

Designing A Recommendation System For Crowdsourced Road Monitoring Applications

By

RUSAN AHSAN

Class Roll: 001910503024

Master in Computer Application (MCA)

Computer Science & Engineering Department

Jadavpur University

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

CERTIFICATE OF RECOMMENDATION

This is to certify that the project/thesis entitled “DESIGNING A RECOMMENDATION SYSTEM FOR CROWDSOURCED ROAD MONITORING APPLICATIONS” has been satisfactorily completed under my guidance and supervision by **RUSAN AHSAN** (University Registration No.:136912 of 2016-17, Examination Roll No.: MCA226023, Class Roll No.: 001910503024). I hereby recommend that the project be accepted in partial fulfilment of the requirement for the Degree of Master of Computer Application, Department of Computer Science and Engineering in Faculty of Engineering and Technology, Jadavpur University, Kolkata for the academic year 2021-2022.

Dr. Chandreyee Chowdhury (Thesis Supervisor)
Associate Professor
Department of Computer Science and
EngineeringJadavpur University, Kolkata-
700032

Countersigned

Prof. Anupam Sinha
Head, Department of Computer Science and
Engineering,Jadavpur University, Kolkata-700032.

Prof. Chandan Majumdar
Dean, Faculty of Engineering and
Technology,Jadavpur University, Kolkata-
700032.

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

CERTIFICATE OF APPROVAL

This is to certify that the project/thesis entitled “DESIGNING A RECOMMENDATION SYSTEM FOR CROWDSOURCED ROAD MONITORING APPLICATIONS” is a bona fide record of work carried out by RUSAN AHSAN in partial fulfilment of the requirements for the award of the degree of Master of Computer Application in the Department of Computer Science and Engineering, Jadavpur University during the period of January 2022 to July 2022. It is understood that by this approval the undersigned do not necessarily endorse or approve any statement made, opinion expressed or conclusion drawn there in but approve the thesis only for the purpose for which it has been submitted.

Signature of Examiner

Date:

Signature of Supervisor

Date:

FACULTY OF ENGINEERING AND TECHNOLOGY
JADAVPUR UNIVERSITY

DECLARATION OF ORIGINALITY AND COMPLIANCE OF ACADEMIC
ETHICS

I hereby declare that this project/thesis entitled “DESIGNING A RECOMMENDATION SYSTEM FOR CROWDSOURCED ROAD MONITORING APPLICATIONS” contains literature survey and original research work by the undersigned candidate, as part of his Degree of Master of Computer Application. All information in this document has been obtained and presented in accordance with academic rules and ethical conduct. I also declare that, as required by these rules and conduct, I have fully cited and referenced all materials and results that are not original to this work.

Name: RUSAN AHSAN

University Registration No. : 136912 of 2016-17

Examination Roll No. : MCA226023

Project/Thesis Title: DESIGNING A RECOMMENDATION SYSTEM FOR
CROWDSOURCED ROAD MONITORING APPLICATIONS

Signature

Date:

ACKNOWLEDGEMENT

I express my deep sense of gratitude to my respected and learned guide, Dr. CHANDREYE CHOWDHURY for her valuable help and guidance. I am grateful to her for the encouragement she has given me in completing the project.

RUSAN AHSAN
ROLL NO- 001910503024

Contents

1	Introduction	4
1.1	Motivation	6
1.2	Contribution	6
1.3	Organization of the report	6
1.3.1	Related work	6
1.3.2	Proposed methodology	6
1.3.3	Implementation details	7
1.3.4	Experimental results	7
1.3.5	Conclusion	7
2	Related Work	8
2.1	CRSM: a practical crowdsourcing-based road surface monitoring system ^[1]	8
2.2	Crowdsourced Exploration of Security Configurations ^[2]	8
2.3	Prediction and Analysis of Hotel Ratings from Crowd-sourced Data ^[3]	8
2.4	A web-based pervasive recommendation system for mobile tourist guides ^[4]	8
2.5	Enhancing Mobile App User Understanding and Marketing With Heterogeneous Crowdsourced Data: A Review ^[5]	9
2.6	Crowdsourcing for bioinformatics ^[6]	9
3	Proposed Methodology	10
3.1	Task Definition	10
3.2	System Framework	10
3.3	Data Acquisition and preprocessing	11
3.4	Learning and Training Model	13
3.4.1	Training of ML algorithm(off-server)	13
3.4.2	Review validation by mean difference(off-server)	14
3.4.3	Instant Review Validation(on-server)	15
3.5	Recommendation phase	16
4	Implementation details	17
4.1	Technologies used ^[7]	17
4.2	Sign up and Sign in configuration	18
4.2.1	JSON Web Token ^[8] (JWT)	19
4.2.2	Bcrypt.js ^[9]	20
4.3	Home Page details	21
4.3.1	Search prediction using TomTom API ^[10]	21
4.3.2	Map Rendering With Mapquest ^[11]	22
4.3.3	Route selection	23
4.3.4	Review submission	24
4.4	Review training and validation	27
4.4.1	Training offline	27
4.4.2	Predicting reviews realtime	27
4.4.3	Offline review validation by mean difference	28
4.4.4	Instant Review Validation	28
4.5	React-Bootstrap ^[12]	29
4.6	Security Features	30
4.7	Deployment	31
5	Experimental Results	34
5.1	Experimental setup	34
5.2	Evaluation Metric	34
5.2.1	Random Forest Regression ^[13]	34
5.2.2	Simple Linear Regression ^[14]	35
5.2.3	Decision Tree Regression ^[15]	35
5.2.4	Polynomial Regression ^[16]	35
5.2.5	Exponential Regression ^[17]	35
5.2.6	Theil Sen Regression ^[17]	36
5.2.7	Multivariate Linear Regression ^[18]	36
5.3	Result Analysis	36
5.3.1	Real vs synthetic dataset	42
5.4	Results of working of route recommendation	44
5.4.1	Setup for route recommendation	44
5.4.2	Review submission	44
5.4.3	Route recommendation results	47

5.4.4	Results and discussion	49
6	Conclusion	50
6.1	Scope of improvement	50
6.2	Future aspects	50

1 Introduction

Crowdsourcing^{[19][20]} entails a large number of dispersed people providing or creating goods or services for payment or as volunteers, such as ideas, votes, microtasks, and funds. In his essay "The Rise of Crowdsourcing," author Jeff Howe of Wired magazine created the phrase crowdsourcing, which is a combination of crowds and outsourcing. Crowdsourcing is the process of gathering information, opinions, or work from a large number of individuals using the Internet. It allows businesses to save time and money while gaining access to people with a variety of skills and perspectives from all around the world. Crowdfunding wants money to help individuals, charities, or start-up businesses, whereas crowdsourcing seeks knowledge or work product. Cost reductions, timeliness, and the chance to work with people who have abilities that an in-house team may lack are all advantages of crowdsourcing.

Companies can use *crowdsourcing*^{[21][22]} to get brilliant ideas for a new product or service, or a fresh way to solve a difficult problem. This not only aids in problem-solving, but it also makes groups feel more connected to businesses and organisations. In terms of marketing, brand visibility, and customer loyalty, forming this community of contributors may be extremely beneficial. Crowdsourcing brings together communities around a common project or cause. It has Lower costs than hiring people to solve the actual problem. It is an Efficient way of solving time-intensive problems. We can also find unexpected solutions to tough problems by crowdsourcing. It leads to deeper engagement by communities, who resonate and build loyalty to the product or solution. crowdsourcing can be a great source of marketing buzz.

Crowdsourcing isn't a silver bullet for businesses looking to reduce their workload while pursuing the next big thing. Many times, someone will have to filter through all of the concepts being pitched, fundraising targets in all-or-nothing funding platforms can fall short, and finding and engaging the correct crowd can be challenging. Results can be easily skewed based on the crowd being sourced. It has lack of confidentiality or ownership on an idea. Crowdsourcing projects aren't as easy to manage and control as traditional internal projects. It has Potential to miss the best ideas, talent, or direction and fall short of the goal or purpose. Crowdsourcing always has a risk of inconsistent outcomes.

There are various applications that we use in our daily routine that has crowdsourcing as their main agenda for performing their relevant operations and tasks.

Amazon's Mechanical Turk (MTurk)^[23] is a crowdsourcing marketplace that businesses can use to outsource parts of their jobs, everything from data validation to research to content moderation. Anyone can sign up through their Amazon account to be a Mechanical Turk Worker. *StackOverflow*^[24] is a community-based space to find and contribute answers to technical challenges, and one of the most popular websites in the world. People can vote for correct solution to the problems and the solution with large number of votes is displayed with a green validation tick mark which marks the authenticity of the solution and is often proven to be correct. *Glassdoor*^[25] is a website where current and former employees anonymously review

companies. Glassdoor also allows users to anonymously submit and view salaries as well as search and apply for jobs on its platform. *Quora*^[26] is a Q&A platform that empowers people to share and grow the world's knowledge. It is used to connect with people who contribute unique insights and quality answers. *Googlemaps*^[27] is one of the most extensively used crowdsourcing app. Google places helps to rate different places like restaurants, organizations, public buildings, market places, etc. Sometimes based on the user's location google maps sends notifications to the user whether they want to share their experience about their visited place. It can be used to judge and choose between different alternatives of places and their qualities. We can give both comments and ratings which can be viewed publicly. *Wikipedia*^[28] is a free online encyclopedia, created and edited by volunteers around the world and hosted by the Wikimedia Foundation. Almost, every thing in the world which has relevance can be found in some article in wikipedia and hence one of most extensive crowdsourcing app for sharing knowledge around the globe.

Crowdsourcing can bring a revolution in road monitoring system. There are various applications of crowdsourcing in road monitoring system. The crowdsourced traffic data can provide supplementary information for incidents (like potholes, traffic light faults, missing road signs) already reported through other sources and it can contribute to earlier detection of incidents, which can lead to faster response and clearance time. Road condition is also one of the most requested data by users to choose between roads. In maximum case road condition matters and not the distance. Often traffic congestion is one of the most irritating and usual incident in office hours. There are crowdsourcing applications than tracks real time traffic congestions. Crowd sourcing platforms can be used to identify busy places or unvisited places which can help us to see which areas are good for women safety.

There are a lot of challenges faced by organization who tries to use crowdsourcing specially in road monitoring. Here are few such examples. The biggest challenge of crowd sourcing is data collection. Often user do not have incentives to give data on the platforms to help other. Certain organization gives monetary benefits for their reviews and can often suffer setbacks specially the new organizations. Users fail to trust a non-reputed crowd sourcing platform in respect to the safety of their data majorly. Also, the users cannot fully trust the data shown by the platform, which can lead to low number of users and ultimately hamper and diminish the growth of the companies. Often there are huge amount of spam or fake reviews of the users. Users generally gives such reviews to create a bias and tender the state of the apps in their favour. Thus, this can lead to a wrong idea of many users if they indeed trust the application. users can give biased reviews. So example suppose a user has a tendency to give only positive review; so he will give positive review for rather not so good places or incidents. Similarly, a user who is pessimistic can give bad reviews for a lot of things without properly experiencing it. Also, a person may be biased about certain places he likes and can give positive/negative reviews about places near it.

1.1 Motivation

There are lot of apps that use crowdsourcing like hotel websites, mobile tourist guides, bioinformatics as discussed in 2 but there are very few apps that utilizes crowdsourced data to improve road scenarios. We can see a lot of accidents that occurs on a daily basis due to poor road condition. If the people knew before hand which road is more suitable for them to travel, these kinds of accidents may be avoided. Also, if we have an app that has data based on user's review that will send a strong message to the government about different conditions of the roads and will help them to work and improve on road condition.

There are also factors like women safety, which is a key factor these days in rural and urban areas. An app which can suggest, different routes according to the women safety is a key factor that we need to build such motivation system to enhance their safety. There are also other fields where we lack crowdsourced knowledge are traffic congestion, availability of public toilets, wifi or gas stations. We can save a lot of our time if we have knowledge about these conditions from past user's experience. In 2.1 we can see that some of apps are available that tries to detect potholes in the roads but there is not a single app that considers different review factors of the roads, validate user's review and then recommend a route to the user.

1.2 Contribution

We have designed a crowdsourced system on road monitoring system which has multiple steps of data validation and a recommendation system which will recommend routes to the users based on their preference. We are also storing our data in NOSQL cloud database and used different server for data training so that our data validation process is fast. Our data validation includes ML regression algorithms, review reduction by mean difference, validating reviews based on user's location and spam review detection. Our recommendation is based on storing data of path segments independently and accumulating the segments required to recommend the path required by the user.

1.3 Organization of the report

1.3.1 *Related work*

It consists of some descriptions, references etc. of some previous works done on crowdsourced data.

1.3.2 *Proposed methodology*

This portion of the thesis consists of an theoretical overview of it and what are the modules. The thesis mainly consists of two modules namely the Validation Module and the Route Recommendation module. We get a clear concept of the system model and the dataflow. In the validation module, it is explained how we have used a Machine Learning Regression Model to predict user's data and thereby calculating the percentage of error of the input review to judge it's authenticity. We also introduced

instant validators based on user's location and the nature of review(whether spam or not). In the recommendation phase it is show how maintaining review against every path segment helps us recommend routes to the users.

1.3.3 Implementation details

In this section, each and every working details of the website is discussed(i.e, the implementation of the algorithms and methods proposed in "Proposed methodology"). The MERN achitecture is explained along with the http calls and interaction made by the frontend with the server and finally the data being stored in the cloud database. We have also shown a glimpse of how we render the map in our website using Mapquest api and Tomtom api for search predictions. Then, we discussed Bootstrap classes and responsiveness of the website. Finally, we discussed deployment of the app using heroku. We also explained in details the sign in and sign up process and how the security of the app is implemented using JWT, bcrypt.js, CORS and dotenv. We took extra care to protect the api keys of Mapquest and Tomtom.

1.3.4 Experimental results

In this portion of the thesis, we discuss how the app handles different types of data(both real and synthetic). In addition to the Machine Learning Regression algorithm(Random Forest Regression) we used six more ML Regressor algorithm with both real and synthetic data. For each visualization we have created 3 multibar graph comparisons between the various ML algorithms for the three routes available between Jadavpur-Howrah (Synthetic data). Then we created 2 multibar graph for the same ML algorithms for the routes between Priyo Builders and Sealdah railway station. We also gave a working example of the recommendation phase. We have shown how the data is organized in the database when user submits a review by the use of tables and we used these tables to recommend the route according to user's preference.

1.3.5 Conclusion

It comprises of how Crowdsourcing can be used not only for monitoring roads but the same data validation and optimization framework can be used for other crowd sourcing applications which use users' ratings and feedbacks. We discuss different improvements which could be done in the app and also the future of crowdsourcing.

2 Related Work

With growing population and technical advancement researchers are trying to utilize the crowd to solve various complex problems. Till present day many Crowdsourcing platforms have come up like MTurk, Upwork etc. Researchers have also proposed certain algorithms for validation of data in crowdsourcing system. Let us have a look on some of the existing works relating data validation in crowdsourced system.

2.1 CRSM: a practical crowdsourcing-based road surface monitoring system^[1]

Detecting road potholes and road roughness levels is key to road condition monitoring, which impacts transport safety and driving comfort. In [1] Chen et al. proposed a crowdsourcing-based road surface monitoring system, simply called CRSM. Using inexpensive accelerometers and GPS sensors, CRSM can accurately identify road potholes and gauge the degree of road roughness. The servers gather data from several vehicles and it operates with 90 percent accuracy and with practically zero false alarms. A unique light-weight data mining technique is suggested to identify road surface events and broadcast probable pothole information to a central server in light of the high cost of onboard storage and wireless transmission.

2.2 Crowdsourced Exploration of Security Configurations^[2]

Today's smartphone apps ask users for permission to access a wide range of private resources, which they must fully approve in order to install the app. Users are unable to decide which permissions are necessary in an educated manner. [2] Ismail et al. advocated an efficient 'lattice-based' crowd-management strategy to explore the space of permissions sets. A study on the permission sets for the Instagram app, which involved 26 participants, came to the conclusion that it was an effective crowd-management technique and demonstrated how usability ratings for different user types could be predicted based on how they would use it.

2.3 Prediction and Analysis of Hotel Ratings from Crowd-sourced Data^[3]

The tourism industry and consumers now rely heavily on crowdsourcing as a source of information. Large amounts of data are shared daily amongst stakeholders through searches, postings, shares, reviews, and ratings. [3] Leal et al. in this paper presents a tourist-centred analysis of crowd-sourced hotel information collected from the Expedia platform. To forecast trends and patterns that are important to tourists and businesses, the analysis uses data mining approaches. First, correlation and multiple linear regression are used to lessen the dimensionality of the crowdsourced data. The Alternating Least Squares technique is then used to forecast hotel ratings.

2.4 A web-based pervasive recommendation system for mobile tourist guides^[4]

In order to help travellers choose where to travel, there is a significant need for personalised services based on travel recommender systems in mobile tourist guides.

By addressing a crucial part of personalisation, these technologies lighten the user's load of information. In this paper, [4] Gaval et al. extends the idea of trip recommender systems by using collaborative filtering methods and contextual data (such as the user's present location, time, weather conditions, etc.) to produce better recommendations in ubiquitous situations. In order to provide mobile users with a precise location and an affordable experience while publishing reviews regarding points of interest, a wireless sensor network (WSN) is set up around popular tourist destinations (POI). In order to distinguish between users who review POIs using the mobile tourist guide application while on-site and others who rate POIs using the Internet away from the POI, "context-aware rating" is utilised to weight user ratings uploaded through WSN infrastructures more heavily.

2.5 Enhancing Mobile App User Understanding and Marketing With Heterogeneous Crowdsourced Data: A Review^[5]

The mobile app market has been surging in recent years. It has some key differentiating characteristics which make it different from traditional markets. To enhance mobile app development and marketing, it is important to study the key research challenges such as app user profiling, usage pattern understanding, popularity prediction, requirement, feedback mining, and so on. In this paper [5] Guo et al. reviews CrowdApp, a research field that leverages heterogeneous crowdsourced data for mobile app user understanding and marketing. First, the potential offered by the CrowdApp are characterised, and then the main research problems and cutting-edge solutions are presented. We also talk about the CrowdApp's current problems and potential future developments.

2.6 Crowdsourcing for bioinformatics^[6]

There are several issues in bioinformatics that call for human intervention. Human contribution is beneficial for tasks like protein structure determination, genomic annotation, picture analysis, and knowledge base population. [6] Good et al. provides a framework for understanding and applying several different types of crowdsourcing. The framework takes into account two major categories: systems for completing high-difficulty "megatasks" and systems for handling large-volume "microtasks." System types such as using volunteer labour, games with a goal, microtask markets, and open innovation competitions are covered within these classes. A handbook for connecting problems to crowdsourcing solutions that outlines the advantages and disadvantages of various strategies concludes each system type with successful bioinformatics examples.

3 Proposed Methodology

3.1 Task Definition

The users submits their ratings on a particular route from a source to a destination out of the n routes available. The ratings contains an array of values $[x_1, x_2, x_3, \dots, x_k]$ based on some review factors(e.g- women safety, traffic congestion) and $1 \leq x_i \leq P$ where $1 \leq i \leq k$ and $P \in \mathbb{N}$.

The users get a recommended route out of the n routes of a particular source and destination based on the user's choice on review factors.

3.2 System Framework

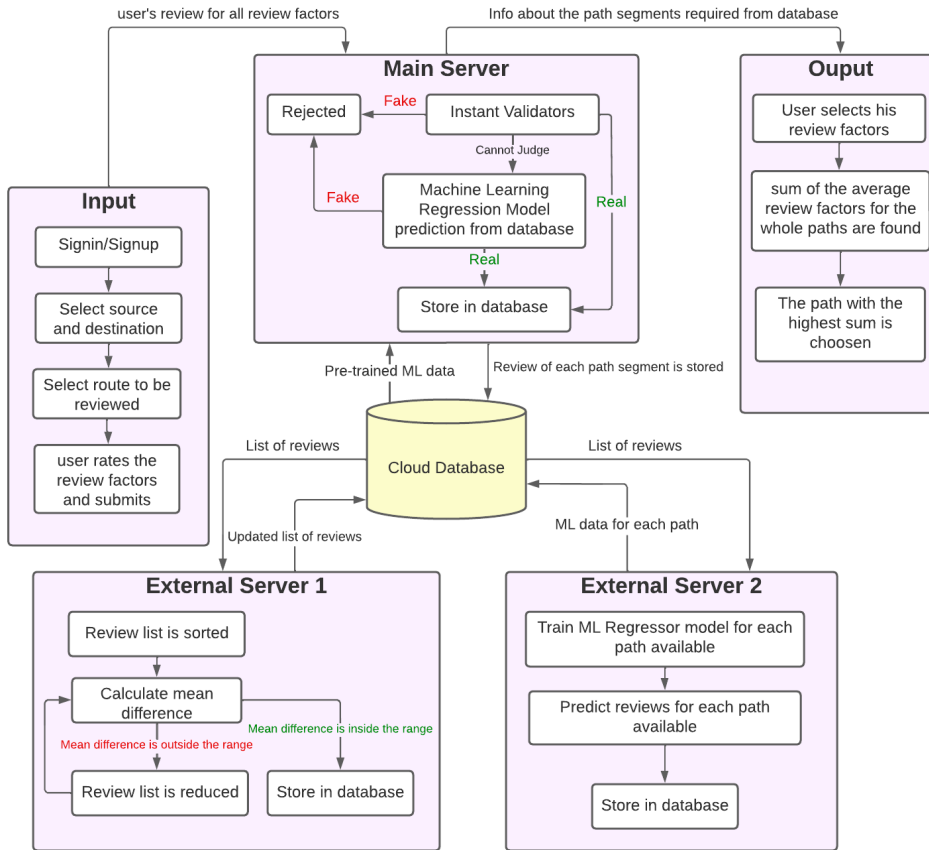


Fig. 1 System model workflow

The system is based on a 3-tier architecture where the first tier consists of the clients who use the app by laptops, mobile phones, tablets, etc , the second tier consists of the servers (one main server and two external servers) which does our

data validation and recommendation part and the third tier consists of the cloud database where all the data and tables of the apps are stored including training data and recommendation data. The main phases of the proposed workflow is as follows:

- Data acquisition and preprocessing
- Data validation
- Recommendation phase

The data validation consists of three sub-phases that includes ML based data validation of reviews (off-server), statistical data validation of reviews (off-server), instant validation of reviews (on-server).

This work is detailed in the following subsections as follows:

3.3 Data Acquisition and preprocessing

Initially when our database is empty data collection is done in two ways- one is collecting data from a trusted group or organization and another generation of reviews(synthetic dataset) for our application. Since collecting data about each and every routes is a tedious task, the project mainly uses synthetic dataset for the authentication of the results and algorithm. As we go on gradually collecting more and more data, we would be reliant more on the actual real world dataset.

Review set is generated from a dataset X whose reviews range from $[1, p]$ (matching our actual dataset) and r is the likely rating of the dataset (Maximum of the reviews are r but not all); $1 \leq r \leq p$ and $p \in \mathbb{N}$. Suppose, the review factors are $[x_1, x_2, \dots, x_k]$ and the likely review for these review factors are r_1, r_2, \dots, r_k , $r_i \in [1, p] \forall i \in [1, k]$, we can introduce injective functions $f_1(p), f_2(p), \dots, f_k(p)$ such that $f_1(p)$ maps r to $r_1, f_2(p)$ maps r to r_2, \dots and $f_k(p)$ maps r to r_k . All the elements which are not r are injectively mapped to other elements.

So, we can divide our dataset X into k parts, each of the part can be used to generate the k^{th} review factor rating.

Firstly, the user selects their desire source and destination. Then they are shown the set of available routes between the particular source and destination. The user may select any one of them and decide to submit their feedback(review). The user rates the route on all the review factors and the it is stored in form of an array say $[x_1, x_2, x_3, \dots, x_k]$.

Then when the user submits their reviews, firstly it is verified whether the user had rated against all the review factors and the review lies in the legitimate range. If there is an issue, the user may again try to submit his reviews, else the server receives the user's reviews and run some instant validators on it. Instant validators includes location based validation and spam review check. Location based validation is checking whether the user's current location is near to their specified source or destination. If the distance between the user's current location and source or user's current location and destination lies within some fixed small distance value, then

the review is likely to be genuine. More mathematically the expression would be,

$$\text{distance}(U, S) < \epsilon \quad \text{or} \quad \text{distance}(U, D) < \epsilon \quad \Rightarrow \text{Genuine review}$$

where U is the user's current location, S is the source selected by the user and D is the destination selected by the user and ϵ is a fixed small positive distance.

The next step for the server is to check whether the review is likely to be spam or not. If the same user has submitted their review on a particular route more than once in small intervals then the review is likely to be a spam review. More mathematically we have,

$$t > T \quad \Rightarrow \text{Fake review}$$

where t is the last time user submitted the particular route and T is a fixed amount of time.

So, the server basically sends three kinds of response- the review is likely to be fake(when its a spam review), the review is likely to be genuine(when the user's location is close to either, the source or the destination) and when the review is undecided whether fake or genuine(can happen when the review is not spam and the user's location is not close enough to either, the source or the destination). If the review is likely to be genuine, the review are sent to the server to be stored in the database and if the review is likely to be fake then the review is rejected. However if the review is undecided whether fake or genuine, the reviews are sent again to the server to run some machine learning regression algorithms.

Let the ML regression algorithm be M. M is pre-trained on the routes with respect to their review factors ratings. Hence,

$M = \text{regressor}(p_i)$ where $p_i = [x_1, x_2, \dots, x_k]$ and i is the generalized review number.

Let $\text{regressor}(p_i) = [y_1, y_2, \dots, y_k]$

Total Error = $|y_1 - x_1| + |y_2 - x_2| + \dots + |y_k - x_k|$

Total Possible Error = $k * P$, where P is the highest rating possible.

$$\begin{aligned} \text{Error percentage} &= \frac{\text{Total Error}}{\text{Total Possible Error}} \\ \Rightarrow \text{Errorpercentage} &= \frac{|y_1 - x_1| + |y_2 - x_2| + \dots + |y_k - x_k|}{k * P} \end{aligned} \quad (1)$$

If Error percentage $< \epsilon$ (ϵ = Error tolerance percentage) then the reviews are accepted and sent to database for storage else the review is rejected.

Fig. 2 shows all the above process in diagrammatic manner.

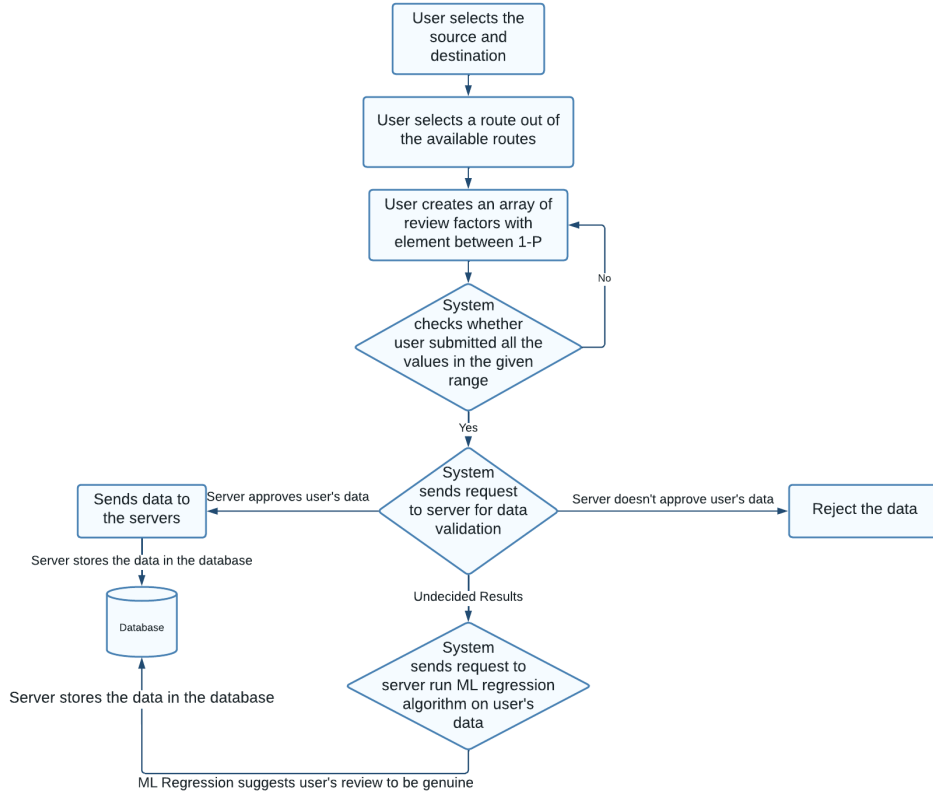


Fig. 2 User's submission of review

3.4 Learning and Training Model

Training is often time consuming and costly process so it is better to do it off-server using some other server(performance of the main server is not disturbed). Then the result of the training can stored in the cloud database and from then on we can the results being reflected on the actual server itself.

3.4.1 Training of ML algorithm(off-server)

As stated early this process is completely off-server. So, we can deploy the extensive and time consuming training process in some high speed processor machine and update the results in the database. First of all for every unique route(routes which have different array of latitudes and longitudes are called unique) in the database is given a UNID(Unique Natural Identity), i.e , a unique natural number. So, every route which has a review can be identified by this number. The ML regressor Model is trained on UNID as the input and the review factors array $R_i = [x_{i1}, x_{i2}, \dots, x_{ik}]$ where R_i is the i^{th} review factors array. After training we run the ML regressor for each path present in our database and store the result in a seperate table. So, when an user tries to submits a review, the training data from the table is checked and verified whether the user has submitted likely genuine review or not, which is expained in details in ??

In fig-3 we have shown how we train the ML algorithm and store the results in the database for quick access later.

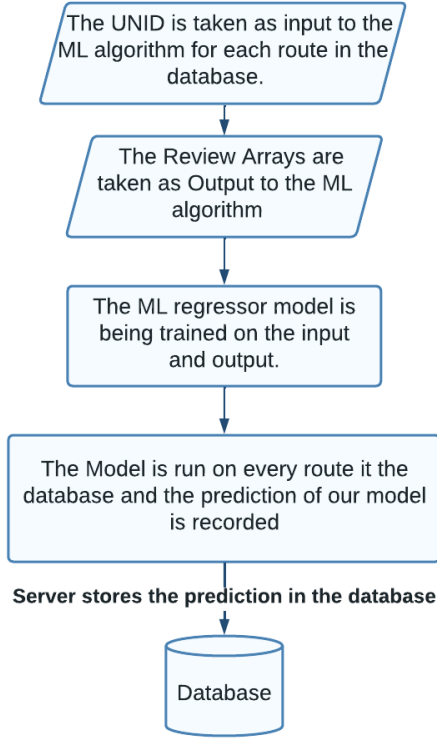


Fig. 3 Training of the Machine Learning Model

3.4.2 Review validation by mean difference(off-server)

In our review system certain review factors may correlate to another in a positive way which means that the review factors' ratings are in close proximity. On the other hand there may be review factors which correlate in a negative way which means that the review factor's ratings mostly have significant difference in them.

Let RF1 correlates positively with RF2 ($RF \Rightarrow$ Review Factor). Then, the array of reviews is taken from the database and sorted according to the absolute difference between RF1 and RF2 in ascending order. Now the average difference is calculated between RF1 and RF2. If this is within some small value ϵ then no change is done, else if the value is greater than ϵ we delete a review from end of the array (the review which has the greater difference between RF1 and RF2 is deleted first and thus reducing the average difference between them).

Similarly, if RF1 and RF2 is negatively correlated, then we sort the array of reviews according to the absolute difference between RF1 and RF2 but in descending order. If this is greater than some fixed value ϵ then no change is done, else if the value is lesser than ϵ we delete a review from end of the array (the review which has

the lesser difference between RF1 and RF2 is deleted first and thus increasing the average difference between them.

The review list is then copied in the database thus eliminating the probable faulty reviews. In Fig-4, it is shown how use mean difference to reduce faulty reviews in the database using an external server.

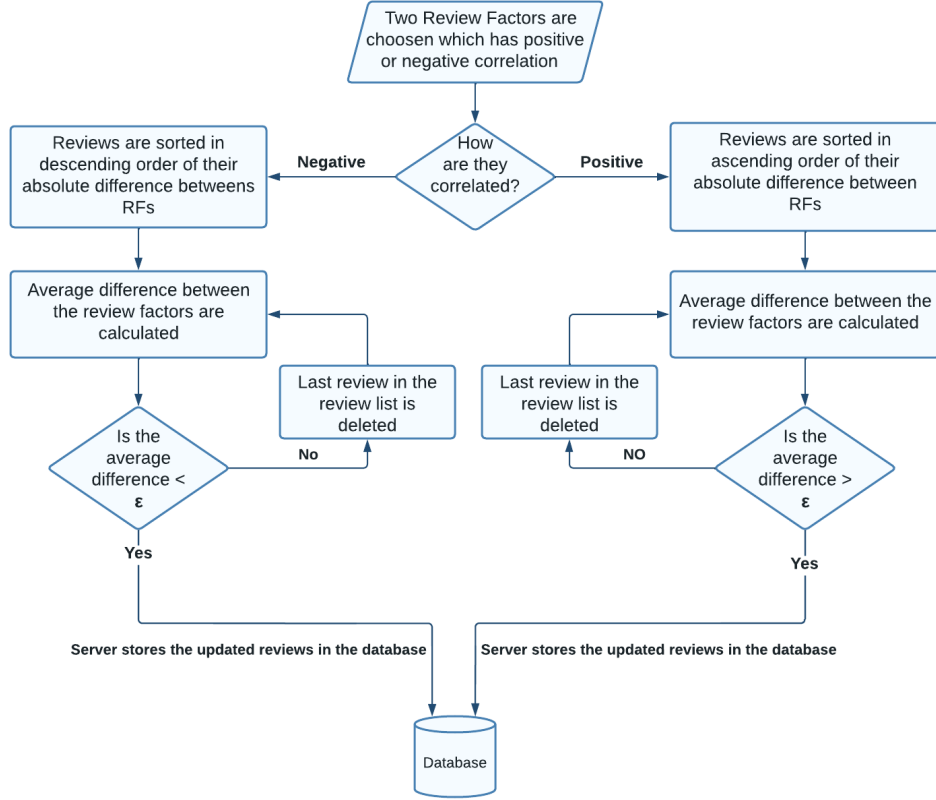


Fig. 4 Review validation

3.4.3 Instant Review Validation(on-server)

Unlike the training of ML algorithms and review validation by mean difference, this validation can be done instantly as the user is submitting the review. This consists of two parts- to check for authentic reviews using user's location and the other is to find whether user is sending spam reviews or not. For the first part, the user's current location is compared with the source's and destination's gps coordinates and finds the distance between them using *Haversine's*^[29] formula. If any one of source or destination is within acceptable distance. Then the review is validated to true other we remain undecided. For the second part, we find the last time when the user submitted the same route's review. If user has submitted a review not very long ago and is trying to submit a review again for the same route, then it will be

considered as a spam review and will be rejected. If it is not a spam review, then we remain undecided about it's authenticity.

3.5 Recommendation phase

When the user searches for a route between a particular source and destination with respect to the some particular review factors x_i , $i \in [1, k]$. They should get the route with highest reviews for those particular review factors. For doing this we maintain an addition table containing average of the reviews for each atomic part of the path, the user submitted. More formally,

Let the array of path from source to destination= $[(X_1, Y_1), (X_2, Y_2), \dots, (X_r, Y_r)]$

where X_1, X_2, \dots, X_r are the latitude points and Y_1, Y_2, \dots, Y_r being the longitude points. $r \in \mathbb{N}$ is the length of the path. The server maintains the record of average reviews between each pair of segments of the path submitted by the users namely $[(X_i, Y_i) \Leftrightarrow (X_j, Y_j)]$ and $i \neq j$. So, when users asks to fetch the reviews for a path, the servers sends the data(average review rating) for each path segment.

In Table 1 we can see the actual representation of how record of every segment of the path is maintained. From table 1 we can get the average review for the whole path.

$$Path\ Ratings = [\frac{\sum_{i=1}^{r-1} x_{i1}}{r-1}, \frac{\sum_{i=1}^{r-1} x_{i2}}{r-1}, \dots, \frac{\sum_{i=1}^{r-1} x_{ik}}{r-1}]$$

Table 1 Average review for each segment of the path

PATH SEGMENT	AVERAGE REVIEW
$(X_1, Y_1) \Leftrightarrow (X_2, Y_2)$	$[x_{11}, x_{12}, \dots, x_{1k}]$
$(X_2, Y_2) \Leftrightarrow (X_3, Y_3)$	$[x_{21}, x_{22}, \dots, x_{2k}]$
.....
.....
$(X_{r-1}, Y_{r-1}) \Leftrightarrow (X_r, Y_r)$	$[x_{(r-1)1}, x_{(r-1)2}, \dots, x_{(r-1)k}]$

In Tab-1, we have the generalized format that how the path segment and the average reviews are stored in the database.

So, the path ratings can be easily calculated for each route/path that is available and taking in account of the user's choice of review factors, the route with the highest review ratings is shown.

4 Implementation details

4.1 Technologies used^[7]

The app is extensive made with MERN stack. MERN stands for MongoDB, Express, React, Node, after the four key technologies that make up the stack.

- MongoDB - document database
- Express(.js) - Node.js web framework
- React(.js) - a client-side JavaScript framework
- Node(.js) - the premier JavaScript web server

The MERN architecture allows you to easily construct a 3-tier architecture (front-end, backend, database) entirely using JavaScript and JSON. Fig-5 shows the MERN 3-tier architecture.

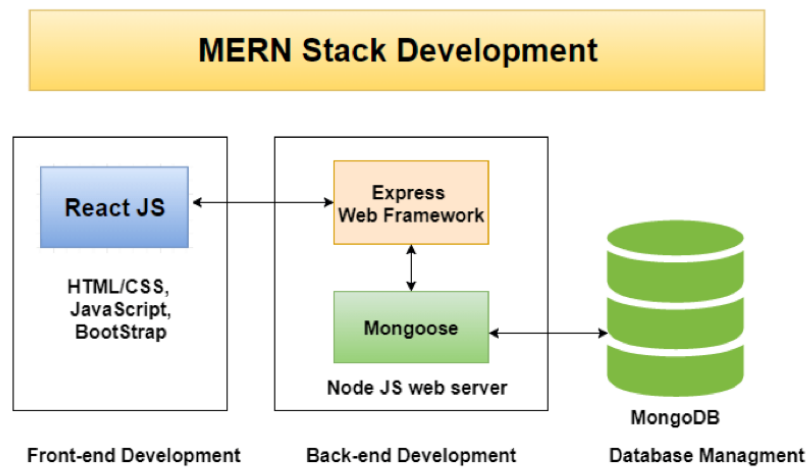


Fig. 5 MERN architecture

React JS^[30] is the MERN stack's top layer and the declarative JavaScript framework for building dynamic HTML client-side applications. We can use React to create complicated interfaces using basic Components, connect them to data on our backend server, and render them as HTML. React excels at handling stateful, data-driven interfaces with minimal code and suffering, and it comes with all the bells and whistles we'd expect from a modern web framework, including excellent support for forms, error handling, events, and lists, among other features.

Express JS^{[31][32]} is a server-side framework that runs inside a Node.js server at the next level below. Express.js describes itself as a "quick, unopinionated, minimalist web framework for Node.js," and it is precisely that. For URL routing (matching an incoming URL with a server function) and handling HTTP requests and answers,

Express.js includes strong models. We can connect to Express.js functions that power the application by sending XML HTTP Requests (XHRs), GETs, or POSTs from your React.js front-end. To access and change data in your MongoDB database, those functions use MongoDB's Node.js drivers, either via callbacks or Promises.

Mongo DB^[33] is a database that is easy to work with, such as React, Express, and Node, and saves all of our app's relevant data, tables, and so on. MongoDB parses JSON documents generated by your React.js front end and sends them to the Express.js server, where they're processed and (assuming they're legitimate) saved in MongoDB for subsequent retrieval.

4.2 Sign up and Sign in configuration

For sign up there are several fields like name, email, password. When the user clicks on the sign up button, the fields are set as a property of an object and the object is sent to the server by post request through axios. Now as the server receives the user's information via the object, it first validates whether the data is valid or not; for example whether the email is in correct format or the password has a minimum length or the values in the field are non-empty. If the validation fails which means user has submitted incorrect data, then the server sends a 401(unauthenticated) error code for incorrect email or password formats and 400(Bad request) error code for empty or null values in some fields. Fig-6 shows sign up and sign in system in the app.

Fig. 6 Signup and Signin feature

```
let res = await axios.post('/api/v1/auth/register', obj);
```

If the server validation is successful, it send back a JWT(JSON Web Token) containing some user info(not the password though), back to the front end. On the other hand, the server stores the user's info into the mongo DB database and the password is stored in encrypted format using Bcrypt.js. The encryption is a good practice since even the organization themselves won't know the passwords of all the users and safe guards user's data privacy.

For sign in, similar approaches are used. The user's details are sent in object format to the server by a post request through axios. Firstly, its checked whether the fields sent to the server is non-empty or not. Then it looks up in the database for the same record and checks whether the password matches or not. If password doesn't match, 401(unauthenticated) error code is sent back as response.

```
let res = await axios.post('/api/v1/auth/login', obj);
```

Fig-7 shows sign in and sign up process simultaneously side by side on how user can give their details and finally get the JWT to enter the website's home page.

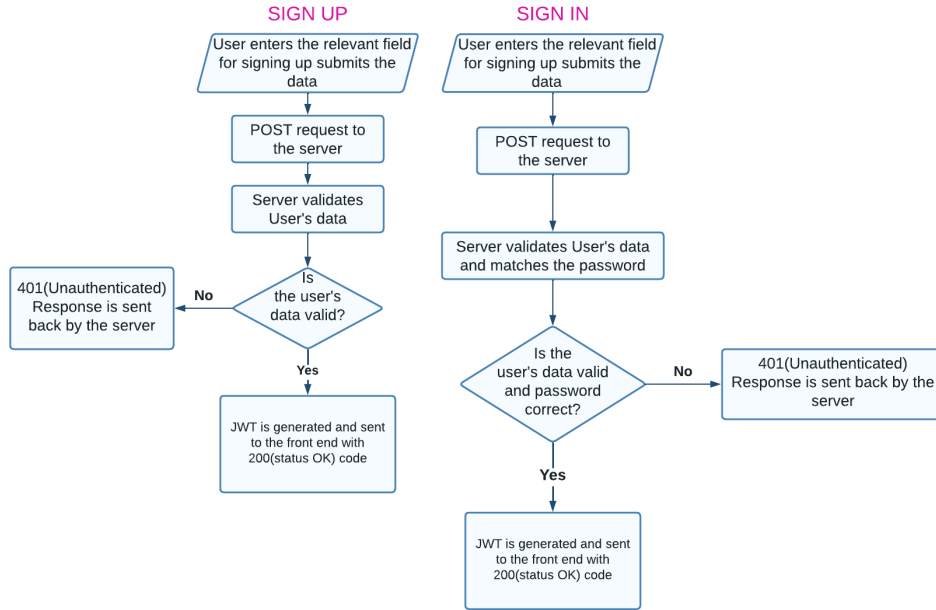


Fig. 7 Sign in and Sign up

4.2.1 JSON Web Token^[8](JWT)

JSON Web Token (JWT) is an open standard (RFC 7519) that defines a compact and self-contained way for securely transmitting information between parties as a JSON object. This information can be verified and trusted because it is digitally signed. JWTs is signed using a secret (with the HMAC algorithm) or a public/private key pair using RSA or ECDSA. In its compact form, JSON Web Tokens consist of three parts separated by dots (.), which are:

- Header
- Payload
- Signature

Therefore, a JWT typically looks like xxxxx.yyyyy.zzzzz

The header typically consists of two parts: the type of the token, which is JWT, and the signing algorithm being used, such as HMAC SHA256 or RSA.

To create the signature part, the server has to take the encoded header, the encoded payload, a secret, the algorithm specified in the header, and sign that.

The payload is the part where user details like name, email, etc are stored (not password).

All the routes of the server are protected which means, the user has to send this token along with the data to be a valid request to the server. It is sent in the Authorization header using the Bearer schema.

Authorization : Bearer < token >

Fig-8 shows the interaction of the frontend with the backend by JWT and without JWT.

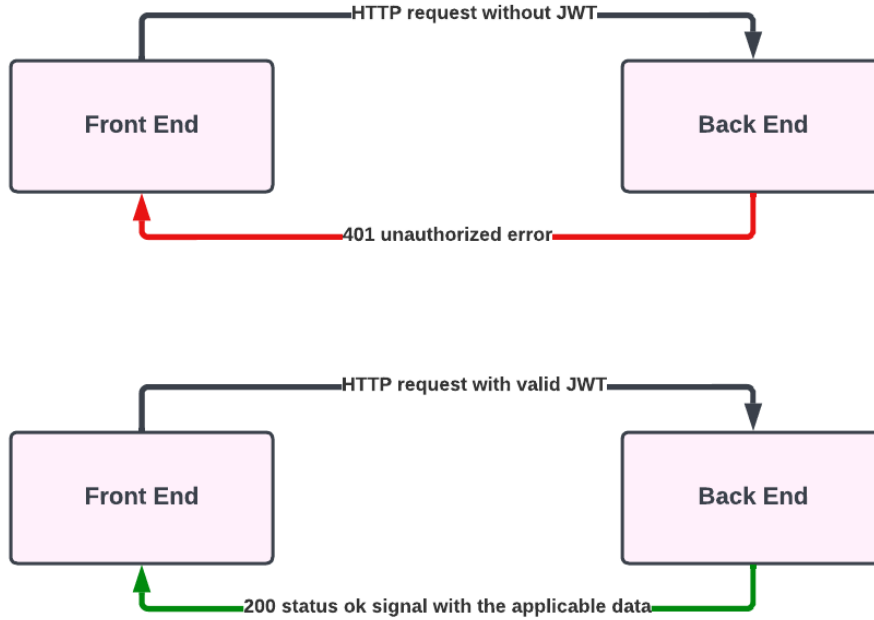


Fig. 8 HTTP request with and without JWT

4.2.2 Bcrypt.js^[9]

Bcrypt.js is a javascript package in node js that is used to hash password in our database instead of using just a plain text. It incorporates a salt to protect against rainbow table attacks, bcrypt is an adaptive function: over time, the iteration count can be increased to make it slower, so it remains resistant to brute-force search attacks even with increasing computation power.

To hash a password in database:


```
const salt = bcrypt.genSaltSync(10);
//bigger the salt, greater the security but also slower in hashing and
matching passwords
const hash = bcrypt.hashSync("B4c0//", salt);
```

To check a password:

```
// Load hash from your password DB.
bcrypt.compareSync("B4c0//", hash); // true
bcrypt.compareSync("notbacon", hash); // false
```

In the app, the passwords of the user are first encrypted using a salt. The more the length of the salt, the better the encryption

4.3 Home Page details

Home page is the main hub of activity of the app. Here user can enter their source or destination and get multiples routes on the map with a single route highlighted based on their preference like road condition, women safety, etc. An user can also select a particular route and review that route.

4.3.1 Search prediction using TomTom API^[10]

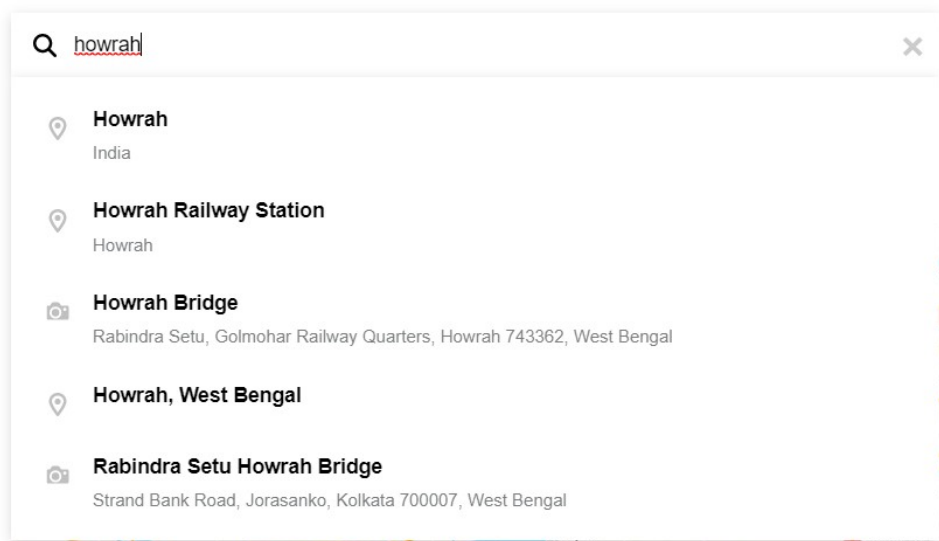


Fig. 9 search autocomplete feature(TomTom)

Fig-9 shows an example of search prediction for a text entered by the user in the search box. For using TomTom search box autocomplete, we have to create an object of the type `SearchBox` and pass in the constructor some search options like labels, idle time press, minimum number of character to start autocomplete, etc; along with the `Services` object from `@tomtom-international/web-sdk-services`

package. .

```
const ttSearchBox = new SearchBox(services, searchBoxOptions);
```

```
useEffect(() => searchRef.current.appendChild(
  ttSearchBox.getSearchBoxHTML()), []);
```

Here searchRef is a useRef react hook which refers to the search box which imple-

ments search autocomplete. so, we add our search box dynamically in our webpage by appendChild method of javascript. This is being done in a useEffect hook with empty dependency list, which means that only once as the contents of this page is loading this useEffect function will run.

We also have a current location button in the app. If it is pressed then the current location of the user automatically gets updated in the source field.

```
navigator.geolocation.getCurrentPosition(showPosition,showError);
```

The above line is executed as the current location button is pressed. showPosition is a function that is used to perform the activities after we get the location of the user and showError is another function which is used to handle error in case we don't get user's location. In showPosition function, we call the reverse geocoding api of tomtom using the user's current latitude and longitude. The api replies back with the user's address. Then this address is rendered in the source search box.

```
const res=await axios.get('https://api.tomtom.com/search/2/
reverseGeocode/{latitude},{longitude}.json?key=${process
.env.REACT_APP_TOMTOM_KEY}&language=en-GB&
entityType=MunicipalitySubdivision')
```

In Fig-10, an example of use of current location is shown.

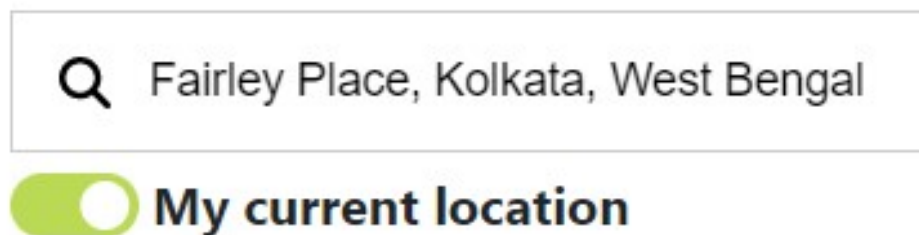


Fig. 10 current location feature

4.3.2 Map Rendering With Mapquest^[11]

After the user selects his source and destination in the respective field, he or she presses the get routes button to get the map and the routes between the respective location. Firstly, the key is set in the mapquest object by **L.mapquest.key**

property. Then a **directions** object is made by calling **L.mapquest.directions()**. Then we use the **route()** function of it, where we supply the source(that is typed by user), destination(that is typed by user) and other details in from of a object and pass another parameter **createMap()** function to create a map according to the property specified in the object passed.

```
L.mapquest.key = process.env.REACT_APP_MAPQUEST_KEY
let directions = L.mapquest.directions();
directions.route({
  start: document.getElementsByClassName(
    'tt-search-box-input')[0].value,
  end: document.getElementsByClassName(
    'tt-search-box-input')[1].value,
  options: {
    timeOverage:100,
    maxRoutes: 5,
  }
},createMap);
```

In the **createMap()** function, first **L.mapquest.map('map',obj)** is called to create the map object where obj is an object where we pass important properties of the map like it's center, layer, zoom, etc. Then we create a **directionsLayer** object by calling the **const customLayer=L.mapquest.**

```
directionsLayer(obj) where in obj we pass important properties of the start
marker, end marker, route ribbon, alternate route ribbon, etc. Then we set the
customLayer.on('route_selected', function(eventResponse) {
.....
})
```

In the event response function, the piece of code which needs to execute when the user selects a route is written. First, in the function, the **maneuvers**(containing array of latitudes and longitudes of the route selected by the user) are extracted from the **eventResponse** parameter passed in the function. Then the **maneuvers** are stored in the local storage for further access in different part of the website. We also store the source and the destination name in the local storage for the same reason. Then the **customLayer** is added to the map by **customLayer.addTo(map);**

4.3.3 Route selection

After the map is rendered in the browser, by default a route(most likely the one having minimum distance) is shown as the main route and others as an alternate route. There are list of checkboxes available to the user. The user can click on some of the boxes and those review factors will be taken into account and the corresponding main route will be calculated and displayed. Fig-11 shows a modal where user can select their choice of review factors according to which route recommendations will be given to them

So at first(while rendering the map itself), we send a http post request to the server

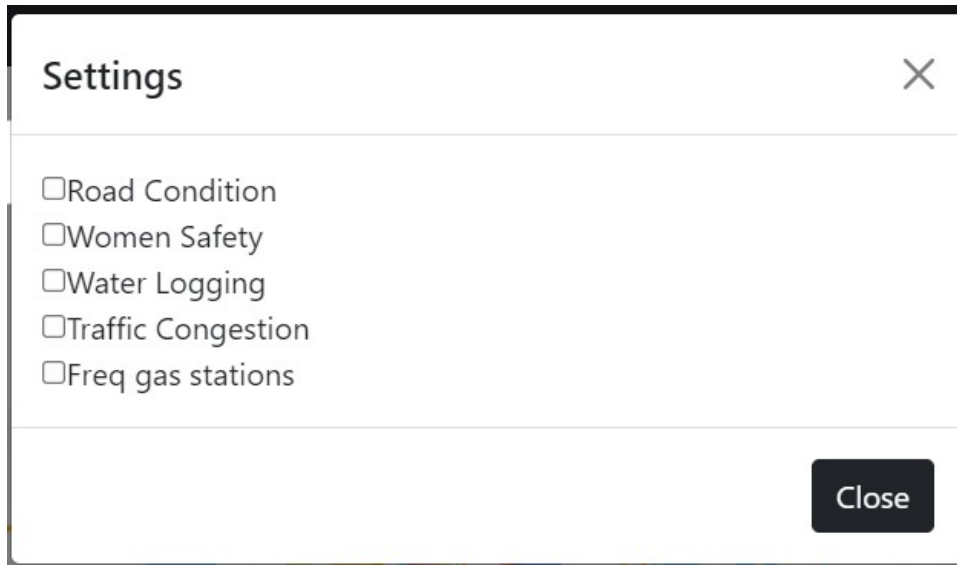


Fig. 11 checkboxes for route selection

along with the array of maneuvers(array of latitudes and longitudes) and the server returns an array containing average rating of each part of the part. This is possible because in the database we maintain a record for each part of the path, the user reviewed and that part's average rating is returned. This is more clearly explained in ??.

```
await axios.post('/api/v1/latlng',{maneuvers},{
  headers: {
    Authorization: 'Bearer ${localStorage.getItem("token")}',
  }
});
```

Here, we can notice that this token is actually the JWT token we got we logged in or signed up. Now for each review factor we store the sum of the respective ratings(of those review factors which the user wants) of all the parts in an array. Then from the array we see which route has the maximum rating. After knowing which route is to highlighted as the main route, that route is selected by the line

```
customLayer.selectRoute(reviewOptimization.optimizedRoute);
```

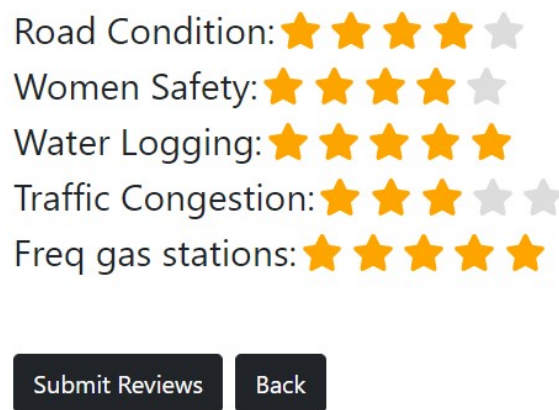
Here customLayer is the layer of the map and **selectRoute()** is the function to change the main route. **selectRoute(0)** selects the default route (by mapquest). **reviewOptimization** is a global object that keep records of results returned by the server after the http post request to **'/api/v1/latlng'** and we are setting the **optimizedRoute** property to the route number which we calculated before.

4.3.4 Review submission

The user can select his or her route by clicking on it and can decide to review that particular by clicking on a button. Now as the user selects a route, as stated ear-

lier, the maneuvers(containing array of latitudes and longitudes) are saved in local storage by **JSON.stringify(maneuvers)** along with the source and destination. Then the user is redirected to a page where they can give rating to all types of review factors.

After the user gives the rating of all the review factors and clicks on the submit button, its checked whether the user has given reviews to all the review factors or not. If not, then the user is being requested to review properly and submit it again. The maneuvers are seperated into two arrays of latitudes and longitudes. The source and destination also is retrieved from the local storage. All these info about the related path and the review of the user is stored in form of an object and send via http post request to server. Then server goes on a series of validation as explained in 3.4. Fig-12 shows the different review factors and the ratings an user has given.



Road Condition: ★★★★★

Women Safety: ★★★★★

Water Logging: ★★★★★

Traffic Congestion: ★★★★★

Freq gas stations: ★★★★★

Submit Reviews Back

Fig. 12 Review submission by user

```
await axios.post('/api/v1/reviews',obj,{
headers: {
Authorization: 'Bearer ${localStorage.getItem("token")}',
}});

await axios.post('/api/v1/train',obj,{
headers: {
Authorization: 'Bearer ${localStorage.getItem("token")}',
}});

await axios.post('/api/v1/validate',obj,{
headers: {
Authorization: 'Bearer ${localStorage.getItem("token")}',
}});
```

Http post request to **/api/v1/reviews** is used to submit the user's rating, where as **/api/v1/train** is used to check whether review is fake or not against our pretrained machine learning model and **/api/v1/validate** is to check whether the review is spam review or the review's originality can be confirmed by user's location.

Fig-13 we see how user selects their routes and based on that that how they can submit their reviews corresponding to that route.

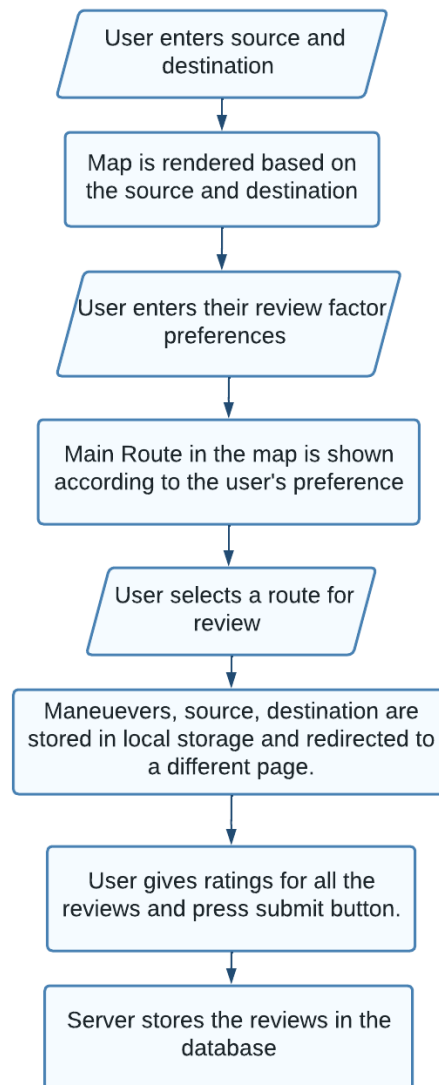


Fig. 13 Route selection and review submission

4.4 Review training and validation

Review training and validation is explained in great details in 3.4. In this subsection, we are going to see in details how machine learning algorithms are used to train our model and then using it for validation of reviews. Also, we will see about offline validation and instant validation.

4.4.1 Training offline

In offline training first we make two arrays, one is the training set and another is the prediction set. Each member of the training set is an UNID (Unique Natural Id) i.e. the unique natural number given to a unique path. Each member of the prediction set is an array denoting the ratings given by users for review factors. Thus, if we give a particular natural number, our model can prediction the review array, which in turn will help us validating the reviews.

```
const options = {  
  seed: 3,  
  replacement:true,  
  nEstimators: 25  
};  
const regression = new RFRegression(options);  
regression.train(trainingSet, predictionSet);  
const result = regression.predict(trainingSet);
```

We are using random forest regression in this case with seed as 3 and n estimators as 25. After we populate the training set and the prediction set, we first train the our model using the above **.train()** method. Then we predict the reviews of every route available in the database by **.predict()** method and store in our database. So, in real time when the user submits the review, the machine learning model needn't be trained again or predict anything. Instead we look in our database for the path and the predicted review for it. If we see a lot of deviation from the predicted review, the user's review will be rejected.

4.4.2 Predicting reviews realtime

As the user submits their review via http post request to the server. The server first checks whether the path is reviewed or not by it's UIPS(Unique Identifiable Path String). UIPS is made by turning latitudes and longitudes array into string and concatenating them

```
(JSON.stringify(latitudes)+JSON.stringify(longitudes))
```

We actually keep a table in our database with UNID(Unique Natural Id) against UIPS for fast access. So, first the server sees whether UNID is present against the particular UIPS. If not then it sends a msg to the user that this route has never been reviewed before and thus our machine learning model won't be able to predict it correctly. If the server indeed gets a UNID corresponding to it's UIPS, it checks another table(where we stored the predictions of reviews against all routes) and fetches out the prediction. If such record is not found again it sends a msg to the

user that this route is not yet trained in the model. If the record is found it sends the user complete details how much percentage error is there is the review submitted by him or her and based on this whether server accepted or rejected the review.

4.4.3 Offline review validation by mean difference

Certain review factors like 'Road Condition' and 'Water Logging' goes hand in hand, meaning if road condition is good, then water logging will not occur and if the road condition is bad, then there is a tendency of water logging. So, if we find the mean of difference of ratings of 'Road Condition' and 'Water Logging' for all the reviews. Then, this mean must be very small and negligible. On the other hand review factors like 'Women Safety' and 'Traffic congestion' goes in opposite ends with each other, i.e. the more bad the traffic congestion is, the better the women safety will be and vice versa. Thus, if we find the mean difference of ratings of 'Women Safety' and 'Traffic congestion' for all the reviews, we will find that the mean is likely to be very high.

For the first example 'Road Condition' and 'Water Logging' we go through all the reviews by `arr.map((item)⇒{...})` and for each item in the array we find `RF[ind1]-RF[ind2]` and store it in a variable to find sum of difference. Here, ind1 is the index of 'Road Condition' review and ind2 is the index of 'Water Logging' review. Then, we sort our initial array in the increasing order of their difference. Then, until the mean difference comes back under some small value, we remove elements from the end of the array using `.pop()` method. Thus, after this operation, the reviews in our list will have a low mean difference between 'Road Condition' and 'Water Logging' since the mean difference is greater at the end of the array and we are removing from the end. Similarly, for review factors like 'Women Safety' and 'Traffic congestion' that have an opposite relation; we sort our review array in the opposite order and remove the elements from the end of the array until the mean difference is significantly high. Here, in this case, the end of the array has reviews with small mean difference; so removing them increases average difference. Further details can be found in 3.4.2

4.4.4 Instant Review Validation

As discussed in 3.4.2, there are two types of instant review validation done, one being based on the frequency of the review in the same route (detecting spam reviews) and the other is the based on the user's location i.e. if the user's current location is close to source or destination, then it is most likely to be a real review. For the former, we find out the time (in days) when the same user reviewed the same route. If he or she is giving the reviews to the same route in a very small span of time, then his or her review will be considered as a spam review.

const days=(Date.now()-Date.parse(date))/(1000*60*60*24);

In this way we get number of days elapsed before the user last reviewed the route. For the later, we use haversine's formula to find the distance between the user's current location and the source given by the user. Also, we find out the distance between user's current location and the destination given by the user. Then, if any one of the distance is small, it means user is close to either source or destination. Hence, the review will be considered as likely genuine.

Haversine's formula^[29] :

$$a = \sin^2(\Delta\text{latDifference}/2) + \cos(\text{lat1}) \cdot \cos(\text{lat2}) \cdot \sin^2(\Delta\text{lonDifference}/2)$$

$$c = 2 * \text{atan2}(\sqrt{a}, \sqrt{1-a})$$

$$d = R * c$$

where,

$\Delta\text{latDifference} = \text{lat1} - \text{lat2}$ (difference of latitude)

$\Delta\text{lonDifference} = \text{lon1} - \text{lon2}$ (difference of longitude)

R is radius of earth i.e 6371 KM or 3961 miles

and **d** is the distance computed between two points.

4.5 React-Bootstrap^[12]

Our goal is to make our app responsive i.e. our app looks good in all types of screen sizes like mobile screen, laptop screens, etc. We use react-bootstrap for this purpose. Bootstrap works on breakpoints on the screen. According to these breakpoints, the bootstrap elements may be customized to look like. For example on a small mobile screen, a Col can spread across the whole screen where as on a large screen, it may be spread across on half of the screen. This can be done using `<Col sm="12" lg="6">content</Col>`. Here 12 denotes the whole width of the screen and thus 6 denotes only half the screen. These breakpoints are regulated by media queries in the internal setup of bootstrap.

We used Container, Row, Column elements of bootstrap to properly align items in grid format. Note: by default Row element of bootstrap has property of `display:flex`. In addition to this we used Modal element to make modals in our app and Table to print table. The navbar is made using Navbar element. Tab-2 shows the breakpoints for different screen sizes.

Table 2 Bootstrap Breakpoints

Breakpoint	Class infix	Dimensions
Extra small	None	<576px
Small	sm	≥576px
Medium	md	≥768px
Large	lg	≥992px
Extra large	xl	≥1200px
Extra extra large	xxl	≥1400px

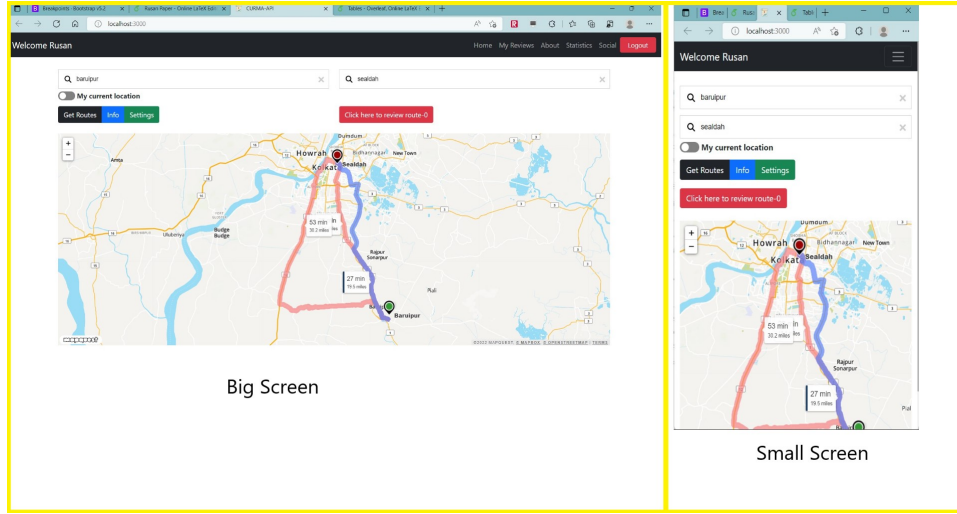


Fig. 14 Responsiveness of website

4.6 Security Features

There are mainly four npm packages that is used to maintain security and illegal access of users in the app.

- *JWT*^[8] (discussed in 4.2.1)
- *Bcrypt.js*^[9] (discussed in 4.2.2)
- *Dotenv*^[34]
- *Cors*^[35]

There are certain values we cannot hard code in our server end or front end like key of an app or password of database. For this we use *dotenv*^[34] package. we never hard code our key directly but take the value from system variables. These variables can be setup in *.env* file in node js and *.env.local* file in react js in the local environment and in deployment we can set it up in the domain website configuration. We hide the *MONGO_URI*, *JWT_SECRET*, *JWT_LIFETIME* in the node js configuration. *MONGO_URI* contains address along with password to connect with mongo db. *JWT_SECRET* is a 256 bit generated text to sign the JSON web token. *JWT_LIFETIME* is the validity of the JWT. Similarly, in react configuration we have *REACT_APP_MAPQUEST_KEY* and *REACT_APP_TOMTOM_KEY*. They are app keys of mapquest and tomtom. We can access them anytime in the website by example:
`process.env.MONGO_URI,`
`process.env.REACT_APP_MAPQUEST_KEY`

Cors^[35] package is used to modify the type of requests which can access the server. We can enable cross-origin requests as well as disable it. We can also use it to enable cross origin only for a particular domain. So, a lot of customization can be done with the cors package and it provides us security in the fields and domain we desire.

```
app.use(cors())
```

This allows all cross origin reference. Any site can send http request to the server.

```
const corsOptions = {
  origin: 'http://example.com',
  optionsSuccessStatus: 200
}
app.use(cors(corsOptions))
```

This is an example where we send an corsOptions object in cors as parameter. Here we have set the origin to a website. So, requests from only this website or domain will be served. Any request from any other domain won't be entertained by the server and will throw CORS error in the browser.

4.7 Deployment

We deployed our app in *heroku*^[36]. There are few configuration which needs to be done before we deploy our app to heroku. At, first we have to add a few lines of code in our app.js file from where our whole node js server runs.

```
if(process.env.NODE_ENV === 'production'){
  app.use(express.static('./client/build'));
  app.get('*', (req, res) => {
    res.sendFile(path.resolve(__dirname, 'client', 'build', 'index.html'))
  })
}
```

Our app is in deployment it will run a post script build and store the static pages of react in './client/build'. By the above lines of code snippet we are telling the server that when it is in production and it gets http request other than the ones defined in the backend, it will render the index.html page that is generated by post build script.

In our package.json file we also have to make some change like we have to set

```
"dev": "concurrently \ "npm run server \ " \ "npm run client \ ""
"heroku-postbuild": "NPM_CONFIG_PRODUCTION=false npm in-
stall --prefix client && npm run build --prefix client"
```

The first one is for starting the react js and node js application simultaneously. We need to install concurrently package for executing this. we can always start the server along with react application in our local environment with npm run dev. The second one is for running the heroku postbuild script which tells the server to run npm build in the react application(when not in production).

The next step is to global install heroku CLI and git. Then we need to create our account in heroku. Then we should run the following commands:

```
$ heroku login
$ git init (If we are making the git repository for the first time)
$ git add .
$ git commit -am "make it better"

$ heroku create -a curma-api
OR
$ heroku git:remote -a curma-api

$ git push heroku main
```

So, first we are logging into our heroku account. Then we initialize our git repository if it is not. Then we add all the changes made in the git. Then we commit our changes in the git. If we already created our remote site repository in our heroku account we can use heroku git:remote otherwise we use heroku create to create a new repository in heroku for deployment. Then we push our app to heroku server by git push heroku main(In some cases main has to be replaced by master). The app is deployed in the address <https://curma-api.herokuapp.com/>. One last final thing to do is to set up our environment variables in the server. To do this we go in our app repository in heroku and click on the settings tab. There we can click on the Reveal Config Vars button and write our secret variables like mongodb connecting url and map keys.

Fig-15 shows the reveal config vars button in the heroku website, where we can set the system variables with confidential information like mapquest, tomtom api keys, mongo db connect url, etc.

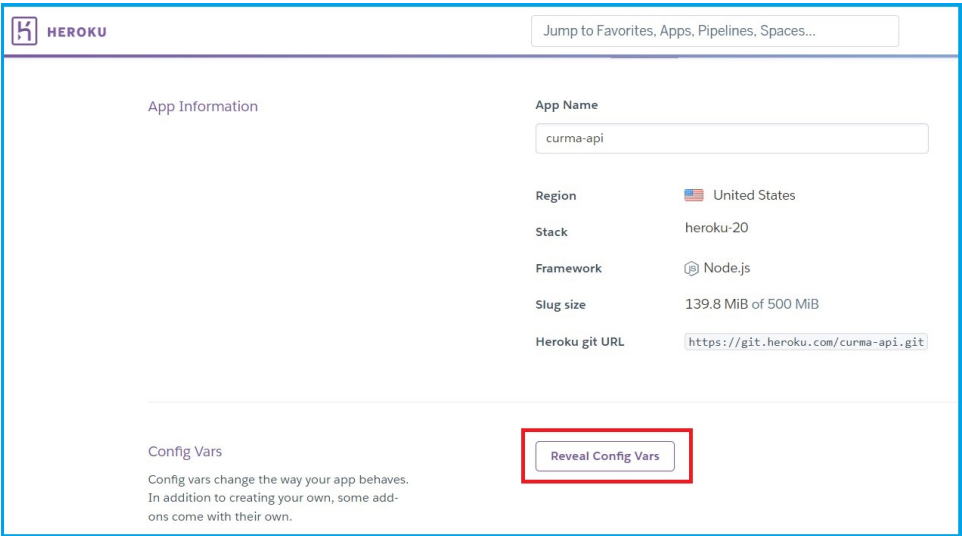


Fig. 15 Setting up environment variables

5 Experimental Results

In this section, we will see the performance of few of the important machine learning regression algorithm on both real and synthetic dataset via multibar graphs. We will also see the actual working of the route recommendation system.

5.1 Experimental setup

We have two kinds of data- Real and synthetic. The collection of real data was simple; the app was deployed on heroku and the website link is passed to different users for review submission. For generation of synthetic data we made a dummy controller and made a set of synthetic data using Amazon Musical Systems as proposed in 3.3. We took 5 review factors: Road Condition, Women Safety, Water Logging, Traffic Congestion, Frequent Gas Filling Stations. We generated our synthetic dataset on the three routes of jadavpur-howrah and we took two routes of priyo builders-sealdah railway station for real data analysis.

To generate graph we need data about the performance of all the machine learning algorithms. A separated controller is made in node js. We can access this controller via a HTTP GET request with a request param in the URL:

`/api/v1/train/:type` . In the controller firstly reviews are taken from the database by : `const record=await Review.find({})` .We specify the machine learning regression algorithm we want in the type parameter. Then we use postman (An app for testing http requests of the server without building frontend) to send request to the server to get ml algorithm's info one by one. Then we use these data route by route to make the charts. After that based on our request, the machine learning algorithm is run and result is sent to the user in JSON format, where against each route we have array of predicted reviews by the ML algorithm.

Before each chart there is a table which depicts clearly how many reviews are there and its' distribution in different review factors to get a clear understanding of our data

5.2 Evaluation Metric

We have used 7 ML regressor algorithm namely Random Forest Regression, Simple Linear Regression, Decision Tree Regression, Polynomial Regression, Exponential Regression, Theil Sen Regression and Multivariate Linear Regression.

5.2.1 Random Forest Regression^[13]

Random Forest Regression is a supervised learning approach for regression that use the ensemble learning method. The ensemble learning

method combines predictions from several machine learning algorithms to produce a more accurate forecast than a single model. we set in options object seed: 3, replacement:true, nEstimators: 25.

```
const { RandomForestRegression } = require('ml-random-forest');
const regression = new RandomForestRegression(options);
regression.train(trainingSet, predictions);
const result = regression.predict(trainingSet);
```

5.2.2 Simple Linear Regression^[14]

A linear regression model with a single explanatory variable is known as simple linear regression. That is, it deals with two-dimensional sample points with one independent variable and one dependent variable, and it seeks out a linear function that predicts the dependent variable's values as a function of the independent variable.

```
const SimpleLinearRegression = require('ml-regression-simple-linear');
const regression = new SimpleLinearRegression(x, y);
regression.predict(value);
```

5.2.3 Decision Tree Regression^[15]

In the shape of a tree structure, a decision tree constructs regression or classification models. It incrementally cuts down a dataset into smaller and smaller sections while also developing an associated decision tree. A tree with decision nodes and leaf nodes is the end result.

```
const { DecisionTreeRegression } = require('ml-cart');
const reg = new DTRegression();
reg.train(x, y);
const estimations = reg.predict(x);
```

5.2.4 Polynomial Regression^[16]

The link between the independent variable x and the dependent variable y is treated as an n th degree polynomial in x in polynomial regression. Here we have taken degree=5.

```
const PolynomialRegression = require('ml-regression-polynomial');
const regression = new PolynomialRegression(x, y, degree);
console.log(regression.predict(value));
```

5.2.5 Exponential Regression^[17]

The process of obtaining the optimal exponential function equation for a collection of data is known as exponential regression. As a result, we

get an equation of the form $y=ab^x$ where $a \neq 0$.

```
const ExponentialRegression=require('ml-regression-exponential');
const regression = new ExponentialRegression(x, y);
regression.predict(value);
```

5.2.6 Theil Sen Regression^[17]

Theil–Sen Regression is a method for fitting a line to a set of sample points in the plane (Simple linear regression) by selecting the median of all line slopes via pairs of points.

```
const TheilSenRegression=require('ml-regression-theil-sen');
const regression = new TheilSenRegression(inputs, outputs);
const y = regression.predict(value);
```

5.2.7 Multivariate Linear Regression^[18]

Multivariate Regression is a supervised machine learning algorithm involving multiple data variables for analysis. Multivariate regression is an extension of multiple regression with one dependent variable and multiple independent variables. Based on the number of independent variables, we try to predict the output. For other algorithms, we have taken the review factors one by one and trained model for each of them separately. But in multivariate linear regression we train the model with the whole review factor array against each route available all at once.

```
const MLR=require('ml-regression-multivariate-linear');
const mlr = new MLR(x, arrayOfArrayOfReviewFactors);
console.log(mlr.predict(arrayOfReviewFactors));
```

5.3 Result Analysis

The graphs in fig-16, fig-17, fig-18, we ran 7 different ML algorithms on the three route between jadavpur and howrah (Route-0, Route-1, Route-2) and can be compared by the vertical multibar graphs for the three routes.

The graphs in fig-19 and fig-20, we do the same type of graph on two routes between priyo builders and sealdah station (Route-0 and Route-1).

	One	Two	Three	Four	Five	Total Reviews
Road Condition	6	9	30	124	331	500
Woman Safety	7	11	40	105	337	500
Water Logging	5	9	103	353	30	500
Traffic Congestion	345	100	39	8	8	500
Freq gas stations	11	27	362	94	6	500

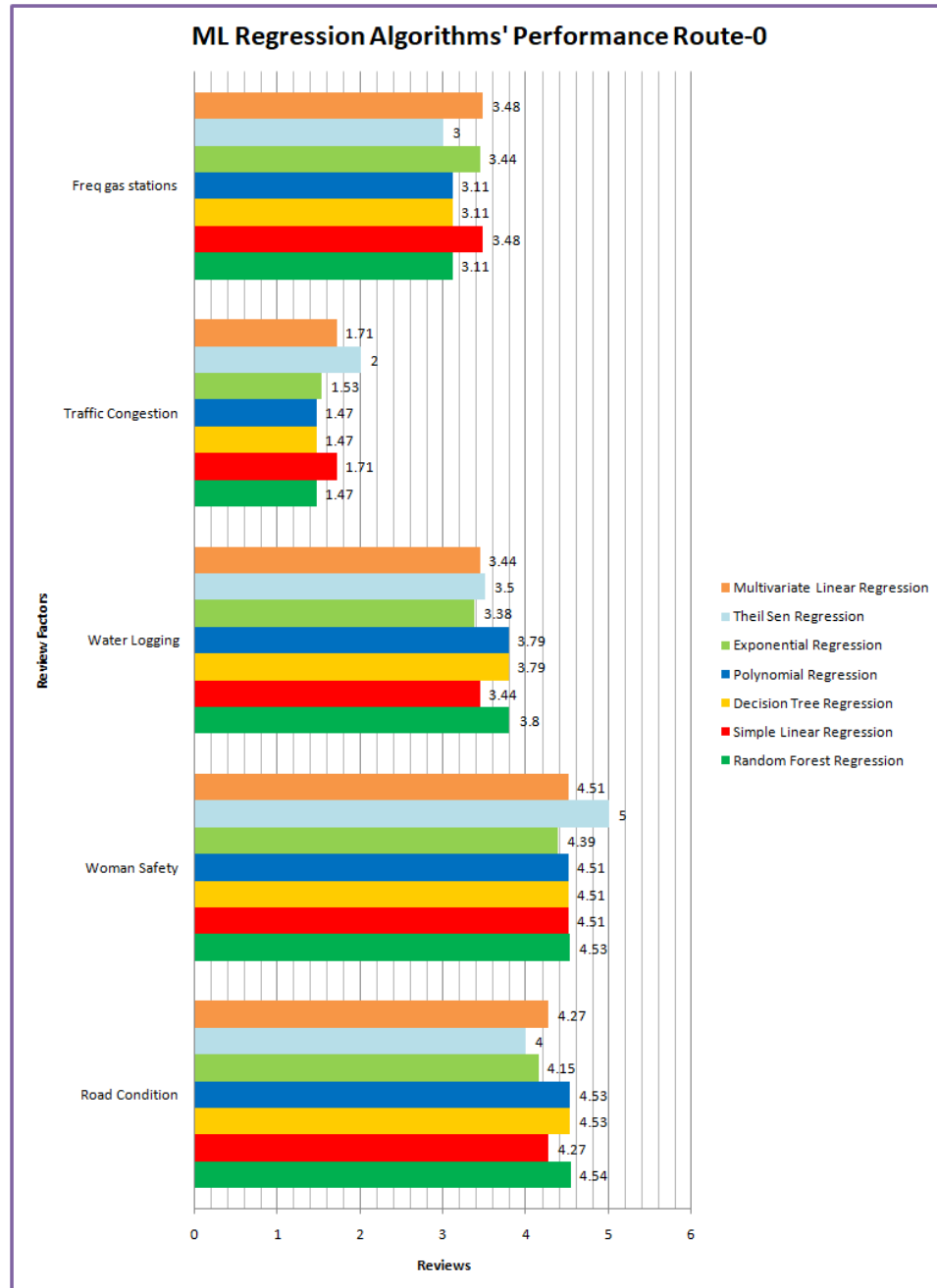


Fig. 16 Jadavpur-Howrah Route-0 graph for synthetic dataset

	One	Two	Three	Four	Five	Total Reviews
Road Condition	10	16	96	346	32	500
Woman Safety	13	10	32	103	342	500
Water Logging	12	42	341	89	16	500
Traffic Congestion	39	330	109	7	15	500
Freq gas stations	10	8	107	334	41	500

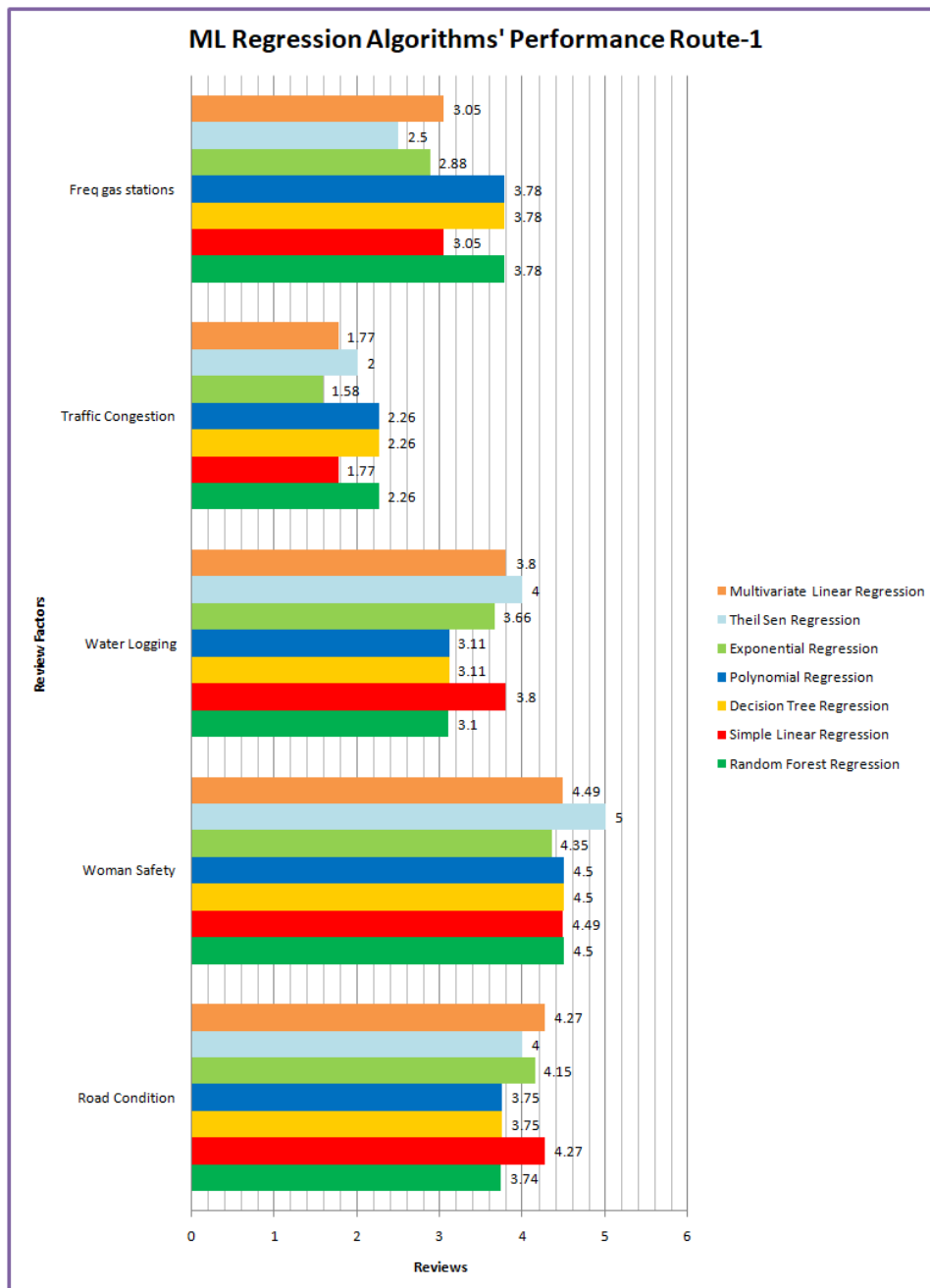


Fig. 17 Jadavpur-Howrah Route-1 graph for synthetic dataset

	One	Two	Three	Four	Five	Total Reviews
Road Condition	7	13	32	98	351	501
Woman Safety	12	16	37	100	336	501
Water Logging	11	10	35	100	345	501
Traffic Congestion	325	99	47	17	13	501
Freq gas stations	47	332	84	26	12	501

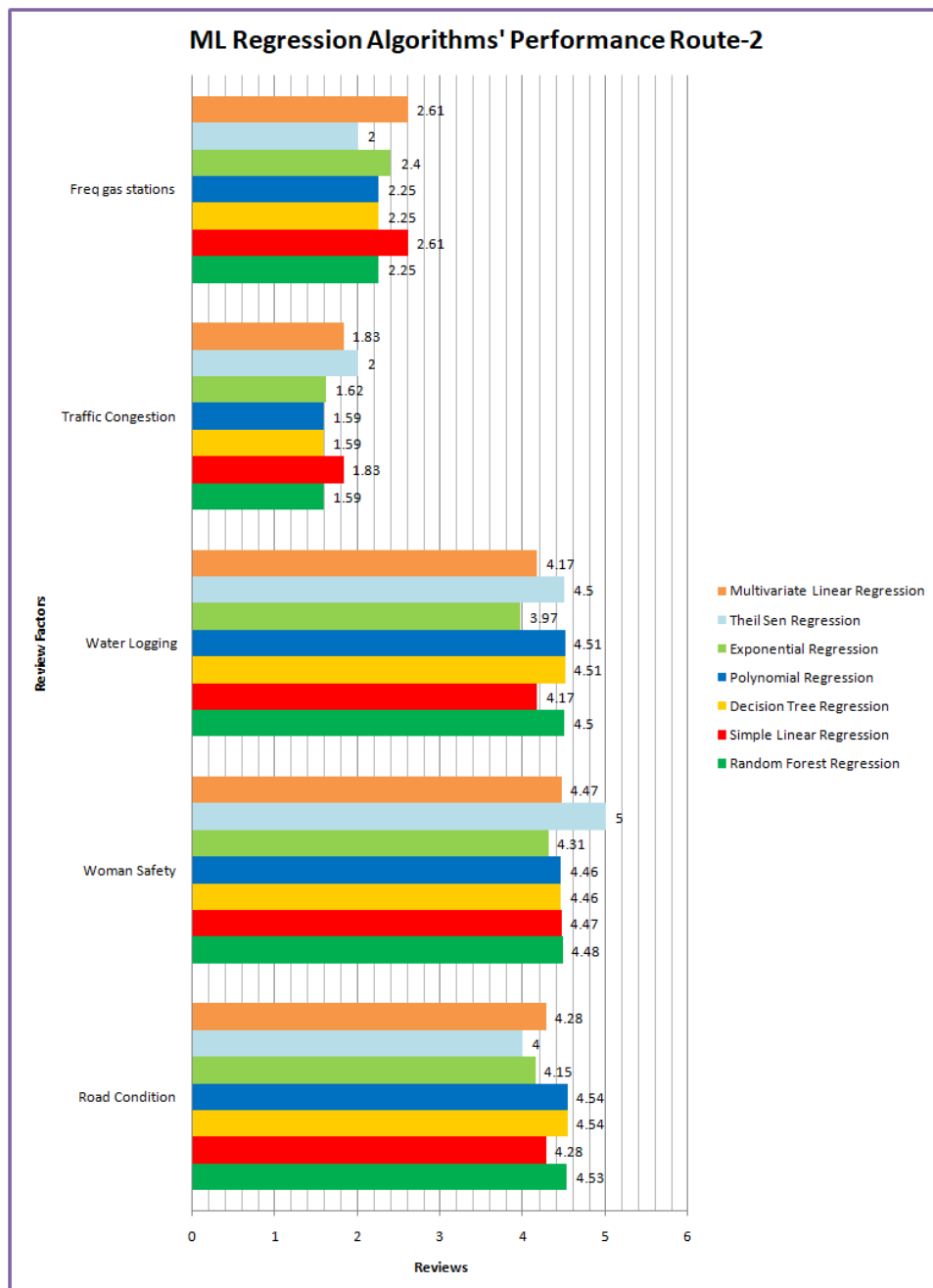


Fig. 18 Jadavpur-Howrah Route-2 graph for synthetic dataset

	One	Two	Three	Four	Five	Total Reviews
Road Condition	0	0	3	2	0	5
Woman Safety	0	0	2	3	0	5
Water Logging	0	1	4	0	0	5
Traffic Congestion	1	1	3	0	0	5
Freq gas stations	0	2	1	2	0	5

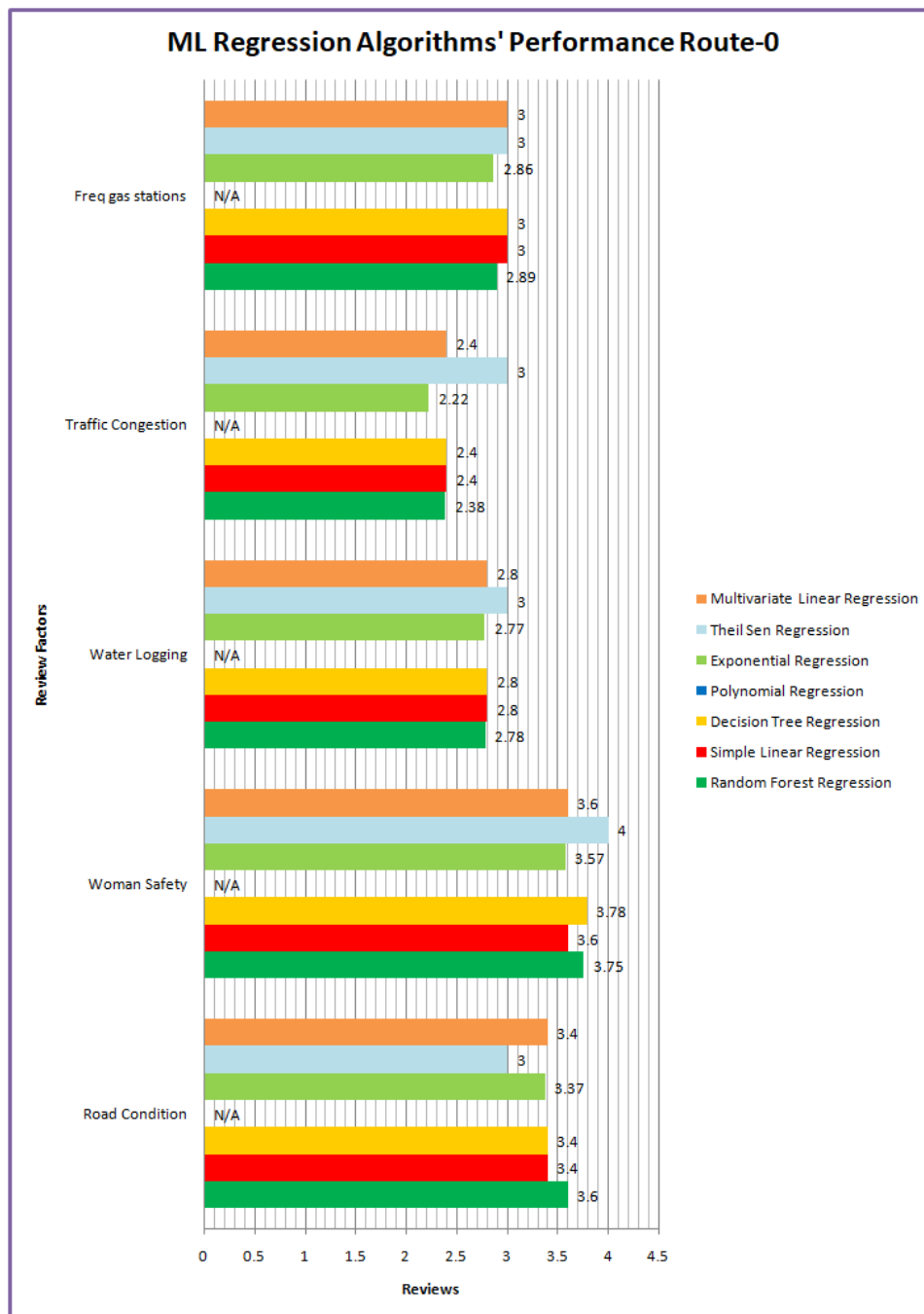


Fig. 19 Priyo builder-sealdah railway station Route-0 graph for real dataset

	One	Two	Three	Four	Five	Total Reviews
Road Condition	0	0	0	4	0	4
Woman Safety	0	0	0	4	0	4
Water Logging	0	0	2	1	1	4
Traffic Congestion	0	1	2	1	0	4
Freq gas stations	0	0	1	3	0	4

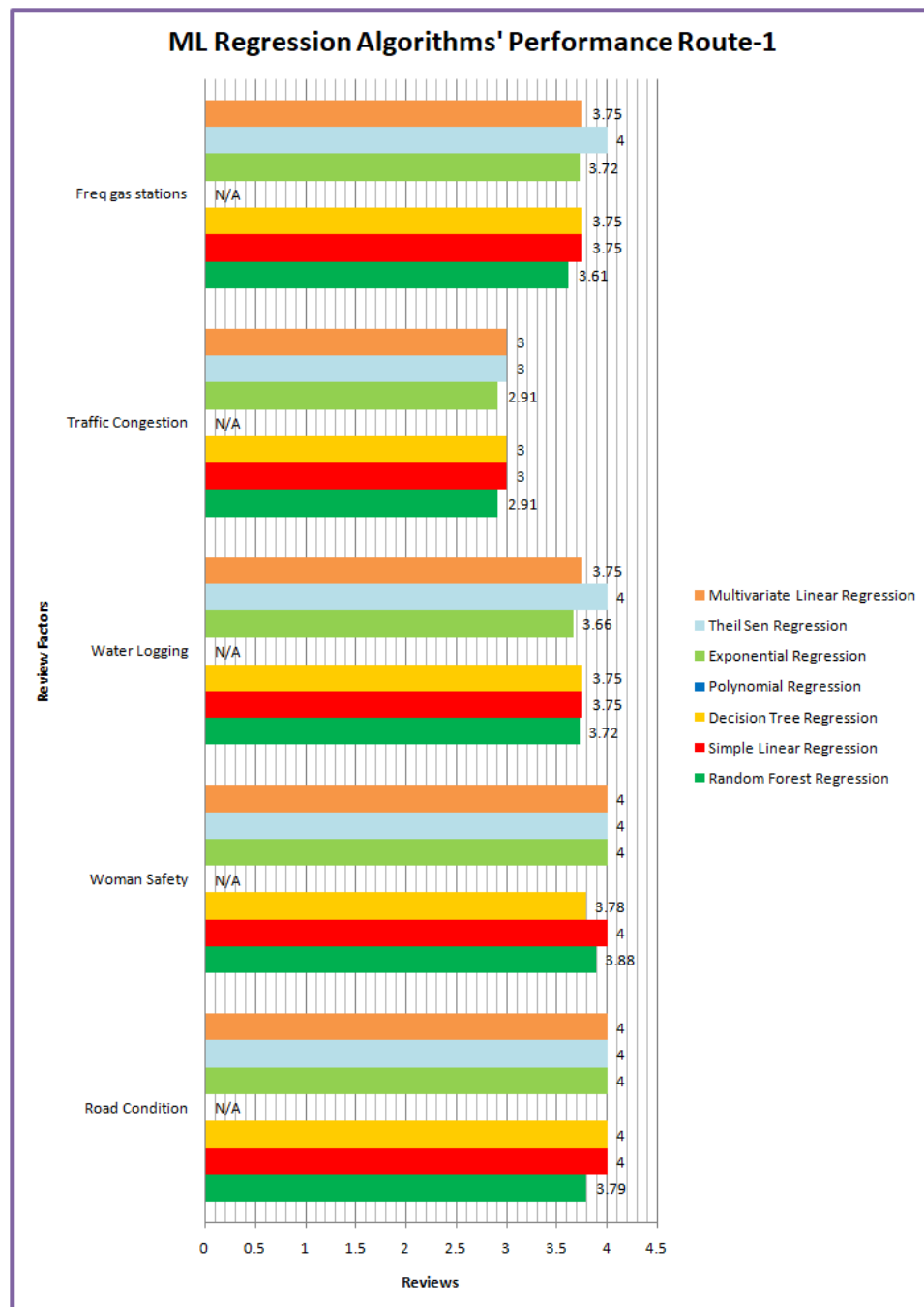


Fig. 20 Priyo builder-sealdah railway station Route-1 graph for real dataset

Here are few inferences based on the chart results and ML algorithm outputs from Fig.16, Fig.17, Fig.18

- Random Forest Regression, Decision Tree Regression and Polynomial regression almost gives same result in most of the cases barring some rare cases of difference in few decimal points.
- Simple Linear Regression and Multivariate Regression is exactly the same for all the review factors in all the graphs.
- Theil Sen Regression overfits our data as it either gives whole number as a result or half way between the whole numbers.
- Random Forest Regression takes way more time to train than other as it creates multiple random models to predict the outcome.
- Exponential Regression gives different output in all cases in comparison with the other regression algorithm. Thus, it is not a good estimator for our data.

5.3.1 Real vs synthetic dataset

Synthetic dataset is majorly used to validate the efficiency and working of the app. Here are some of the pros of using synthetic dataset, which can be also illustrated from Fig.18, Fig.19.

- Real data takes a lot of time to collect from different users and agencies where as synthetic data can be generated instantly.
- There is always low number of real data which may cause most of the machine learning algorithm to give wrong result as training data is very less. As in our case in Fig.18, Fig.19 we can see that polynomial regression algorithm couldn't fit a polynomial curve to our result as LU matrix was singular. But we did n't face such kind of problem in synthetic set graphs.
- In real data user may give a biased review as he can prefer only one particular type of area or can follow some trends. In synthetic data set there is no such case and can judge the effectiveness of the machine learning algorithms efficiently.
- Synthetic data has huge amount of data, thus we have our bases covered for all kinds of data including corner cases and thus gives us a fair idea about how our app will work when real data of the users increases gradually.
- In the Fig.18, Fig.19 we can see lot of variations among different ML algorithms than in Fig.16, Fig.17, Fig.18; which suggests us that the ML algorithms can make more sense of the synthetic data as we have sufficiently large amount of data.
- Synthetic data need not be validated by offline validators(explained in 3.4.2), whereas real data needs to be validated.

Some of the cons of using synthetic dataset are as follows:

- Synthetic dataset may show the effectively of working of our ML algorithms but fails to give the original picture or view of the real roads and places, which is the primary objective of the app.

-
- **Synthetic dataset only validates the working of our app. So, it is only a temporary solution to review collection and ultimately we have to collect real data to make the app effective in real world.**

5.4 Results of working of route recommendation

5.4.1 Setup for route recommendation

In the app there are several areas where input is given but we will only concentrate on one area i.e., Jadavpur-Howrah on how the three routes between them are stored in our database and then recommend a road between two places(Quest Mall-Sailen Manna Stadium) where there are no direct reviews given but derived from part of the roads between jadavpur and howrah routes. The average reviews factors arrays are rounded upto 2 decimal places and the review factors array denotes the factors Road Condition, Women Safety, Water Logging, Traffic Congestion, Frequent gas stations in this order.

5.4.2 Review submission

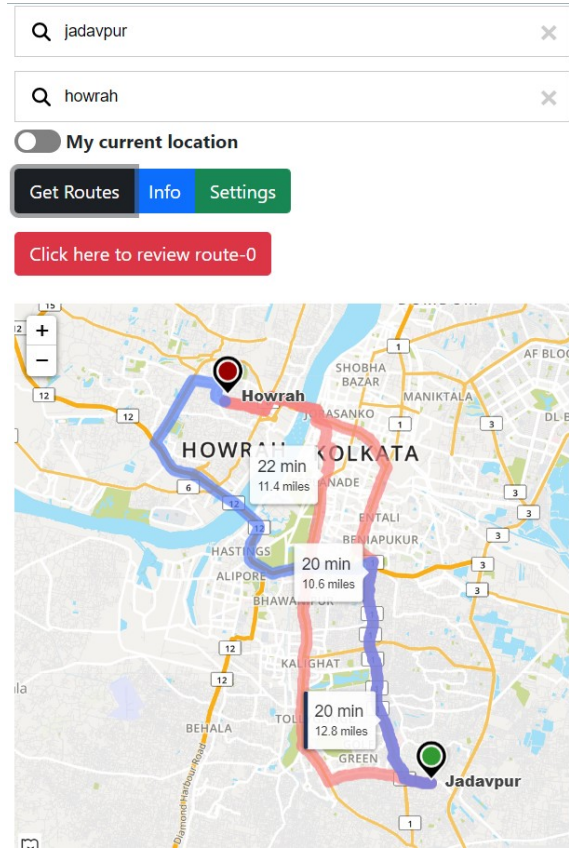


Fig. 21 Routes between Jadavpur and Howrah

Jadavpur-Howrah Route-0 latitude array: [22.482809, 22.483868, 22.516142, 22.540028, 22.539999, 22.542913, 22.540934, 22.54063, 22.575369, 22.592461,

22.591572, 22.588057, 22.586754]

Jadavpur-Howrah Route-0 longitude array: [88.383324, 88.375473, 88.366486, 88.365417, 88.365417, 88.366081, 88.34127, 88.334686, 88.301804, 88.31295, 88.320549, 88.319809, 88.322624]

Jadavpur-Howrah Route-1 latitude array: [22.482809, 22.483868, 22.516142, 22.540028, 22.539999, 22.543324, 22.546701, 22.568577, 22.583838, 22.584484, 22.587116, 22.588093, 22.584112, 22.586754]

Jadavpur-Howrah Route-1 longitude array: [88.383324, 88.375473, 88.366486, 88.365417, 88.365417, 88.366112, 88.361488, 88.370148, 88.349525, 88.349297, 88.339439, 88.336143, 88.334045, 88.322624]

Jadavpur-Howrah Route-2 latitude array: [22.482809, 22.484537, 22.483122, 22.493059, 22.508566, 22.551325, 22.55154, 22.559755, 22.56529, 22.569489, 22.573973, 22.574648, 22.584063, 22.587116, 22.588093, 22.584112, 22.586754]

Jadavpur-Howrah Route-2 longitude array: [88.383324, 88.373436, 88.35318, 88.345139, 88.345421, 88.349121, 88.349159, 88.350578, 88.351578, 88.35305, 88.353638, 88.351456, 88.349709, 88.339439, 88.336143, 88.334045, 88.322624]

Lets say we enter 3 reviews, one for each route of jadavpur-howrah in order(route-0 then route-1 then route-2) as follows:

Route-0 review: [5,5,4,1,3] (from Tab-3)

Route-1 review: [4,5,3,2,4] (from Tab-4)

Route-2 review: [5,5,5,1,2] (from Tab-5)

Table 3 Database after route-0 review input

Path String	Number Of Reviews	Average Review Factors
22.540028:88.365417::22.539999:88.365417	1	[5.0, 5.0, 4.0, 1.0, 3.0]
22.54063:88.334686::22.575369:88.301804	1	[5.0, 5.0, 4.0, 1.0, 3.0]
22.588057:88.319809::22.586754:88.322624	1	[5.0, 5.0, 4.0, 1.0, 3.0]
22.542913:88.366081::22.540934:88.34127	1	[5.0, 5.0, 4.0, 1.0, 3.0]
22.482809:88.383324::22.483868:88.375473	1	[5.0, 5.0, 4.0, 1.0, 3.0]
22.540934:88.34127::22.54063:88.334686	1	[5.0, 5.0, 4.0, 1.0, 3.0]
22.539999:88.365417::22.542913:88.366081	1	[5.0, 5.0, 4.0, 1.0, 3.0]
22.575369:88.301804::22.592461:88.31295	1	[5.0, 5.0, 4.0, 1.0, 3.0]
22.483868:88.375473::22.516142:88.366486	1	[5.0, 5.0, 4.0, 1.0, 3.0]
22.592461:88.31295::22.591572:88.320549	1	[5.0, 5.0, 4.0, 1.0, 3.0]
22.516142:88.366486::22.540028:88.365417	1	[5.0, 5.0, 4.0, 1.0, 3.0]
22.591572:88.320549::22.588057:88.319809	1	[5.0, 5.0, 4.0, 1.0, 3.0]

Table 4 Database after route-0 and route-1 review input

Path String	Number Of Reviews	Average Review Factors
22.540028:88.365417::22.539999:88.365417	2	[4.06, 5.0, 3.06, 1.94, 3.94]
22.54063:88.334686::22.575369:88.301804	1	[4.06, 5.0, 3.06, 1.94, 3.94]
22.543324:88.366112::22.546701:88.361488	1	[4.0, 5.0, 3.0, 2.0, 4.0]
22.588057:88.319809::22.586754:88.322624	1	[4.06, 5.0, 3.06, 1.94, 3.94]
22.584484:88.349297::22.587116:88.339439	1	[4.0, 5.0, 3.0, 2.0, 4.0]
22.583838:88.349525::22.584484:88.349297	1	[4.0, 5.0, 3.0, 2.0, 4.0]
22.483868:88.375473::22.516142:88.366486	2	[4.06, 5.0, 3.06, 1.94, 3.94]
22.539999:88.365417::22.543324:88.366112	1	[4.0, 5.0, 3.0, 2.0, 4.0]
22.568577:88.370148::22.583838:88.349525	1	[4.0, 5.0, 3.0, 2.0, 4.0]
22.516142:88.366486::22.540028:88.365417	2	[4.06, 5.0, 3.06, 1.94, 3.94]
22.546701:88.361488::22.568577:88.370148	1	[4.0, 5.0, 3.0, 2.0, 4.0]
22.584112:88.334045::22.586754:88.322624	1	[4.0, 5.0, 3.0, 2.0, 4.0]
22.542913:88.366081::22.540934:88.34127	1	[4.06, 5.0, 3.06, 1.94, 3.94]
22.587116:88.339439::22.588093:88.336143	1	[4.0, 5.0, 3.0, 2.0, 4.0]
22.482809:88.383324::22.483868:88.375473	2	[4.06, 5.0, 3.06, 1.94, 3.94]
22.540934:88.34127::22.54063:88.334686	1	[4.06, 5.0, 3.06, 1.94, 3.94]
22.539999:88.365417::22.542913:88.366081	1	[4.06, 5.0, 3.06, 1.94, 3.94]
22.575369:88.301804::22.592461:88.31295	1	[4.06, 5.0, 3.06, 1.94, 3.94]
22.592461:88.31295::22.591572:88.320549	1	[4.06, 5.0, 3.06, 1.94, 3.94]
22.588093:88.336143::22.584112:88.334045	1	[4.0, 5.0, 3.0, 2.0, 4.0]
22.591572:88.320549::22.588057:88.319809	1	[4.06, 5.0, 3.06, 1.94, 3.94]

Table 5 Database after route-0 and route-1 and route-2 review input

Path String	Number Of Reviews	Average Review Factors
22.540028:88.365417::22.539999:88.365417	3	[4.81, 5.0, 4.62, 1.19, 2.38]
22.54063:88.334686::22.575369:88.301804	1	[4.81, 5.0, 4.62, 1.19, 2.38]
22.543324:88.366112::22.546701:88.361488	2	[5.0, 5.0, 5.0, 1.0, 2.0]
22.588057:88.319809::22.586754:88.322624	1	[4.81, 5.0, 4.62, 1.19, 2.38]
22.584484:88.349297::22.587116:88.339439	2	[5.0, 5.0, 5.0, 1.0, 2.0]
22.583838:88.349525::22.584484:88.349297	2	[5.0, 5.0, 5.0, 1.0, 2.0]
22.483868:88.375473::22.516142:88.366486	3	[4.81, 5.0, 4.62, 1.19, 2.38]
22.539999:88.365417::22.543324:88.366112	2	[5.0, 5.0, 5.0, 1.0, 2.0]
22.568577:88.370148::22.583838:88.349525	2	[5.0, 5.0, 5.0, 1.0, 2.0]
22.516142:88.366486::22.540028:88.365417	3	[4.81, 5.0, 4.62, 1.19, 2.38]
22.546701:88.361488::22.568577:88.370148	2	[5.0, 5.0, 5.0, 1.0, 2.0]
22.584112:88.334045::22.586754:88.322624	2	[5.0, 5.0, 5.0, 1.0, 2.0]
22.542913:88.366081::22.540934:88.34127	1	[4.81, 5.0, 4.62, 1.19, 2.38]
22.587116:88.339439::22.588093:88.336143	2	[5.0, 5.0, 5.0, 1.0, 2.0]
22.482809:88.383324::22.483868:88.375473	3	[4.81, 5.0, 4.62, 1.19, 2.38]
22.540934:88.34127::22.54063:88.334686	1	[4.81, 5.0, 4.62, 1.19, 2.38]
22.539999:88.365417::22.542913:88.366081	1	[4.81, 5.0, 4.62, 1.19, 2.38]
22.575369:88.301804::22.592461:88.31295	1	[4.81, 5.0, 4.62, 1.19, 2.38]
22.592461:88.31295::22.591572:88.320549	1	[4.81, 5.0, 4.62, 1.19, 2.38]
22.588093:88.336143::22.584112:88.334045	2	[5.0, 5.0, 5.0, 1.0, 2.0]
22.591572:88.320549::22.588057:88.319809	1	[4.81, 5.0, 4.62, 1.19, 2.38]

5.4.3 Route recommendation results

Now suppose user selects Quest Mall as source and Sailen Manna Stadium as destination; even though it is not explicitly reviewed by any user, as it is already a part of jadvapur-howrah route; we intend to recommend the correct route according to the user's choice of review factors.

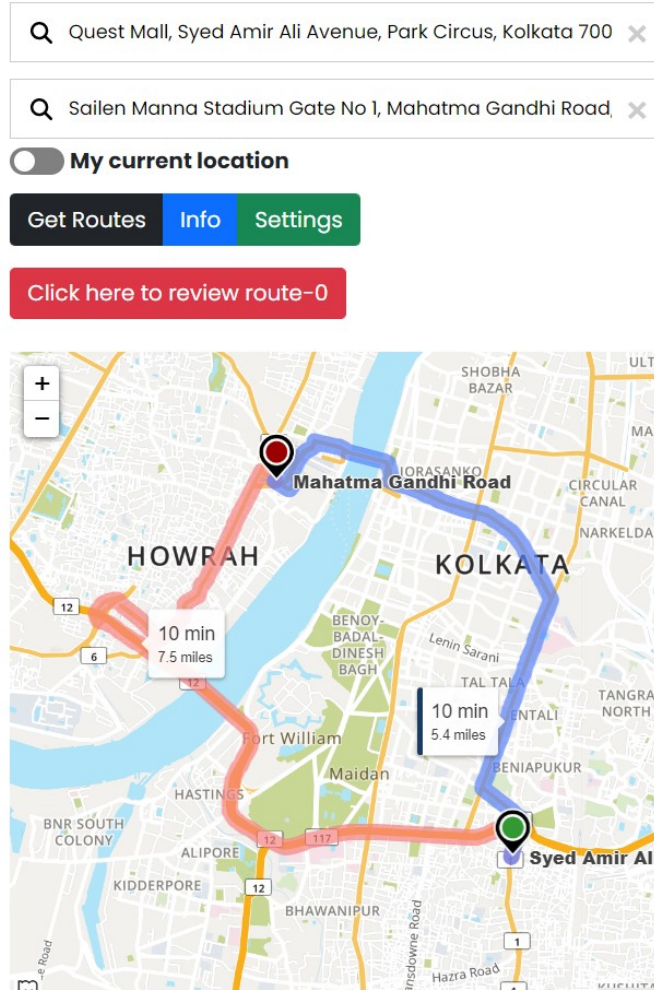


Fig. 22 Routes between Quest Mall and Sailen Manna Stadium

Quest Mall-Sailen Manna Stadium Route-0 latitude array:

[22.538733, 22.540028, 22.539999, 22.543324, 22.546701, 22.568577, 22.583838, 22.584484, 22.586926, 22.58317, 22.581377, 22.581305, 22.582247]

Quest Mall-Sailen Manna Stadium Route-0 longitude array:

[88.36515, 88.365417, 88.365417, 88.366112, 88.361488, 88.370148, 88.349525, 88.349297, 88.340424, 88.33741, 88.337372, 88.337082, 88.33548]

Quest Mall-Sailen Manna Stadium Route-1 latitude array:

[22.538733, 22.540028, 22.539999, 22.542913, 22.540934, 22.54063, 22.563448, 22.565554, 22.565746, 22.568069, 22.568422, 22.563938, 22.583414, 22.582247]

Quest Mall-Sailen Manna Stadium Route-1 longitude array:

[88.36515, 88.365417, 88.365417, 88.366081, 88.34127, 88.334686, 88.318321, 88.315002, 88.314507, 88.313805, 88.315804, 88.3209, 88.333672, 88.33548]

RC, WS, WL, TC, FG stands for Road Condition, Women Safety, Water Logging, Traffic Congestion, Frequent Gas Stations in Tab-8

Table 6 Available data for Route-0 Quest Mall-Sailen Manna Stadium

Path String	Is available?	Average Review Factors
22.538733:88.36515::22.540028:88.365417	No	N/A
22.540028:88.365417::22.539999:88.365417	Yes	[4.81, 5.0, 4.62, 1.19, 2.38]
22.539999:88.365417::22.543324:88.366112	Yes	[5.0, 5.0, 5.0, 1.0, 2.0]
22.543324:88.366112::22.546701:88.361488	Yes	[5.0, 5.0, 5.0, 1.0, 2.0]
22.546701:88.361488::22.568577:88.370148	Yes	[5.0, 5.0, 5.0, 1.0, 2.0]
22.568577:88.370148::22.583838:88.349525	Yes	[5.0, 5.0, 5.0, 1.0, 2.0]
22.583838:88.349525::22.584484:88.349297	Yes	[5.0, 5.0, 5.0, 1.0, 2.0]
22.584484:88.349297::22.586926:88.340424	No	N/A
22.586926:88.340424::22.58317:88.33741	No	N/A
22.58317:88.33741::22.581377:88.337372	No	N/A
22.581377:88.337372::22.581305:88.337082	No	N/A
22.581305:88.337082::22.582247:88.33548	No	N/A

Table 7 Available data for Route-1 Quest Mall-Sailen Manna Stadium

Path String	Is available?	Average Review Factors
22.538733:88.36515::22.540028:88.365417	No	N/A
22.540028:88.365417::22.539999:88.365417	Yes	[4.81, 5.0, 4.62, 1.19, 2.38]
22.539999:88.365417::22.542913:88.366081	Yes	[4.81, 5.0, 4.62, 1.19, 2.38]
22.542913:88.366081::22.540934:88.34127	Yes	[4.81, 5.0, 4.62, 1.19, 2.38]
22.540934:88.34127::22.54063:88.334686	Yes	[4.81, 5.0, 4.62, 1.19, 2.38]
22.54063:88.334686::22.563448:88.318321	No	N/A
22.563448:88.318321::22.565554:88.315002	No	N/A
22.565554:88.315002::22.565746:88.314507	No	N/A
22.565746:88.314507::22.568069:88.313805	No	N/A
22.568069:88.313805::22.568422:88.315804	No	N/A
22.568422:88.315804::22.563938:88.3209	No	N/A
22.563938:88.3209::22.583414:88.333672	No	N/A
22.583414:88.333672::22.582247:88.33548	No	N/A

Table 8 Route selection on user's choice

User's choice	Resultant sum for Route-0	Resultant sum for Route-1	Route selected
RC	4.97	4.81	Route-0 selected
WS	5.0	5.0	Route-0 selected
WL	4.93	4.62	Route-0 selected
TC	1.03	1.19	Route-1 selected
FG	2.07	2.38	Route-1 selected
RC+WS	9.97	9.81	Route-0 selected
RC+WL	9.9	9.43	Route-0 selected
RC+TC	6.0	6.0	Route-0 selected
RC+FG	7.04	7.19	Route-0 selected
WS+WL	9.93	9.62	Route-0 selected
WS+TC	6.03	6.19	Route-1 selected
WS+FG	7.07	7.38	Route-1 selected
WL+TC	5.96	5.81	Route-0 selected
WL+FG	7.0	7.0	Route-0 selected
TC+FG	3.1	3.57	Route-1 selected
RC+WS+WL	14.9	14.43	Route-0 selected
RC+WS+TC	11.0	11.0	Route-0 selected
RC+WS+FG	12.04	12.19	Route-1 selected
RC+WL+TC	10.93	10.62	Route-0 selected
RC+WL+FG	11.97	11.81	Route-0 selected
RC+TC+FG	8.07	8.38	Route-1 selected
WS+WL+TC	10.96	10.81	Route-0 selected
WS+WL+FG	12.0	12.0	Route-0 selected
WS+TC+FG	8.1	8.57	Route-1 selected
WL+TC+FG	8.03	8.19	Route-1 selected
RC+WS+WL+TC	15.93	15.62	Route-0 selected
RC+WS+WL+FG	16.97	16.81	Route-0 selected
RC+WS+TC+FG	13.07	13.38	Route-1 selected
RC+WL+TC+FG	13.0	13.0	Route-0 selected
WS+WL+TC+FG	13.03	13.19	Route-1 selected
RC+WS+WL+TC+FG	18.0	18.0	Route-0 selected

5.4.4 Results and discussion

From Tab-6, we have Route-0 average of average review factors of the available parts:

[4.97, 5.0, 4.93, 1.03, 2.07] (6 out of 12 part available)

From Tab-7, we have Route-1 average of average review factors of the available parts:

[4.81, 5.0, 4.62, 1.19, 2.38] (4 out of 13 part available)

RC,WS,WL,TC,FG are in this order in average review factors array in Tab-6 and Tab-7 and Tab-8. The resultant sum is made by adding the elements of average review factors array in Tab-6 and Tab-7 (elements are taken as it is in the rounded format and not the original places of decimal for the sake of simplicity). Here we select a route if its resultant sum is more than the other. In case the sum is same, route-0 is selected.

From Tab-8, we can clearly see which route (route-0 or route-1) will be suggested to user based on the resultant sum we calculated for route-0 and route-1 from Tab-6 and Tab-7

6 Conclusion

In today's world data is power. The more data we have, the better our algorithms will work and we can do a lot of things with it. Crowdsourcing is a evolving future concept which can be used in all domain and the experience of different users can be used to judge the same or upcoming scenarios. The ideas of different human brains can solve a complex problem easily and accurately. Improving the quality of the crowdsourced data is the main objective of the thesis, where various machine learning algorithm does the trick for us. This is the future as the more data we collect from the user, the more will be able to know the road conditions of the entire world. People can choose their roads as they will know the different factors of the routes given by other users. This will lead to less traffic accidents, people can avoiding traffic jams, women safety will greatly increase and also people will know the locations of important places like public toilets, public wifi or gas filling stations.

6.1 Scope of improvement

There are few areas in this thesis where there are some scope of improvements. They are:

- Privacy of the user's data is not ensured as users can see other user's data in the public page of the website. Malicious users can use this data to locate the user or sell user's data to malicious organizations.
- We didn't use any two-step verification/protection in the sign in/sign out system and may be prone to hacking.
- There is no forgot password options enabled, so if an user forget his/her password; they cannot login again.
- Machine Learning Classifier based validation can be more decorated and variety of classification can be included with the existing module

6.2 Future aspects

Crowdsourcing(after data validation) can be a trend setter as well as a show stealer in near future. Day by day data generation is increasing at a high rate. At the same time data is becoming less reliable and losing its trustworthiness. Data is becoming noisy. To get rid of all such data failure data validation can be a great help. Crowdsourcing can be used in every sector including banking, sports, health care, insurance, consumer trade etc. We can see various app that use crowdsourced data like google maps, glassdoor, Quora,etc which will grow and improve in future. These apps will further motivate new organization to power their logic using crowdsourcing and this will result in improved functionality of lot of important day to day apps.

References

1. Kongyang Chen, Guang Tan, Mingming Lu, and Jie Wu. Crsm: a practical crowdsourcing-based road surface monitoring system. *Wireless Networks*, 22(3):765–779, 2016.
2. Qatrunnada Ismail, Tousif Ahmed, Apu Kapadia, and Michael K Reiter. Crowd-sourced exploration of security configurations. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 467–476, 2015.
3. Fátima Leal, Benedita Malheiro, and Juan Carlos Burguillo. Prediction and analysis of hotel ratings from crowd-sourced data. In *World Conference on Information Systems and Technologies*, pages 493–502. Springer, 2017.
4. Damianos Gavalas and Michael Kenteris. A web-based pervasive recommendation system for mobile tourist guides. *Personal and Ubiquitous Computing*, 15(7):759–770, 2011.
5. Bin Guo, Yi Ouyang, Tong Guo, Longbing Cao, and Zhiwen Yu. Enhancing mobile app user understanding and marketing with heterogeneous crowdsourced data: a review. *IEEE Access*, 7:68557–68571, 2019.
6. Benjamin M Good and Andrew I Su. Crowdsourcing for bioinformatics. *Bioinformatics*, 29(16):1925–1933, 2013.
7. <https://www.mongodb.com/mern-stack>.
8. auth0.com. <https://jwt.io/introduction>.
9. <https://openbase.com/js/bcryptjs/documentation>.
10. <https://developer.tomtom.com/>.
11. <https://developer.mapquest.com/documentation/>.
12. <https://react-bootstrap.github.io/>.
13. Chaya Bakshi. <https://levelup.gitconnected.com>, Apr 2022.
14. https://en.wikipedia.org/wiki/simple_linear_regression, May 2022.
15. <https://www.saedsayad.com>.
16. https://en.wikipedia.org/wiki/polynomial_regression, Jun 2022.
17. https://www.varsitytutors.com/hotmath/hotmath_help/topics/exponential-regression.
18. Great Learning Team. <https://www.mygreatlearning.com/blog/introduction-to-multivariate-regression/>, Jun 2022.
19. <https://en.wikipedia.org/wiki/crowdsourcing>, Jun 2022.
20. Mary K. Pratt and Chris Gonsalves. <https://www.techtarget.com/searchcio/definition/>, Jul 2017.
21. Marshall Hargrave. <https://www.investopedia.com/terms/c/crowdsourcing.asp>, Jul 2021.
22. <https://www.braineet.com/blog/crowdsourcing>, May 2022.
23. <https://www.mturk.com/worker>.
24. <https://stackoverflow.com/>.
25. <https://en.wikipedia.org/wiki/glassdoor>, Apr 2022.
26. <https://www.quora.com/>.
27. <https://www.google.com/maps/>.
28. <https://www.wikipedia.org/>, Jun 2022.
29. Author Akshay Upadhyay. <https://www.igismap.com/haversine-formula-calculate-geographic-distance-earth/>, Jun 2018.
30. <https://reactjs.org/>.
31. <https://expressjs.com/>.
32. Node.js. <https://nodejs.org/en/about/>.
33. <https://www.mongodb.com/docs/>.
34. <https://www.npmjs.com/package/dotenv>.
35. <https://www.npmjs.com/package/cors>.
36. <https://www.heroku.com/>.