and neural networks use sophisticated means to calculate the probability that shows a user's interest in that item or feature. In advanced weighted methods, each weight holds a level of importance. Each feature of an item is compared with features of other items to see if there is any similarity between the two.

The weighted methods also depend on the way a user has reviewed the item. For example, if there are like or dislike buttons or one-to-five-star rating methods, then it is not that easy to determine what features the user liked in that item. However, if the user has provided feedback in oral words, then it is a little easier to determine what the user likes. But even this approach is definitely not guaranteed and depends on how much information the user provided. If the users have taken the time to explain in detail what they liked/disliked in the system, then it would help the system to learn about their views. However, few users have the time and patience to list out everything.

2.1.2 Collaborative Filtering

Another approach to design recommender systems is by using collaborative filtering. Collaborative filtering methods gather and analyze a user's information and liked items similarly to content-based filtering [4]. Unlike content-based filtering, collaborative filtering recommends users the items that other users have shown interest in. The items that are being recommended come from the users who have shown similarity with the current user. This information is later used to analyze any plausible patterns or trends that may

arise. An assumption that this technique makes is that people who liked an item in the past will also like it in the future. An advantage to collaborative filtering is that the system is not required to understand a user's behavior. It just has to gather the data about the user and items. The new recommendations will come from other users who have shown similar interest in that item.

There are three problems associated with collaborative filtering:

i. Cold start

Since collaborative filtering recommender systems need user and item information as well as data from many other users, an absurd amount of storage is necessary to store this data. In order to make new recommendations, data has to be imported from secondary storage into main memory. Collaborative filtering is not recommended is the size of secondary storage is too less. Otherwise, this would result in a loss of accuracy.

ii. Scalability

Since large data is required for storage, it is important to note that heap errors do not occur while processing this data. To avoid such errors, a large amount of computation power is required to make new recommendations.

However, since this project is a mobile-based system, then it's important for the processing power to be as low as possible. Since phones have a low disk space and RAM as compared to computers, then they will not be able to process and handle massive data.

iii. Sparsity

As established earlier, databases in collaborative filtering are colossally large. Even if many users are registered onto websites, each user would have rated only a very small subset of the overall database. From the database size's perspective, even the most trending items would have very few ratings. When these ratings are represented as a matrix, then the matrix will look very sparse.

2.1.3 Hybrid Recommender Systems

Hybrid recommender systems are those systems which incorporate both collaborative filtering and content-based filtering. This approach is effective sometimes. Hybrid systems can be implemented in diverse ways. Recommendations can be made by using both of the methods separately and then combining them. Usually the union of both models is taken, but some systems also accept intersection of the systems. A hybrid system's performance is evaluated by comprehensively comparing the system's performance between solely content-based filtering methods and collaborative filtering methods. Methods of the same type can be combined together also. For example, multiple collaborative filtering methods can be used together to increase the accuracy of the prediction.

2.1.4 Types of Input

When a user gives a feedback to the system about the item, this feedback is taken as input into one of the algorithms explained above. There are two types of input:

i. The deliberate and intentional action of a user rating an item or reviewing it is known as ***explicit recommendation***.

ii. A user's action of clicking an item is known as ***implicit recommendation.*** For example, watching a video, reading half of a book and not completing it, are examples of implicit input. These actions of the users are all taken into consideration to recommend an item.

2.1.5 Types of Output

Whether the input is of explicit or implicit, there are two forms of output:

i. A similar item or an item based on the user's actions is known as a ***recommendation.***

ii. A ***prediction*** is the probability that a user might like the recommended item.

2.1.6 Factors That Play A Vital Role in Recommender Algorithms

When designing recommendation systems, poor recommendations and suggestions will obviously not be received well among users. It is important to maintain the accuracy as highly as possible. There are a number of factors that every programmer should consider while developing the system:

i. Privacy and trustworthiness

This is one of the biggest issues that recommender systems have to consider. In websites like Amazon, users are required to give out sensitive information like name, address, and credit card information. Before asking to give out this information, users have to be clearly shown and explained to as to whom and why they are giving out this information.

ii. Robustness

Robustness refers to durability and stability of a recommender system. The more durable it is, the safer the users feel, i.e., allowing privileged access to just about anyone would result in fraudulent identities. When users interact with the recommendation engine, they should be assured that they are moving about in a safe environment. Many websites ask registered users to give out sensitive data. These sites are more susceptible and vulnerable to attacks. Therefore, the system should be strong enough to detect, prevent, and defend against such attacks.

iii. Serendipity

Serendipity is a measure of how oscillating the recommendations are. Users will eventually become bored and dissatisfied with the obvious suggestions. For example, asking a user to buy notebooks in a stationary shop is a good recommendation, but clearly obvious. Serendipity is concerned with giving surprised results, but also useful ones at the same. It has become difficult for researchers to implement both serendipity and usefulness in the same system because of the complexity. High serendipity scores mean that the

recommendations are too random for the user to register. This could hurt the system and lower its accuracy.

iv. Recommender persistence

Sometimes, it is better to show the same recommendations over again to the user and allow the user rate the same item before suggesting new ones. There is a reason for not showing new recommendations each time a user clicks on an item. Users are oblivious. They usually do not take the time to view each and every new recommendation. Instead, they only focus on the item that they want. So, instead of showing new suggestions on each and every page, reshowing the same or related items might catch their attention.

v. Diversity

Diversity is related to serendipity. Serendipity refers to how surprising the results are, but diversity refers to broader categories. Users tend to be more intrigued with recommendations from different artists. This is mostly appreciated in applications such as entertainment applications like Netflix and YouTube.

vi. Domain

The domain is the content which is being recommended. It should be relevant. For example, in an application such as Netflix, suggesting a news article would be inaccurate and irrelevant since Netflix is a movie recommendation application.

2.2 CLUSTERING ALGORITHMS

2.2.1 DBSCAN Algorithm

The DBSCAN (density-based spatial clustering of applications with noise) algorithm is a clustering algorithm such that when given a set of points, the algorithm compiles the data and clusters the points according to density. Points that lie far off from a density grouping are identified as outliers. There are four types of points seen in DBSCAN [14]:

i. Core points

Point *P* is identified as a core point if there are at least minimum points, denoted by *min_Points,* within a distance ε (epsilon), where ε is the maximum radius taken from *P*.

ii. Directly reachable points

Point *Q* is directly reachable from *P* if *Q* is within near vicinity from *P* (within ε distance) *and P must* be a core point.

iii. Reachable points

Point *Q* is reachable from *P* if there exists a path $p_1, p_2, \ldots, p_n$ where $p_1 = p$ and $p_n = q$, where each p_{i+1} is directly reachable from p_i. All the points on the path should be core points (*Q* does not have to be so).

iv. Outliers

All points which do not fall into the above three categories, i.e., not reachable from any other point, are known as outliers.

2.2.2 k-Means Clustering

k-means is a popular clustering algorithm which partitions a given set of *n* points into *k* clusters where each point belongs to one cluster according to the closest mean value. It, however, is computationally difficult and extremely time-consuming especially if the given dataset is a large one [14].

k-means uses centers of clusters to model data to form groups which have not been explicitly labelled in dataset. In other words, similar data or points akin to each other are grouped together and this group, in turn, is known as a cluster. Each centroid identifies a cluster. Each point is assigned to the nearest regional centroid, based on squared Euclidean distance. The algorithm iterates many times, where in each iteration, centroids are recomputed. The iteration process stops where the condition is met: the data points stay in their respective clusters and no more changes in the current iteration are noted from the

previous iteration. The algorithm will converge into a result although there is no guarantee that the result is the best result. *k*-means fails to acknowledge outliers or erroneous data points. Another disadvantage to this clustering method is that this is an iterative process. This approach proves to be infeasible in unquantifiable datasets. Therefore, *k*-means is not a preferred approach.

2.2.3 k-Medoids Clustering

The *k*-medoids clustering algorithm is akin to *k*-means algorithm. Both of them are approaches to partition the dataset into groups. A medoid of a finite dataset is a most centrally located data point within the set. The clustering is done in a way such that the objects in one cluster are more similar with each other than with any other data points in another cluster. The method is very similar to *k*-means clustering, but *k*-medoids is more resilient to noise and outliers, which the former method is unable to handle well. The most common and generalized method is the Partitioning Around Medoids (PAM) algorithm which uses a greedy search method. Although optimal solution is not guaranteed all the time, the process is guaranteed to be faster than an exhaustive search. However, *k*-medoids is usually used for small datasets. Therefore, this approach also prevents efficient clustering [14].

2.3 TRAVELLING SALESMAN PROBLEM (TSP)

The travelling salesman problem (TSP) is an optimization problem. When given a group of *N* cities, TSP finds the shortest possible route that visits each city and ends up back at the starting point or origin city. TSP tries to minimize the distance travelled so each city is visited only *once*. This project is inspired from TSP, but also includes more parameters and a novel and realistic method.

2.4 MOBILE SEQUENTIAL RECOMMENDATION (MSR)

The mobile sequential recommendation (MSR) problem is similar to the traditional TSP. MSR is used in this project [12,14]. It suggests a route among *N* pick-up points for a taxi driver such that he/she can get more passengers with minimized travelling cost. Since this is a shortest path problem, the source and destination are known in advance. The main difference between TSP and MSR is that TSP makes sure to cover all *N* locations while MSR does not necessarily have to. MSR covers optimal subset of *N* locations. In this project, there are two problems that every system has to consider first:

i. Primarily, high processing power is required for complex computations. Since the pick-up points are uncertain, there are countless possible routes between source and destination.
ii. Travelling cost is another issue *especially* when a cab's operational status is 0 (i.e., the cab is empty).

To take care of both challenges, researchers have proposed a few temporal solutions which are explained in the later sections.

2.5 PROBLEM FORMULATION USING MOBILE SEQUENTIAL RECOMMENDATION

Consider a scenario where a large number of GPS traces have been collected from taxi drivers over a certain period of time. In this collection, location, timestamps, and operational status are included. Using this dataset, a mobile sequential recommendation (MSR) problem is formulated. MSR is presented to suggest a travel route connecting pick-up points sequentially for a taxi driver such that he may pick up more passengers with less travel cost. One of the key challenges of implementing this problem is its high computational complexity typically rising exponentially or in terms of large polynomials.

Although MSR sounds similar to the traditional Travelling Salesman Problem (TSP), both of them are slightly different. TSP evaluates a combination of exactly N given locations where all N locations have to be involved, i.e., all the locations must be included in the route. However, MSR problem deals with only a subset of the given N locations. The reason for this is because MSR deals with uncertain pick-up points.

Let T be the taxi (*PoCab* is the position of the cab) and C1, C2, C3, and C4 be the pick-up points. Assume operational status of T is empty. This means that there are no passengers in the cab.

The problem statement here is to suggest an optimal route in such a way that the cab driver gains the maximum profit by traversing along the path, i.e., the sequence of pick up points recommended to him in minimum time. Say, he chooses to go to C1 first. Here, P(C1) stands for the probability of finding a person at location C1. Even if P(C1) > P(C4), there is also a probability of not finding a person at this location. In this case, T has to travel back to *PoCab* and follow the path where C4 is on the way. This is a waste of both time and cost. This is especially seen in case P(C1) < P(C4). So, in both ways, T is at either loss or no profit. So, even if P(C1) is higher than P(C4), the system should recommend to T to traverse along to C4. So, first, C4 is recommended rather than C1. C1 has second priority. Also, from an external point of view, it is clear that C1 appears to be an outlier location where C4, C3, and C2 form a cluster [13].

After collecting a dataset of GPS traces, it can be observed that there will be some pick-up points more frequently visited than others. In other words, there will be some regions where clusters of location points exist. The centroids of these clusters will be used to recommend pick-up points. Centroids are calculated using driving distance measure instead of the conventional Euclidean distance. The driving distances are observed during different time intervals of the day and also during different time periods (such as over the course of one year). Not only will clustering approach produce better results but it will also greatly reduce the computational cost of MSR problem.

The input for MSR consists of a set of parameters: a set of *N* plausible pick-up locations *C1, C2, C3, ..., CN,* a set of probabilities *P(C1), P(C2), P(C3),..., P(CN),* where *P(CN)* refers to the probability of finding a customer at location *CN,* and position of the cab driver at *PoCab.* There is another function considered while designing MSR. The Potential Travel Distance (PTD) observes each pick-up point's probability by looking at past events. If there are no events of customers getting inside a cab for a certain pick-up point, then the probability of getting passengers in that location will be zero. In case of small number of data points, PTD function will yield results quickly. However, in terms of large data, analyzing all of the points is time-consuming. Therefore, it will require lots of processing power.

To avoid computational power, a constraint, *L,* is added. It is practically not possible to search for an optimal solution of the general MSR problem because this problem requires an unreasonable amount of computation. *L* stands for length and is used to suggest an optimal route. Instead of suggesting an optimal path by traversing through *all* of the pick-up points (similar to Travelling Salesman Problem (TSP)), only *L* number of pick-up points are used in the suggested sequential route.

2.6 POTENTIAL TRAVEL DISTANCE (PTD) FUNCTION

The Potential Travel Distance (PTD) function is used to take care of the problem in MSR. This function will increase the probability of finding new passengers in the proposed path by focusing on populated pick-up points. To lessen the computing cost, PTD manages drivers to get around to a subset of the locations in an efficient way that enhances the drivers' business and minimization time and efforts [12,13].

Suppose, the cab driver's route is represented by $\overline{R^L}$ and the probability of each pick-up event is represented by $P(C_i)_{1 \le i \le N}$ where C_i stands for each cluster. A pick-up that happens at C_i will have probability $P(C_i)$ and a pick-up event that does not happen at C_i will have the probability $\overline{P(C_i)} = 1 - P(C_i)$. For a cab at some initial position, say *PoCab,* that has a

recommended driving route *PoCab* → C_1 → C_2 → C_3 → C_4 where length of the suggested route as $L=4$, there is still a possibility that pick-up events do not happen even after driving through all four points. In that case, the probability that zero pick-up events occur is $\overline{P(C_1)} \cdot \overline{P(C_2)} \cdot \overline{P(C_3)} \cdot \overline{P(C_4)}$. We take another variable, *D*, to denote the travel distance. The distance travelled to each pick-up point in the recommended route is expressed in the vector $D_{\overline{R^L}} = (D_1, (D_1 + D_2), (D_1 + D_2 + D_3), (D_1 + D_2 + D_3 + D_4, D_\infty)$. D_∞ is set when pick-up events do not occur along all points of the recommended route. Similarly, like *D*, *P* is used to represent the probabilities: $P_{\overline{R^L}} = (P_1, \overline{P(C_1)} \cdot P(C_2), \overline{P(C_1)} \cdot \overline{P(C_2)} \cdot P(C_3), \overline{P(C_1)} \cdot \overline{P(C_2)} \cdot \overline{P(C_3)} \cdot P(C_4), \overline{P(C_1)} \cdot \overline{P(C_2)} \cdot \overline{P(C_3)} \cdot \overline{P(C_4)})$. Finally, the PTD function *F* is defined as the mean of the travel distance distribution:

$$F = D_{\overline{R^L}} \cdot P_{\overline{R^L}}$$

F is the dot product of the two vectors.

2.7 POTENTIAL RESEARCH GAPS IDENTIFIED

One of the biggest gaps identified while implementing this project is the constraint in TSP. According to TSP, if there are *N* given pick-up points in a map, then it is mandatory to visit all *N* points and generate an optimal path where each and every point is present somewhere along the route. To overcome this issue, MSR is presented to suggest a travel route connecting an optimal *subset of N* pick-up points sequentially for a taxi driver such that he/she has a greater chance of picking up more passengers with less travel cost.

Although MSR takes care of the problem in TSP, the former presents a key challenge in return. Implementing MSR takes high computational complexity because many combinatorial subsets exist of different lengths. The asymptotic time complexity rises exponentially or in terms of very large polynomials. To take care of this issue, a constraint, *L*, is taken. *L* stands for the total length of the route, i.e., total number of pick-up points present in the route. This way, the time complexity is heavily reduced because instead of

instead of traversing through every possible combination of every length, it is easier to consider only one specified length.

2.8 PROBLEM DEFINITION

The problem here is known as mobile sequential recommendation (MSR). The input for MSR consists of a set of parameters: a set of plausible pick-up points C1, C2, C3,...,CN, a set of probabilities P(C1), P(C2), P(C3),..., P(CN), where P(CN) refers to the probability of finding a customer at location CN, and position of the cab driver at PoCab. There is another function considered when designing MSR. The Potential Travel Distance (PTD) is introduced. PTD observes each pick-up point's probability by looking at past events. If there are no events of customers getting inside a cab for a certain pick-up point, then its probability will be zero. In case of small number of data points, PTD function will yield results quickly. However, in terms of large data, analyzing all of the points is time-consuming. Therefore, it will require lots of processing power.

To avoid computational power, a constraint, *L*, is added. It is practically not possible to search for an optimal solution of the general MSR problem because this problem requires an unreasonable amount of computation. *L* stands for length and is used to suggest an optimal route. Instead of suggesting an optimal path by traversing through *all* of the pick-up points (similar to Travelling Salesman Problem (TSP)), only *L* number of pick-up points are used in the suggested sequential route [12,13].

PROPOSED ARCHITECTURE

3.1 ARCHITECTURE

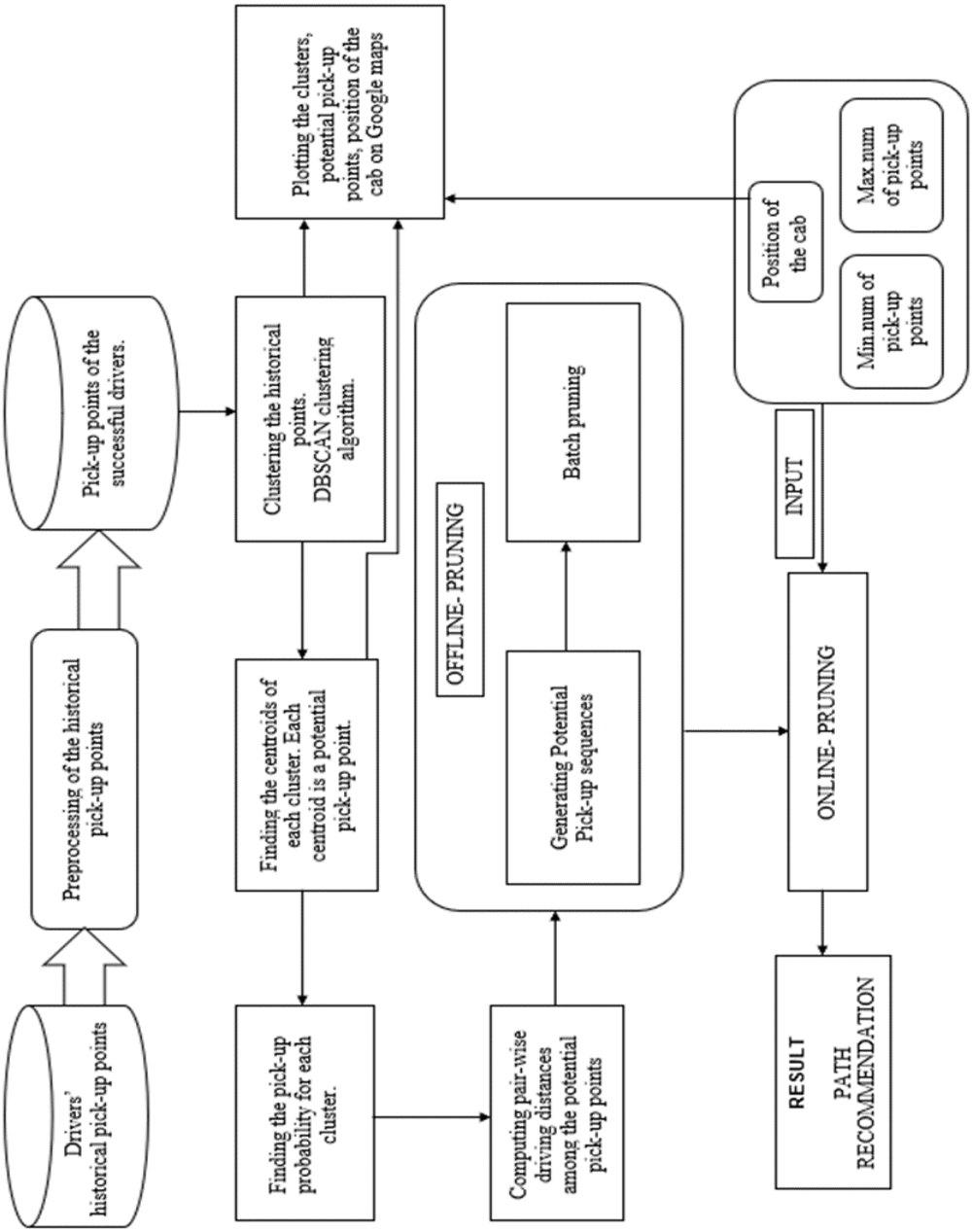

Figure 3.1: Flowchart of Location-based Recommender System

3.2 PROPOSED WORK

i. Preprocessing

The dataset is collected and preprocessing is done on the list of files, where, each file represents a cab driver. List of successful cab drivers are collected basing on their occupancy rate and the number of hours each cab driver has travelled.

ii. Clustering

The cab drivers' data is collected and the historical pick-up points are clustered based on the spatial density of the points.

iii. Finding centroids

The centroid of each cluster is computed which represents as a potential pick-up point of each cluster.

iv. Finding pick-up probabilities

The pick-up probability of each cluster is evaluated as the number of pick-up events in a cluster divided by the number of empty cabs travelling in a cluster.

v. Computing pairwise driving distances

The pair-wise driving distances are computed between the centroids of each cluster using Google Maps API.

vi. Offline pruning

Each possibility of sequence of pick-up points is considered and pruned using Potential Travel Distance (PTD) Function which will be seen further.

vii. Online pruning

During online-pruning, the position of the cab and the length of the path is provided. Using PTD function, the remaining candidate sequences are pruned and the results are produced.

METHODOLOGY

4.1 PREPROCESSING

4.1.1 Flowchart

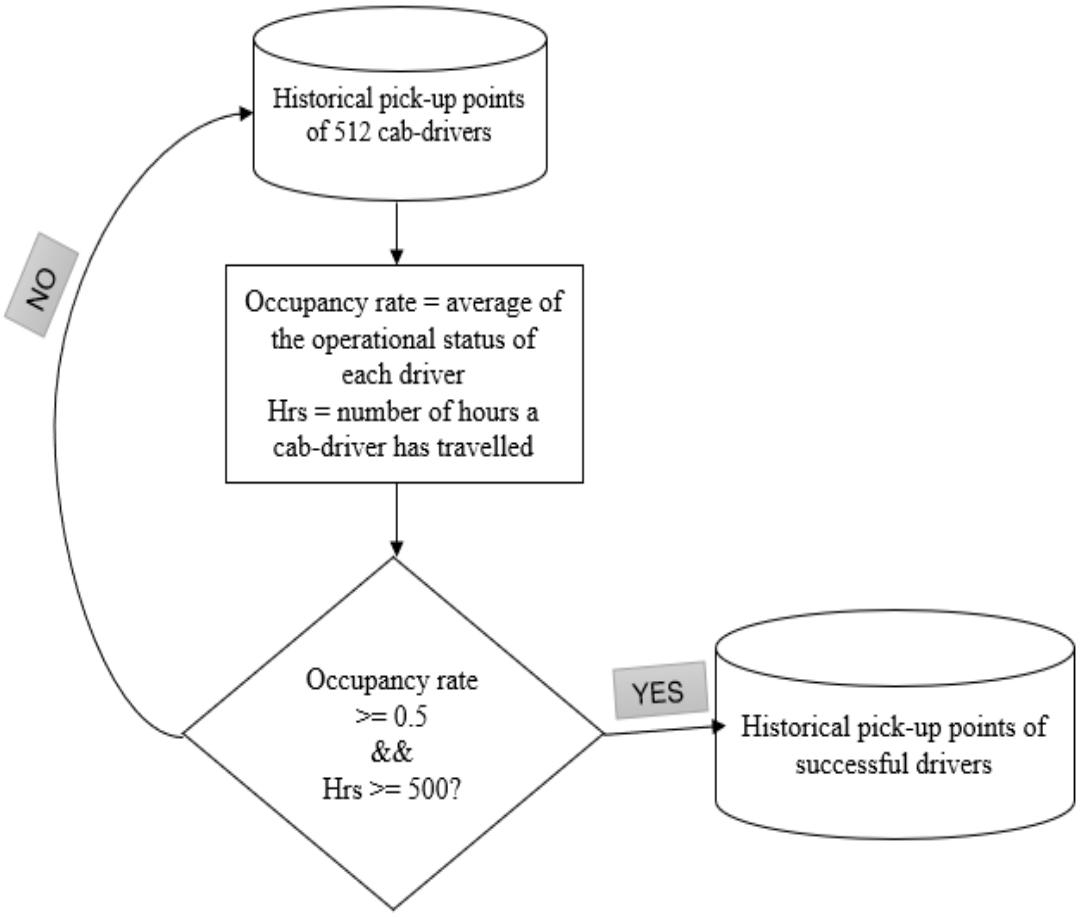

Figure 4.5: Preprocessing of Dataset

The above flowchart depicts the preprocessing of Dataset. First, the historical pick-up points of 512 cab-drivers are collected and are filtered using the occupancy rate and total number of driving hours.

4.1.2 Algorithm

Input: 'cabspotting' dataset directory containing all the files representing the drivers'routes
Output: preprocessed text files

1. // inputs: 'cabspotting' dataset directory containing all the files representing the drivers'routes
2. // each line in a file contains 4 attributes namely, latitude, longitude, operational status and timestamp.
3. // output: Result is a file containing all the traces of successful drivers in the 2PM-3PM duration.

4. for each present in the 'cabspotting' data directory
   {
   compute the occupancy rate and the number of hours the cab-driver travelled;
   if (occupancy rate >= 0.5 and number of hours >= 250)
   {
   write the filenames to a file f1; // These files represent the successful cab-drivers
   }
   }

5. for each line in f1
   {
   file = read_line(f1)
   for each line in file
   {
   convert the epoch timestamp into GMT time zone
   if (time zone is between 2PM to 3PM)
   {
   write the points to a file f2;
   }
   }
   }
6. return f2;

}

The dataset is of 512 files with approximately 20,000 records in each [29]. Perusing each record would take an unquantifiable amount of time. The time complexity that arises from such scanning is not favorable and therefore, many files have to be removed from the algorithm.

The dataset is pruned using two parameters. The third parameter, operational status, is studied and analyzed for each and every unique cab driver. Next, the last parameter, the location trace timestamp, is used. Since each file already has records in descending order, the range of the number of hours travelled by the cab driver is calculated by considering the maximum and minimum timestamps. If the average of the operational status of all location traces is greater or equal to 0.5 and if the number of hours that each driver travelled is greater than or equal to 500, then this cab driver is labelled a *successful cab driver.*

This process is repeated for every cab driver. Those drivers that satisfy the condition are taken as successful drivers and the rest are removed. This means that only high-performance drivers with high occupancy rates and adequate driving hours are considered. After pruning datasets, the size of the set has been reduced from 512 to mere 20. Clearly, twenty datasets are better for usage rather than using all 512.

Next, for each of these twenty sets, the timestamp has been converted from Unix Epoch time to its proper date and GST time in a 24-hour format for the sake of simplicity.

Drivers show different performance levels at various times throughout the day. Instead of taking pick-up events from all 24 hours, a sample time period is considered which is between 2:00 PM to 3:00 PM. The resulting set consisted of historical pick-up points of all the cab drivers, where the total size of set is 15808.

4.2 CLUSTERING BASED ON DENSITY OF PICK-UP POINTS

Relatively speaking, drivers experience higher pick-up events at some places than others. Therefore, clustering proposes to group those historical pick-up points of successful cab drivers into density-based clusters. DBSCAN algorithm is used for clustering those pick-up points. As explained in the literature review previously, DBSCAN clusters the pick-up points according to density. There are four input parameters to the algorithm: epsilon (ε), minimum number of samples to be taken, algorithm, and metric. As explained previously, epsilon is the radius distance from a pick-up point. Minimum number of samples to be taken, *min_Points*, represents the number of points that should be present with the specified distance, ε. The third input is the 'ball-tree' algorithm. The specified algorithm is called ball tree algorithm. The purpose of ball tree algorithm is to organize points in a multi-dimensional space. The last input is a distance metric. The dataset consists of geographical data. This means that a measure like Euclidean distance, the shortest distance between two given points, is not used to compute the distance between latitude and longitude. The real world does not have straight roads. In order to take care of that issue, another type of distance is used. The ***great-circle distance*** gives an approximate driving distance between two given points. In this case, the two points constitute of a location. This distance is also known as Haversine distance. It is tightly coupled with ε.

The reason why DBSCAN algorithm is used is because it outsmarts many other clustering algorithms [14]. *k*-means and *k*-medoids are popular algorithms but there are several drawbacks. The biggest disadvantage is that an estimated number of clusters has to be provided initially before performing the algorithm. However, DBSCAN clusters data points based on density. *k*-means is an exhaustive clustering algorithm. With large datasets in inventory, a brute-force search will exhaust and use up so much processing and computing power. This is always unfavorable and therefore, an algorithm that automatically takes care of computing issues is preferred. In addition, *k*-means is not at all

effective especially when there is an uncertainty in the number of outliers. *k*-medoids, on the other hand, is better at taking care of outliers but it can only deal with small datasets. Therefore, DBSCAN is preferred over both of them [16,17,19,21,23,25,27].

As a side note, clustering will significantly diminish the computational cost problem explained in MSR problem.

4.3 CALCULATION OF CENTROIDS

DBSCAN clustering is applied over 15808 pick-up points. The result gave five clusters, not counting outliers. Using all 15808 points as potential pick-up points is not a feasible choice. Therefore, the centroids of the five clusters are taken as potential pick-up points [19,20,24,28].

4.4 PROBABILISTIC CALCULATIONS

4.4.1 Algorithm

Input: A file ‘f’ containing the pick-up points of all the cab-drivers, which has five attributes each: cab-driver, latitude, longitude, operational status and timestamp

Output: A set of pick-up probabilities ‘prob’ at each cluster

1. ‘pick’ and ‘empty_cabs’ are the lists containing number of pick-up events happening at a cluster and the number of empty_cabs at a cluster respectively.
2. for c in range of number of clusters:
    a. pick[c] = 0
    b. empty_cabs[c] = 0
3. for each line in f:
    a. l = line. split ()
    b. m = nextline. split()
    c. if (m [operational status] == 0): # It means the cab is empty
        i. empty_cabs[c]++, where ‘c’ belongs to a cluster

ii. if (l[operational status] ==1): # It means a pick-up event took place

1. pick[c]++ where ‘c’ belongs to a cluster

4. prob[c] = pick[c] / empty_cabs[c] , for all ‘c’ in clusters
5. return prob
6. End Algorithm

The Potential Travel Distance (PTD) function increases the probability of finding new passengers *optimally* in the proposed path by focusing on populated pick-up points. the PTD function *F* is defined as the mean of the travel distance distribution:

$$F = D_{\overline{RL}} \cdot P_{\overline{RL}}$$

Here, $D_{\overline{RL}}$ is the distance vector and $P_{\overline{RL}}$ is the probability vector.

In this section, the probability vector is calculated. The probabilities are calculated to measure how the success of each pick-up point, i.e, the profit a pick-up profit can bring. The parameters in the input are the number of empty cabs and the number of pick-up events. Let *#r* denote the number of empty cabs and *#p* denote the number of pick-up events. The probability, *P(Ci)*, is calculated by:

$$P(Ci) = \#p/\#r, \text{ where } 1 < i < N$$

Here, *#p* and *#r* are calculated for each pick-up event at different time periods [13].

4.5 PAIRWISE DRIVING DISTANCE CALCULATIONS

As seen previously, currently there are five potential pick-up points. In order to calculate the distance vector, $D_{\overline{RL}}$, a distance matrix is taken of *NxN* where *N* is the number of centroids. In this case, since the number of centroids is five, the matrix size is 5*x*5. Then, the driving distance between each possible pair is calculated using the Google Maps API and is inputted into the matrix.

4.6 OFFLINE PRUNING

Offline pruning is mainly done to diminish the computational complexity. In online pruning, the position of the cab driver is given. But before that, the non-optimal paths are sequentially generated and are pruned effectively [12].

4.6.1 Algorithm

Input: A set of potential pick-up points *C*, the probability set *P* for all pick-up points, the pairwise driving distance matrix of pick-up points

Output: A set of potential sequence candidates *R* with length *L* from 1 to *N*

1. Initialize *R1* as a null set.
2. For each point in *C:*
    a. *r.append(c)*
    b. *F(r) = 0*
    c. *PE(r) = P(c)*
    d. *R1 = R1.append(r)*
3. For *L* from 2 to *N:*
    a. For each r in R(L-1):
        i. Generate possible sequences *p* of length *L*.
        ii. Compute F(*p)* and PE(*p)*
        iii. For each *q* in R(L):
            1. If (source[*p*] == source[*q*] && length(*p*) == length(*q*)) do:
                a. *Prune(L).append(q)*
        iv. If (Prune(L) is empty) do:
            1. *R(L).append(p)*
        v. Else: #if *F(p) < F(q) for all q in R(L)*
            1. *R(L).append(R(L)-Prune(L))*
            2. *R(L).append(p)*

4. End algorithm

4.6.2 Potential Sequence Candidates Generation

Suppose, $\vec{r}$ is a sequence vector.

According to the definition of distance vector mentioned in Potential Travel Distance (PTD) function previously,

$$D(\vec{r}) = \langle D_{c_1,c_2}, \left(D_{c_1,c_2} + D_{c_2,c_3}\right), \dots, \sum_{i=2}^{L} D_{c_{i-1},c_i} \rangle$$

Similarly, the probability vector for pick-up events is:

$$P(\vec{r}) = \langle P(c_2), \overline{P(c_2)} \cdot P(c_3), \dots, \prod_{i=2}^{L-1} \overline{P(c_i)} \cdot P(c_L) \rangle$$

As per the definition of PTD function:

$$F(\vec{r}) = D(\vec{r}) \cdot P(\vec{r})$$

Let $\vec{r}$ be a potential sequence with length L and $\vec{d} = \langle c_0, \vec{r} \rangle$ be a driving route derived from $\vec{r}$. The probability summation of $\vec{r}$ is the sum of all the dimensions in the probability vector and it is given as:

$$PE(\vec{r}) = P(c_1) + \overline{P(c_1)} \cdot P(c_2) + \cdots + \prod_{i=1}^{L-1} \overline{P(c_i)} \cdot P(c_L)$$

Firstly, the possible sequences of length L ranging from 1 to N, where N is the number of centroids, are considered. However, not all sequences are taken as potential candidate sequences. Before that, incremental pruning is done to remove non-optimal driving routes. The following parameters are computed initially [12]:

1. $\forall c \in C$, where C represents the potential pick-up points, i.e., centroids.
2. $F1(c) = 0$
3. $PE(c) = P(c)$

The following are done iteratively:

$F(c_1, c_2, \dots, c_L) = P(c_2) \cdot F(c_2, c_3, \dots, c_L) + D_{c1,c2} \cdot PE(c_2, c_3, \dots, c_L)$

$PE(c_1, c_2, ..., c_L) = P(c_1) + P(c_1) \cdot PE(c_2, ..., c_L)$

4.6.3 Incremental Pruning

Suppose there are two potential sequence vectors, $\vec{a}$ and $\vec{b}$, of an equal length and the same source. The rest of the points in both sequences should consist of the same points of any order. If $F(\vec{a}) < F(\vec{b})$, then all the driving routes with the postfix sub-sequence *b* cannot be an optimal driving route. Such sequences are pruned and are omitted from possible candidate sequences *incrementally*. The resulting set of sequences after pruning are further pruned using batch pruning [12].

4.6.4 Algorithm

Input: A set of potential sequences *R(L)* with length *L*

Output: A set of remaining sequence candidates *X(L)* with length *L*

1. For each *r* in *R(L)*:
    a. *c=source(r)*
    b. *X(c).append(r)*
    c. For each *q in X(c):*
        i. If (r!=q):
            1. *If F(q)<F(r):*
                a. *X(c) = X(c) – r*
            2. Else:
                a. *X(c) = X(c) – q*
2. *Return X*

4.6.5 Batch Pruning

Suppose there are two potential sequences, $\vec{a} = \langle c_s, c_{a_1}, \dots, c_{a_k} \rangle$ and $\vec{b} = \langle c_s, c_{b_1}, \dots, c_{b_k} \rangle$, with equal length and same source. If $F(\vec{a}) < F(\vec{b})$, $\vec{b}$ is not the optimal driving route. Thus, $\vec{b}$ should be pruned from the sequence candidates [12,13].

4.7 ONLINE PRUNING

4.7.1 Algorithm

Input: A set of sequence candidates *R*, position of cab *PoCab* (c_0) with L_{min} and L_{max}, where 1 <= L_{min} <= L_{max} <= *N*

Output: A set of optimal driving route(s), *Dmin*

1. Initialize *Dmin* and *Fmin* as null sets.
2. For *L* in L_{min} to L_{max},:
    a. For each *r* in *X(L)*:
        i. *c = source[r]*
        ii. *d = (c0,r)*
        iii. *F(d) = F(r)*(1-P(c))+Dco,c*PE(r)+D∞*(1-PE(r))*
        iv. *If (Dmin == null or F(d) == Fmin):*
            1. *Dmin = Dmin+d*
        v. Else:
            1. If F(d) < *Fmin*:
                a. *Dmin* = d
                b. *Fmin* = F(d)
3. Return *Dmin*

In online pruning, the initial position of the cab driver, *PoCab*, is given. Online pruning provides a driving route recommendation service for empty cabs at any dynamic position. A cab at position *PoCab* is denoted by c_0 and when this cab requests the recommendation service, the online pruning algorithm is used to find an optimal driving route from the remaining potential sequence candidates generated in offline pruning [12,13].

Two more inputs are given: L_{min} and L_{max}, where L_{min} is the minimum number of pick-up points the driver intends to travel and L_{max} is the maximum number of pick-up points. The potential driving routes are generated by connecting c_0 with the prescribed length *L*

with each potential sequence candidate generated in the offline pruning stage. Then, for each potential route, PTD is calculated.

The minimum PTD value along with the driving route(s) are selected and returned to the user [13].

TOOLS AND EVALUATION TECHNIQUES

5.1 DATASETS

The datasets which are taken consist of 512 files. Each file contains location traces of a unique cab driver. Each file has around 20,000 records and measures the places that the cab driver visited in 30 days in the city of San Francisco, California. There are four parameters to each record. The first two parameters include the latitude and longitude respectively. The third parameter is the operational status where 0 indicates that the cab is empty and 1 indicates that the cab has at least one passenger [29].

The last parameter is a Unix timestamp (Unix Epoch time). This is a method of describing a point in time, denoted as the number of seconds that have passed since 00:00:00 Coordinated Universal Time (UTC), January 1, 1970 minus the leap seconds (a one-second adjustment) that have taken place since then. The records are present according to the timestamp in descending order, i.e., according to the most recent location trace of the cab driver.

5.2 EVALUATION TECHNIQUES

i. Occupancy rate

The occupancy rate of each cab driver is evaluated by taking the average of the third parameter, operational status, for each driver. The condition is that the occupancy rate must be greater than or equal to 0.5.

ii. Number of travelling hours

The number of travelling hours of each cab driver must be greater or equal to 500 hours. The drivers who satisfy the occupancy rate and number of travelling hours constraints are labelled as successful cab drivers.

iii. Pick-up probability

The pick-up probability of each cluster is evaluated as the total number of pick-up events in a cluster divided by the total number of empty cabs travelling in the cluster.

iv. PTD function

This is the most important parameter in determining the optimal path. The PTD function of a cluster is the dot product of distance vector and probability vector of that cluster. This function is an optimal function that should be minimized.

$$F(\vec{r}) = D(\vec{r}) \cdot P(\vec{r})$$

Here, $\vec{r}$ is a sequence vector.

5.3 PACKAGES USED

i. pandas

pandas is an open source Python library that provides high-performance and user-friendly tools for implementing data structures and performing data analysis.

ii. sklearn.cluster.DBSCAN

This package is used to find core samples of high densities and form them into clusters in Python [25].

iii. NumPy

NumPy is the most fundamental Python package to perform scientific computations. This package also adds support for large and multi-dimensional lists along with high-level mathematical functions that can be used to perform on these lists.

iv. geopy.distance.great_circle

Geopy is a package that has the functions necessary to calculate the shortest possible distance between two geographical locations with respect to a spherical or curved route. This metric gives the approximate driving distance that drivers need to travel. The distance is also known as great circle distance or Haversine distance.

v. shapely.geometry.MultiPoint

Shapely is a licensed Python package typically used for manipulating and analyzing planar geometrical objects. In this case, the geometrical objects are the clusters. *MultiPoint* is a method that is used to extract the centroids of the clusters [24,28].

vi. webbrowser

webbrowser provides a high-level interface that allows contents from the Web to be displayed to the end user. In this project, this module is used to open Google Maps and

display the clusters according to the points' geographical locations as well as their respective centroids.

5.4 SOFTWARE

The following software were used for this project:

Operating System	Microsoft Windows 8 or above
Language	Oracle's Java 8 and Python 2.x or above
Software	Eclipse and Python IDLE

5.5 HARDWARE

The following hardware configurations were used in order to run the various software for this project:

CPU	Intel Core i5 Processor
Installed Memory (RAM)	3 GB minimum (4 GB recommended);
System Type	64-bit Operating System, x64 – based processor

RESULT ANALYSIS

The recommender system requires existing data in order to learn and adapt. Prior data simply refers to the routes previously taken by drivers. In case the driver is new to the city, then there will be no prior data from a driver's perspective. In that case, prior data can be downloaded from a server where data is from different and unique cab drivers. This data should be trained and later recommended to the driver. Therefore, one of the major challenges in implementing this system would be finding ample and sufficient amounts of true data. Many records of data will be needed to test the software.

The following is a representation of the clusters based on density. There are six clusters in total, including noise, which is represented in black. The *x*-axis represents the latitude whereas the *y*-axis represents the longitude.

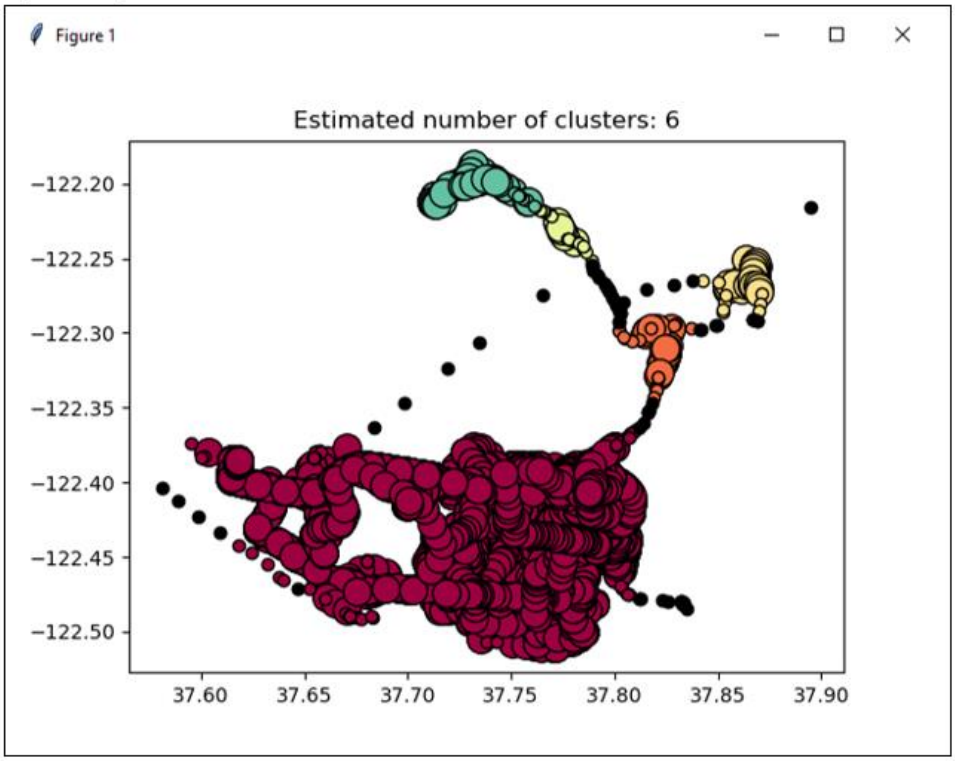

Figure 6.1: Representation of Clusters

The clusters based on density along with their centroids are displayed on Google Maps using Google Maps API. Here, centroids are displayed in large, black translucent circles.

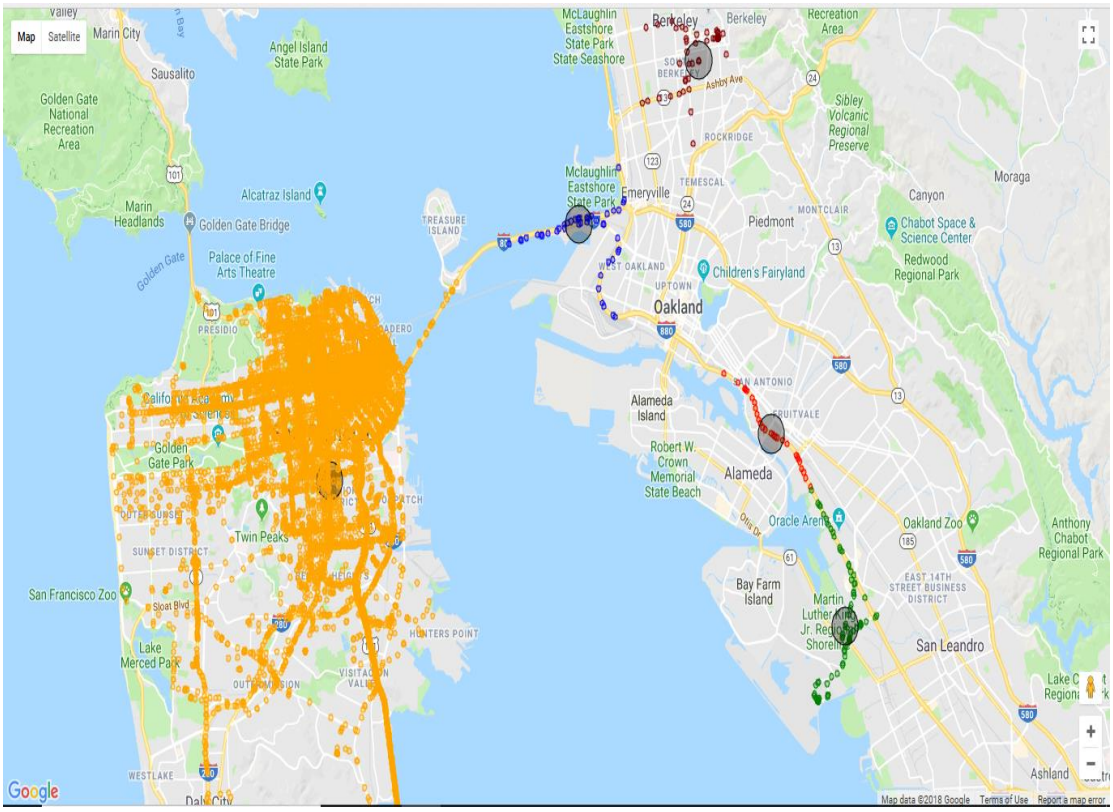

Figure 6.2: Representation of Clusters and Potential pick-up points on Google Maps

After implementing the proposed algorithm, the output is displayed in the form of a list. The output will be the recommended and optimal driving route(s) that is displayed to the cab driver.

CONCLUSION

The biggest key challenge while designing this application would be to account for processing power. Since only a *subset of N locations* is considered, countless number of routes exist from one location to another. Each route's information has to be saved into memory before determining the optimal solution. Therefore, main memory as well as secondary storage has to be large enough to accommodate this data and prevent possible heap errors.

In addition to optimization, there is another concern that should be taken care of while designing the application. Whether recommender systems are collaborative or content-based, some of the systems require personal information from a user. In return for that, recommendations are displayed to the users either according to their previous preferences or others' preferences (depending upon the type of system). Since the system must receive prior information about the users in order to understand them well enough to display recommendations, users must trust that the system will protect their information. Even though algorithms are understood by programmers, users are not presented with the same level of transparency. So, they do not exactly know how or why an item is being recommended to them. Thus, in order for a recommender system to work, it must first gain the user's trust. Therefore, privacy, or lack thereof, is one of the main concerns that should be considered while designing this system. Asking for too much personal and sensitive information from the user also comprises his/her security. Therefore, an ideal recommender system would be one where good recommendations are displayed while not requiring for users to give up too much information about themselves.

Location-based mobile recommender systems are used to improve sparsity in such prediction processes. Prior data of the user and data from other users are analyzed in a combinatorial manner to improve routing paths.

FUTURE SCOPE

Currently, the optimal route is calculated by probabilities at each pick-up location. This route can further be optimized by studying pick-up locations during certain time intervals of the day. Some locations may be populated during a time of the day while grow desolated during another time of the day.

In addition to setting timestamps, the optimal route can be further be enhanced (similar to the concept of genetic algorithms). Routes are typically affected by various parameters such as weather conditions, traffic congestions, previously traversed through routers, ferries, toll roads, highways, interstates, shortcuts, and so on. This is especially useful to drivers who are new to the city. Instead of aimlessly driving around and hoping for passengers, tweaking and slightly manipulating the parameters around will help drivers get around locations with maximized profit and hopefully in the minimum time as well.

In the near future, data will be generated at higher velocities, thus requiring better data managing scalability tools and data processing software. Besides just greater speeds, the amount of available data will also increase. This means that more data has to be analyzed to satisfy users' needs. Due to heavy computational power, applications might suffer with decreasing performance. Therefore, this project finds Big Data technologies a reliable solution to this problem, hence, further extending the scope of this project.

APPLICATIONS

1. One application of this system can be seen during school bus transportations. For example, while picking up children who live a few miles away from school, the application can help the bus driver by providing the optimal bus route.
2. Another application of this system can be related to home deliveries. For example, a pizza delivery boy could efficiently deliver all the pizza deliveries to various places in the city by using this approach.
3. Postal workers, courier boys, and all other delivery systems may also find this application useful.

REFERENCES

[1] "Recommender system," Wikipedia, 10-Oct-2017. [Online]. Available: https://en.wikipedia.org/wiki/Recommender_system

[2] S. Valencia, "An Introductory Recommender Systems Tutorial – AI Society – Medium," Medium, 09-Feb-2017. [Online]. Available: https://medium.com/ai-society/a-concise-recommender-systems-tutorial-fa40d5a9c0fa.

[3] "Introduction to approaches and algorithms," Recommender systems, Part 1, 12-Dec-2013. [Online]. Available: http://ibm.com/developerworks/library/os-recommender1/#fig1.

[4] "Collaborative Filtering," Recommender Systems, 23-Jan-2012. [Online]. Available: http://recommender-systems.org/collaborative-filtering/.

[5] "Content-based Filtering," Recommender Systems, 24-Jan-2012. [Online]. Available: http://recommender-systems.org/content-based-filtering/.

[6] F. Isinkaye, Y. Folajimi, and B. Ojokoh, "Recommendation systems: Principles, methods and evaluation," Egyptian Informatics Journal, vol. 16, no. 3, pp. 261–273, 2015.

[7] Recommendation Systems "http://infolab.stanford.edu/~ullman/mmds/ch9.pdf"

[8] "Intro to Recommender Systems - University of Minnesota," *Coursera*. [Online]. Available: https://www.coursera.org/learn/recommender-systems-introduction/lecture/0ExG1/intro-to-recommender-systems. [Accessed: 06-Nov-2017].

[9] T. Point, "SDLC Spiral Model," www.tutorialspoint.com, 15-Aug-2017. [Online]. Available: https://www.tutorialspoint.com/sdlc/sdlc_spiral_model.htm. [Accessed: 06-Nov-2017].

[10] “Information Filtering,” Recommender Systems, 23-Jan-2012. [Online]. Available: http://recommender-systems.org/information-filtering/. [Accessed: 06-Nov-2017].

[11] Software Engineering Paradigms And Models Information Technology Essay. [Online]. Available: https://www.uniassignment.com/essay-samples/information-technology/software-engineering-paradigms-and-models-information-technology-essay.php. [Accessed: 06-Nov-2017].

[12] J. Huang, X. Huangfu, H. Sun, H. Li, P. Zhao, H. Cheng, and Q. Song, “Backward Path Growth for Efficient Mobile Sequential Recommendation,” *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 1, pp. 46–60, 2015.

[13] Ge, Y., Xiong, H., Tuzhilin, A., Xiao, K., Gruteser, M. and Pazzani, M. (n.d.). An Energy-Efficient Mobile Recommender System. pp.1-9.

[14] Anon, (2018). [online] Available at: https://www.quora.com/Which-clustering-algorithm-is-best-suited-for-geo-spatial-data-and-why [Accessed 26 Apr. 2018].

[15] Busa, N. (2018). *Clustering geolocated data using Spark and DBSCAN*. [online] O'Reilly Media. Available at: https://www.oreilly.com/ideas/clustering-geolocated-data-using-spark-and-dbscan [Accessed 26 Apr. 2018].

[16] data, D. (2018). *DBSCAN for clustering of geographic location data*. [online] Stackoverflow.com. Available at: https://stackoverflow.com/questions/34579213/dbscan-for-clustering-of-geographic-location-data [Accessed 26 Apr. 2018].

[17] Geoff Boeing. (2018). *Clustering to Reduce Spatial Data Set Size - Geoff Boeing*. [online] Available at: http://geoffboeing.com/2014/08/clustering-to-reduce-spatial-data-set-size/ [Accessed 26 Apr. 2018].

[18] Gis.stackexchange.com. (2018). *Geographic Information Systems Stack Exchange*. [online] Available at: https://gis.stackexchange.com/ [Accessed 26 Apr. 2018].

[19] GitHub. (2018). *gboeing/urban-data-science*. [online] Available at: https://github.com/gboeing/urban-data-science/blob/master/15-Spatial-Cluster-Analysis/cluster-analysis.ipynb [Accessed 26 Apr. 2018].

[20] GitHub. (2018). *vgm64/gmplot*. [online] Available at: https://github.com/vgm64/gmplot [Accessed 26 Apr. 2018].

[21] Kong, Q., Kong, Q. and profile, V. (2018). *Clustering with DBSCAN*. [online] Qingkaikong.blogspot.in. Available at: http://qingkaikong.blogspot.in/2016/08/clustering-with-dbscan.html [Accessed 26 Apr. 2018].

[22] Matplotlib.org. (2018). *matplotlib.pyplot.plot — Matplotlib 2.2.2 documentation*. [online] Available at: https://matplotlib.org/api/_as_gen/matplotlib.pyplot.plot.html [Accessed 26 Apr. 2018].

[23] Mccormickml.com. (2018). *DBSCAN Clustering · Chris McCormick*. [online] Available at: http://mccormickml.com/2016/11/08/dbscan-clustering/ [Accessed 26 Apr. 2018].

[24] Points), P. (2018). *Problems with Geometry collections. (Fiona: write output of Multipoints and Points)*. [online] Gis.stackexchange.com. Available at: https://gis.stackexchange.com/questions/140153/problems-with-geometry-collections-fiona-write-output-of-multipoints-and-poin [Accessed 26 Apr. 2018].

[25] Scikit-learn.org. (2018). *Demo of DBSCAN clustering algorithm — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/auto_examples/cluster/plot_dbscan.html#sphx-glr-auto-examples-cluster-plot-dbscan-py [Accessed 26 Apr. 2018].

[26] Scikit-learn.org. (2018). *1.6. Nearest Neighbors — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/neighbors.html#ball-tree [Accessed 26 Apr. 2018].

[27] Scikit-learn.org. (2018). *2.3. Clustering — scikit-learn 0.19.1 documentation*. [online] Available at: http://scikit-learn.org/stable/modules/clustering.html#dbscan [Accessed 26 Apr. 2018].

[28] Toblerity.org. (2018). *The Shapely User Manual — Shapely 1.2 and 1.3 documentation*. [online] Available at: https://toblerity.org/shapely/manual.html [Accessed 26 Apr. 2018].

[29] http://cabspotting.org/.