TIEVS – CLASSIFIED ADVERTISING ENHANCED USING MACHINE LEARNING TECHNIQUES

Final Group Report

2021-195

BSc (Hons) in Information Technology Specializing in Information Technology

Department of Information Technology

Sri Lanka Institute of Information Technology

Sri Lanka

October 2021

TIEVS – CLASSIFIED ADVERTISING ENHANCED USING MACHINE LEARNING TECHNIQUES

Final Group Report

2021-195

Dissertation Submitted in Partial Fulfillment of the Requirements for the BSc (Hons) in Information Technology Specializing in Information Technology

Department of Information Technology

Sri Lanka Institute of Information Technology
Sri Lanka

October 2021

DECLARATION

We declare that this is our own work and this dissertation1 does not incorporate without acknowledgment any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of our knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgment is made in the text.

Also, we hereby grant to Sri Lanka Institute of Information Technology, the non-exclusive right to reproduce and distribute our dissertation, in whole or in part in print, electronic, or another medium. We retain the right to use this content in whole or part in future works (such as articles or books).

Name	Student ID	Signature
R.A.D. Prathapa	IT18122060	
K.S.S Bandaranayake	IT18113532	
Ravihari J.M.S	IT18089400	Ravihari
Madhushani S.K.A.K.I	IT18082548	

The above candidates have carried out research for the bachelor's degree Dissertation under my supervision.

Signature of the supervisor: Date:

Abstract

The scarce use of tangible periodicals led to a consistently soaring popularity of online classified advertising. Nevertheless existing platforms retain complications. Most recommendation systems are built with conventional technologies that are less scalable, less accurate, and having high latency processes. Moreover, customers find it tiring when clarifying a reliable, accurate price value for items they are trying to sell through the classified advertising system. Additionally, austere validation techniques to identify and prevent fraudulent content or images from being published in the systems have been neglected. Therefore, authors have inaugurated a superior classified advertising system, Tievs, as a solution, by appraising said predicaments. It wields a flexible, process-simplifying, concurrency-induced recommendation breakthrough implemented from Universal Sentence Encoding incurred Natural Language Processing and Deep Learning routines. Furthermore, an innovative price prediction system having a supervised regression-based ensemble model forged ensuing a comparative study having excellent accuracy in proactively predicting item prices as to cater hassles faced by customers, was satisfied. Light Gradient Boosting classifier-driven fake description analysis along with a Convolution Neural Network powered figure deception recognition system was introduced, which gained prodigious precision with righteous clarity in fraud detection and prevention. Hence, the proposed solution's objective of surpassing former classified advertising systems in delivering customer's necessities using the most lucrative time-saving, human-centric and error-preventive approaches was accomplished. It was affirmative by the positively responded questionnaire regulated among prospective users by the authors.

Key Words –Classified Advertising, Ensemble Learning, Fraud Detection, Machine Learning, Optimized Recommendation, Price Prediction

ACKNOWLEDGEMENT

We would like to thank our supervisor, Ms. Manori Gamage (Lecturer - Faculty of Computing SLIIT), and co-supervisor, Ms. Suriyaa Kumari (Lecturer - Faculty of Computing SLIIT), for their recommendations and assistance in writing this work. And we would want to convey our gratitude for their consistent support, encouragement, and enthusiasm during the research, as well as their intelligent suggestions for improving the 'TIEVS' system's efficiency and productivity.

Ms. Wasanthi Bandaranayake is also credited with expressing her positive and bad experiences with online classified ad platforms, as well as hypothetical adjustments that should have been included in a more advanced technological element. Her recommendations and ideas paved the way for a new line of inquiry in the domains of information technology and business that has yet to be explored and researched in the current literature.

Mr. Austin Reese, Mr. Baris Dincer, and the Kaggle team deserve special commendation for developing and making analysis-foundational datasets publicly available to all scholars and tech enthusiasts. Moreover, we are grateful to all lecturers of Sri Lanka Institute of Information Technology, research teammates, and all family members and friends for their continued support and encouragement.

TABLE OF CONTENTS

Declaration i

Abstract ii

Acknowledgement iii

Table of Contents iv

List of Figures vi

List of Tables ix

List of Abbreviations x

List of Appendices xi

1 Introduction 1

1.1 Background 1

1.2 Literature Survey 5

1.3 Research Gap 10

1.4 Research Problem 12

1.5 Research Objectives 16

2 Methodology 17

2.1 High-level System Design 17

2.2 Data Source, Tools & Technologies 18

2.3 Commercialization Aspects of The Product 19

2.4 Recommendation System 21

2.4.1 Recommender high-level system overview 21

2.4.2 Recommender implementation 23

2.4.3 Recommender testing 31
2.5 Price Prediction System 33
2.5.1 Prediction model high-level system overview 33
2.5.2 Prediction model implementation 35
2.5.3 Prediction model testing 42
2.6 Fraud Content Detection System 45
2.6.1 Fraud context detection high-level system overview 45
2.6.2 Fraud context detection implementation 46
2.6.3 Fraud context detection testing 52
2.7 Fraud Image Detection System 54
2.7.1 Fraud image detection high-level system overview 54
2.7.2 Fraud image detection implementation 55
2.7.3 Fraud image detection testing 57
3 Results & Discussion 59
3.1 Results 59
3.2 Research Findings 75
3.3 Discussion 80
3.4 Summary of Each Student's contribution 84
4 Conclusion 88
References 91
Appendices 95
Appendix A: User Feedback Questionnaire for Tievs 95
Appendix B: Plagiarism Report from Turnitin 100

LIST OF FIGURES

Page

Figure 1.1 Craigslist Revenue Vs. Newspaper Ad Sales [1] 1
Figure 1.2 Use of online classified ads sites doubled from 2005 -2009 [2] 2
Figure 1.3 Starting point when looking for used cars [3] 2
Figure 1.4 Research gap analyzation 11
Figure 1.5 Craigslist erroneous price clarification part 1 13
Figure 1.6 Craigslist erroneous price clarification part 2 13
Figure 1.7 Fraudulent content submission in Craigslist 14
Figure 2.1 High-Level system architectural design 17
Figure 2.2 Utilized tools & technologies 18
Figure 2.3 System diagram of recommendation system 21
Figure 2.4 Removal of null values from the description column 23
Figure 2.5 USE embedding creation process [23] 24
Figure 2.6 Imported dependencies for Google USE model 24
Figure 2.7 Regex function 25
Figure 2.8 Code for generating embeddings 25
Figure 2.9 Generated embeddings 26
Figure 2.10 Real-time dynamic database 26
Figure 2.11 Go lang cosine similarity calculation code 28
Figure 2.12 Cosine value calculation for all ads 29
Figure 2.13 Go API function for new embedding creation initiation 30
Figure 2.14 GenerateEmbeddins.py source code 30
Figure 2.15 Postman generate embeddings testing 32
Figure 2.16 Postman get similar ads testing 32
Figure 2.17 System diagram of the price prediction system 33
Figure 2.18 Dataset first five rows 35
Figure 2.19 One-Hot-Encoded categorical columns 37

Figure 2.20 Stacking regressor algorithm [24] 40
Figure 2.21 Flask API part 1 41
Figure 2.22 Flask API part 2 41
Figure 2.23 Postman API testing 43
Figure 2.24 Price prediction Loadmill test part 1 44
Figure 2.25 Price prediction Loadmill test part 2 44
Figure 2.26 Overall high-level system diagram of fraud context detection 45
Figure 2. 27 Implementation of “ham” “spam” word cloud visualization 47
Figure 2.28 Pseudocode stop words removal 48
Figure 2.29 Extraction of the base form of words using snowball stemming 48
Figure 2.30 Text extraction process using BOW 49
Figure 2.31 Highest Max_depth obtained from F1 score metric implementation 50
Figure 2.32 Best parameters for LightGBM model 50
Figure 2.33 Flask API fraudDetection() implementation 51
Figure 2.34 Flask API textVectorixer() implementation 51
Figure 2.35 User interface implementation for obtaining user inputs 52
Figure 2.36 Postman of fraudulent input - production mode testing result 53
Figure 2.37 Postman of legitimate input - development mode testing result 53
Figure 2.38 Overall high-level system diagram of fraud image detection 54
Figure 2.39 Train the best parameters to the CNN model 56
Figure 2.40 Shuffle the dataset 56
Figure 2.41 Implementation of vehicle and non-vehicle 57
Figure 2.42 Postman of non-fraudulent image detection 58
Figure 2.43 Postman of fraudulent image detection 58
Figure 3.1 Loadmill load test 59
Figure 3.2 Performance graphs 60
Figure 3.3 Loadmill test alternative view 61
Figure 3.4 Postman top similar ad result 62
Figure 3.5 Single classified ad view 63

Figure 3.6 Top similar classified advertisement view 63
Figure 3.7 Regressor comparison table 64
Figure 3.8 Top 4 regressor predicted values vs. actual values 66
Figure 3.9 Individual and ensemble model accuracy comparison 67
Figure 3.10 Individual and ensemble model mean score boxplot comparison 67
Figure 3.11 Predicted price visualization in the ad submission form 68
Figure 3.12 Confusion matrix visualization for LightGBM model 69
Figure 3.13 Accuracy levels of classifiers 70
Figure 3.14 Classification report of LightGBM classifier 70
Figure 3.15 Visualization of fraud warning as the ad posting 71
Figure 3.16 Model accuracy rate 72
Figure 3.17 DigitalOcean dashboard part 1 73
Figure 3.18 DigitalOcean dashboard part 2 73
Figure 3.19 Droplet SSH tunnel 74
Figure 3.20 Supervisorctl status 74
Figure 3.21 User preference for Tievs web-application recommendation 83
Figure 3.22 Effectiveness of each main component in Tievs 83
Figure 3.23 Overall user review for Tievs system 83
Figure 4.1 GTmetrix performance view part 1 88
Figure 4.2 GTmetrix performance view part 2 88

LIST OF TABLES

Page

Table 3.1 Classification Report of LightGBM classifier 71
Table 3.2 Convolution neural network Statistics 71
Table 3.3 IT18122060 Contribution Summary 84
Table 3.4 IT18113532 Contribution Summary 85
Table 3.5 IT18089400 Contribution Summary 86
Table 3.6 IT18082548 Contribution Summary 87

LIST OF ABBREVIATIONS

Abbreviation	Description
API	Application Programming Interface
CORS	Cross Origin Resource Sharing
CPU	Central Processing Unit
DAN	Deep Averaging Network
DT	Decision Tree
IaaS	Infrastructure as a Service
IT	Information Technology
JAM Stack	JavaScript, API, Markup Stack
JSX	JavaScript XML
KNN	K-Nearest Neighbors
MAE	Mean Absolute Error
ML	Machine Learning
MLP	Multilayer Perceptron
MSE	Mean Standard Error
RF	Random Forest
RMSE	Root Mean Square Error
RSE	Relative Squared Error
SVM	Support Vector Machine
TF-IDF	Term Frequency–Inverse Document Frequency
UI	User Interface
URL	Uniform Resource Locator
USE	Universal Sentence Encoder
WWW	World Wide Web
XGBoost	Extreme Gradient Boosting
XGBRF	Extreme Gradient Boosting Random Forest

LIST OF APPENDICES

Appendix	Description	Page
Appendix – A	User Feedback Questionnaire for Tievs	95
Appendix – B	Plagiarism Report from Turnitin	95

1 INTRODUCTION

1.1 Background

Presently, the utilization of online platforms such as Craigslist, Olx, Quikr, Carewale, CarDheko Facebook Marketplace, ikman.lk, etc., for classified advertisement interaction, has become increasingly popular compared to its predecessor periodicals (newspapers, leaflets, brochures, magazines, booklets), radio, and television communication channels. This inevitability, along with rapid technological growth and digitalization, has made it quite lucrative for both sellers and shoppers in their own endeavors whilst using the evolving marketplace of online classified ads. Newspaper revenues from advertising classifications are continuously declining as Internet classifications are growing, which is being depicted in Fig. 1.1[1]. Furthermore, due to many benefits, as Fig. 1.2 shows, from 2005 to 2009, the use of online classified ads sites has been doubled [2]. Some of those benefits can be known as high availability, cost-effectiveness, higher accessibility, interface attractiveness, time-saving, easy change management, convenient addition, and deletion of classified as necessary. Moreover, as shown in Fig. 1.3 [3], in the UK, more people prefer online websites rather than going directly to a car dealer when they need to buy a used car.

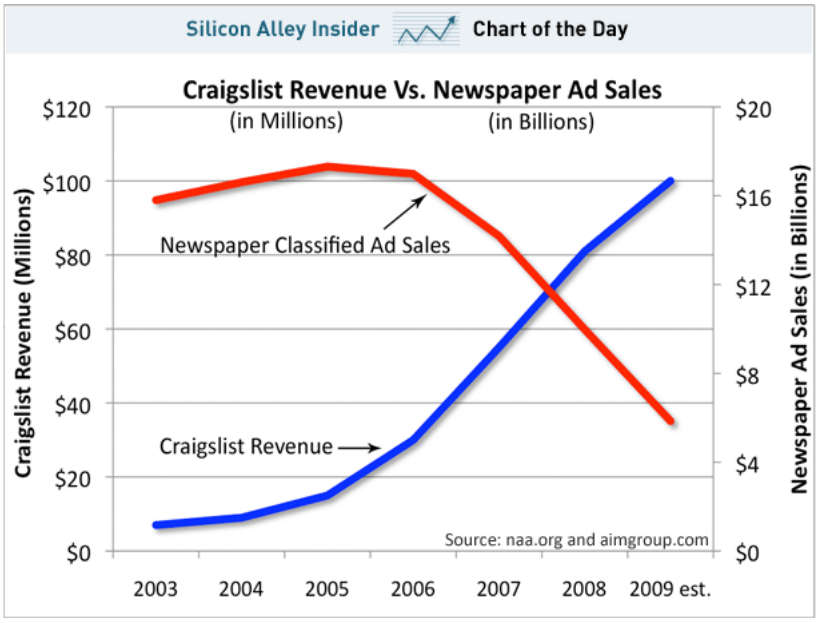


Figure 1.1 Craigslist Revenue Vs. Newspaper Ad Sales [1]

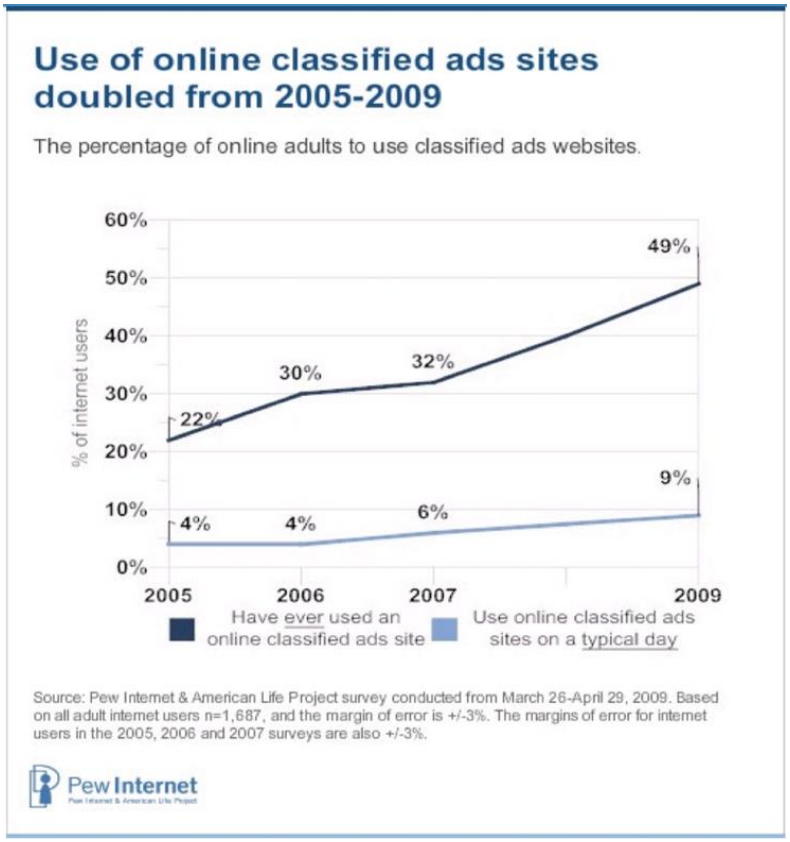


Figure 1.2 Use of online classified ads sites doubled from 2005 -2009 [2]

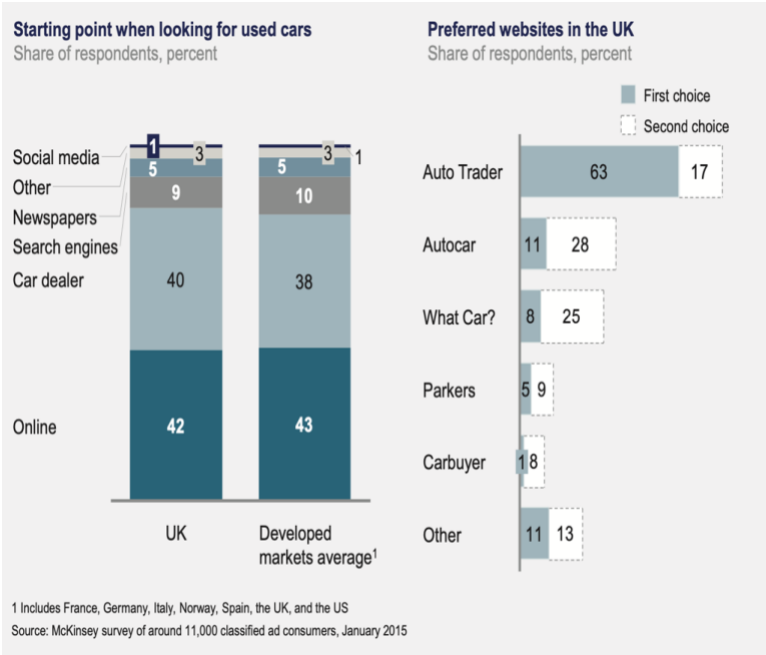


Figure 1.3 Starting point when looking for used cars [3]

Regardless of numerous implementations, a remarkably enhanced application that incorporates complex technologies to efficiently derive prominent functionalities that facilitate customer expectations and requirements has yet to arise. It is undoubtedly troublesome when interactions happen with an application that is not much accommodating to consumer needs appropriately.

When recommending specific ads for customers with regard to their preferences, most systems do not account for meticulous characteristics of the descriptions within the searching products. Existing systems have considered common approaches for recommendation such as Term Frequency – Inverse Document Frequency (TF-IDF) rather than investigating more suitable innovative methods, i.e., Google Universal Sentence Encoder (USE). Having less scalable, less accurate, high latency processes internally running using outdated common technologies degrades the overall system productiveness, causing a waste of time in finding the desired result, which eventually leads to high customer turnover rates as well.

Furthermore, it can be strenuous doing exhaustive research when clarifying the appropriate price for a product to be sold which reduces customer satisfaction and comfortableness. Usage of website crawlers within an application for target price realization is deemed idle due to their possible dependent server failures, incompatible or rigid attributes under consideration, failing requests from heavy network traffic, and future system augmentation immobilization.

Hence an advertising platform providing its own independent product price prediction mechanism is more convenient and currently obscure. Moreover, from an application developer's perspective, they need to implement strict program logic to identify and prevent fraudulent advertisements from being submitted either by accident or deliberately, using verification and validation techniques. If those ads were submitted, in due course, shoppers may view erroneous ads and lose confidence in using the application for their needs prompting the downgrade of the platform's reputation.

Current applications do not prioritize the above, let alone monitor and prevent them from occurring. Even more, many implemented classified advertising systems simply do not exhibit rich User Interfaces (UI s) for smooth functionality or promote quality user experience, in consonance with the latest trending scientific breakthroughs.

After scrutinizing the prevailing issues, the authors have investigated and analyzed several scientific approaches and have proposed a novel solution as an intelligent and advanced web application that incorporates Machine Learning (ML) and Deep Learning (DL) technology, mainly aiming to surpass present-day resembling competitors and provide customers with the finest user experience when browsing online classifieds.

The aforementioned aim was fulfilled by the achievement of an interactive and optimized recommendation system to serve ingenious methods to advocate preferred classifieds, a dynamic price prediction procedure for specific products so that customers are saved from the burden of clarifying prices themselves, and the identification and prevention of fraudulent content and images before ad submission, leading to higher reliability and reputation of the application as a whole. This research was conducted by focusing primarily on **used car classifieds** to maintain scope.

1.2 Literature Survey

Due to inadequate existing research specifically covering the overall proposed solution, previous studies vaguely allying with the main four components of this research are being discussed.

Product recommendation employing sundry algorithms either by content-based, collaborative, or hybrid methods for the movie, social networks, music, e-commerce systems, etc., were targeted by studies done so far rather than heeding for classifieds portals. Authors R. Singla et al. [4] have done a content-based movie recommender engine and utilized 'Distributed Bag of Words' version of Paragraph Vector (PV-DBOW) and Term Frequency methods quite exceptionally for movie plots/descriptions examination and generation of an inter-movie similarity score while Badriyah et al. [5] developed a system to suggest properties by exerting content-based filtering powered with the TF-IDF method.

Although admiring results have been obtained, [4]-[6] have overlooked an ideal technique that this paper bearded, which is USE, that acquire words and average them for the later transaction into a feedforward Deep Neural Network (DNN) to produce sentence embeddings quite efficiently [7]. Hence, the inferiority of PV-DBOW and TF-IDF techniques is comprehensible.

Author S. Shaikh [8] used a graph-based approach for recommendation in an e-commerce website that utilizes an overlap semantic approach to integrate recommendation and semantics where overlap value is calculated using the specific formula and saved in a graph format. Only items with a correlation value of more than 0.4 were considered for recommendation. However , it is unusable for classified ad portals since user-preferred characteristics such as votes, likes/dislikes, ratings, views, feedback, or comments are not generally provided. Furthermore, it induces the Cold Start problem, where information is unavailable for the recommendation engine to

handle newly introduced components efficiently. Therefore, our proposed system would not account for such attributes except for content-based features. Cosine-similarity fusion of these studies tempted this research's refinement in collaboration with the USE technique, which was not priorly investigated and exercised. Supervised learning-based ML algorithms were commonly used for price prediction, either by classification or regression. Ganesh et al. [9] selected Multiple Regression, Lasso Regression, and Regression Trees algorithms to produce predictive models and used One-way Analysis Of Variance with Post-Hoc test to determine error rate variances which were quite avant-garde. Lasso and Regression Trees were deemed better, with error rates less than 5%.

However, their results would have demeaned if a larger data sample was used to train and test than using only 804 records with 11 decisive features, being quite similar to Kuiper [10], who used a Multivariate regression model to predict numerical price values for 2005 General Motor cars using the same dataset extracted from pakwheels.com, encouraging the worth of variable selection techniques and analyzing multiple models' performances.

Using the same dataset, Noor et al.[11] implied that Multiple Linear Regression might outperform all with its prediction precision of 98%, though it's unreliable since only 1699 records and 3 features were considered from feature engineering, reducing forecast perfection.

In [12], a comparative study examined 3 regression-based models where the Gradient Boosting model gave an admirable accuracy having mean absolute error as 0.28 in comparison to Random Forest (RF) and Multiple Linear Regression. Their staggering 304,133 data records with 11 predictors were applauding for model efficiency. Regardless, ensemble techniques yielding better performances were not explored in theirs, unlike this research. Pal et al. [13] investigated Linear Regression and RF, with 3 notable but ill-suited forecasters and RF triumphantly conferred train and test

accuracies as 95.82% and 83.63%. Overfitting still seemed to linger in contemplation of the above values and despite cross-validation and grid search measures taken. One study [14] found RF classifier the best, having test accuracy of 83.08% with only 5 relevant features and withdrawing main ones as vehicle type, price, and transmission, surfacing a feature selection flaw as opposed to ours with 13 estimators.

Authors [9]-[14] utilized datasets scraped from e-commerce websites and/or from Kaggle. Hence they originate the notion that peculiar car price knowledge is futile and confirms the sufficiency of open datasets that inspired this study as well. Contradictorily, Mauritius car data from newspapers were used by Pudaruth [15] to predict prices of used cars using Multiple Linear regression, Naïve Bayes, K-Nearest Neighbors, and Decision Tree algorithms.

The classification accuracies were mediocre, i.e., below 70%, due to fewer car observations at about 90 records, while the inability of Naïve Bayes and Decision Trees to predict continuous variables was indisputably perceptible.

Assessments discussed [9]-[15] only analyzed performance statistics of prediction for several acclaimed models individually. Divergently, Gegic et al. [16] affirmed the higher potential in accuracy and precision of an ensemble model's prediction, assembling Artificial Neural Network, Support Vector Machine, and RF classifiers having solitary correction less than 50%, and stating their combined accuracy on test data as 87.38%. Although having more samples than 797 could have improved prediction power and 90:10 split probably lead to model overfitting that reduced generalization levels, their articulation of the ensemble model is outstanding and encouraged this research to audition the same using regressors. Sporadic attempts to distinguish and obviate online frauds have prevailed as Garg, and Nilizadeh [17] showed that vehicle frauds are prominent on craigslist in 30 locations in the United States (US). Approximately 1.7% of ads were detected as frauds incorporating indecorous keywords. Conversely, using a single indicator and having a limited sample

size made it inconvenient to extrapolate their observations to other sorts of advertising. Another study [18] was the first to perform a statistical evaluation of 2 datasets to detect spam pages. They asserted that spammers utilize templates to create spam content automatically.

As a result, even if specific words vary from one page to another, they will have an equal number of words. Two hundred sample-sized hosted web pages, including a minimum of 10 pages with no word count variation, were spammed 55%. Tran et al. [19] developed a classifier to detect spam on classified websites, and they contended that additional features such as occurrences of URLs or e-mail addresses are required to outperform traditional content or Natural Language Processing-based (NLP) classifiers. The abovementioned characteristics have incorporated domain-specific features such as the current market price of advertised items. They improved their F-1 scores by more than 50%. However, relying on external sources were hard to generalize characteristics which conveyed a significant drawback in their approach.

Two simple yet effective histogram and encoding-based detection methods were proposed by Yuanfang Guo et al. [20] for fake colorized images, which exhibit a proper execution against many state-of-the-art colorization approaches. Yet, their findings elucidate the declining effectiveness of their solution when the testing and training images are generated utilizing various colorization methods or datasets. Likewise, [21] introduced a novel rapid image matching technique, which constructs the k-d tree and uses better BBF algorithms to substitute the Linear algorithm and evaluate the performance of the amended method through testing. Consonantly, few web-based scam detection tools as Ads-Guard [22] assist novice users in ascertaining the legitimacy of certain ads, helping law enforcement inspections avert online classified ad crimes. Application Program Interfaces (APIs) and Image Recognition were used in [22] to estimates the likelihood of a particular classified being a scam by pasting its URL into the tool.
Nevertheless, it had a few limits since it could only accept Craigslist advertisements.

Fruitful precautions on fraud content/ images have not been occupied within persisting classified advertising systems, thus precipitated this research to contrive a desirable measure to conceivably increase application confidentiality altogether.

Two simple yet effective histogram and encoding-based detection methods were proposed by Yuanfang Guo et al. [18] for fake colorized images, which exhibit a proper execution against many state-of-the-art colorization approaches. Yet, their findings elucidate the declining effectiveness of their solution when the testing and training images are generated utilizing various colorization methods or datasets.

Likewise, [19] introduced a novel rapid image matching technique, which constructs the k-d tree and uses better BBF algorithms to substitute the Linear algorithm and evaluate the performance of the amended method through testing. Consonantly, few web-based scam detection tools as Ads-Guard [20] assist novice users in ascertaining the legitimacy of certain ads, helping law enforcement inspections avert online classified ad crimes.

Application Program Interfaces (APIs) and Image Recognition were used in [20] to estimates the likelihood of a particular classified being a scam by pasting its URL into the tool. Nevertheless, it had a few limits since it could only accept Craigslist advertisements. Fruitful precautions on fraud content/ images have not been occupied within persisting classified advertising systems, thus precipitated this research to contrive a desirable measure to conceivably increase application confidentiality altogether.

1.3 Research Gap

The below illustrated Fig. 1.4 depicts the summarized comparison between our Tievs application and other prevailing dominant platforms, which are namely, Craiglist, KSL.com, Quikr, and ikman.lk. As clearly perceived, there were significant gaps in existing systems that should be filled through an innovative approach, which in turn, this research was able to accomplish thoroughly.

What's more, is that scientific or sophisticated experimentation or research has not been done so far as to address the prevailing problems within the classified advertising domain. What type of mechanisms has been used and how complicated these mechanisms can range, or are there any other innovative approaches than the already being used ones were some topics or aspects that were not explored at all. This notion prompted our experimentations to initiate primarily so that our ultimate results and findings can confidently be the pioneer in the classified advertising research area.

Therefore, in order to justify and articulate the research gap of this study, it was necessary to consider the prevailing systems due to the absence and scarcity of particularly all-inclusive preceding research. Thus, the following explanation is the emphasis provided regarding the research gap comparison table from the online classified advertising web applications perspective.

The utilization of complex AI and ML-related technologies has been neglected in the past systems implemented. Hence, scalable and cost-efficient recommendation engines built using the ingenious model Google USE have not been undertaken before. This concurrency enabled, optimized resource utilization induced model was investigated meticulously by this study, which was never to be seen in other platforms. Dynamic price prediction through an ensemble ML model and result visualization through a rich user-centric web design was implemented as well, which, again, other systems failed to incorporate. Although some platforms have used web crawlers, they

further provide inconvenience through dependency and control related matters. A higher number of car attributes for advertising and price evaluation purposes were taken into consideration by our Tievs application than most of the other systems.

The other noteworthy aspect that immensely assisted in diminishing the predominant research gap was that the dynamic identification and prevention of both fraudulent content and images through Tievs upon the advertisement submission into the system. These were essential functionalities that conventionally should be provided for customers, especially through a classified advertising portal, though such measures were ignored substantially in the existing applications. These critical research gaps, which were distinctly apparent, were finally eliminated up to a commendable level through the inauguration of this research project's final product.

Figure 1.4 Research gap analyzation

1.4 Research Problem

Even though it is distinctly apparent that online classified advertising platforms have already been implemented by numerous parties in recent years, there is yet to be a highly ameliorated application that incorporates complex machine learning technology to enhance the prominent functionalities to facilitate customer expectations and requirements efficiently.

It is undoubtedly troublesome when people have to interact with an application that is not much user-centric and accommodating towards their needs, (like most available applications nowadays) considering their busy schedules. Most applications provide the customers only with similar items for the current item search, but they would not be optimized for particular customer needs specifically, considering into depth characteristics of the searching product such as color type, manufactured year, price ranges, etc.

There can be instances where one can preconfigure the necessary characteristics prior to searching the relevant product, although those appliances can be particularly difficult to be utilized by elderly user groups and non-technical individuals, who can be defined as a large part of the target audience of classified advertising. Hence it will evidently cause a waste of time in finding the desired result, leading to a high customer turnover rate as well.

For customers who utilize those platforms to post their relevant advertisements targeting other customers, it can be strenuous when clarifying the accurate price for the product to be sold. Some even fall into misconceptions either accidentally or deliberately and define an unrealistic price value for the item advertisement such as what is being shown in Fig. 1.5 and 1.6, where customers have faultily deemed prices like $0 for their car classified advertisements. These kinds of incidents should be thoroughly distinguished Consequently, those customers will be doing exhaustive research themselves, to find out what is the exact selling price of other similar level

and type products on sale, which takes a huge amount of time out of their schedule again and as a whole, reduce user satisfaction level and comfortableness provided by the application. User-friendly classified advertising platforms that provide the price predictions for selling products are obscure as a whole as well.

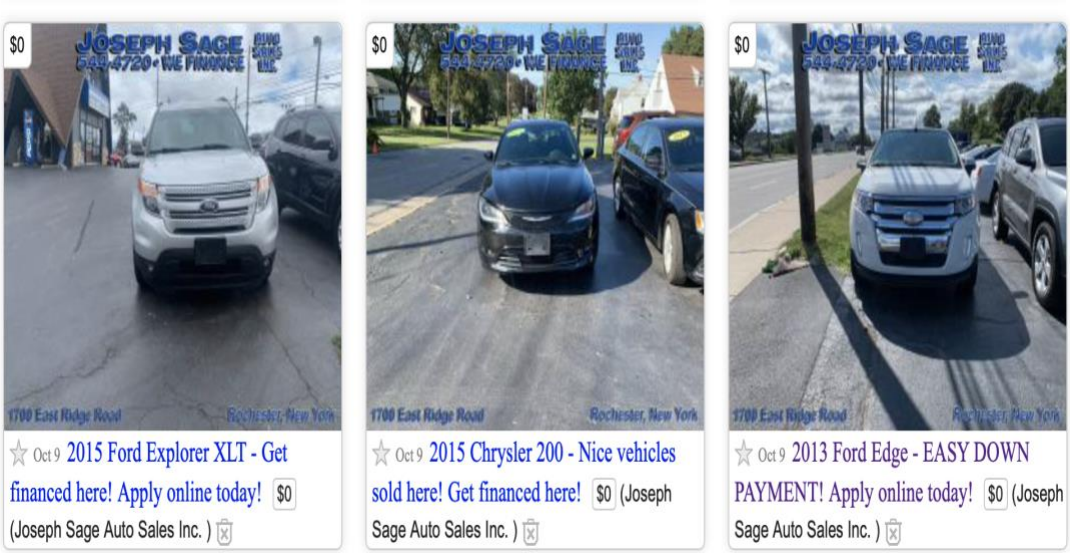


Figure 1.5 Craigslist erroneous price clarification part 1

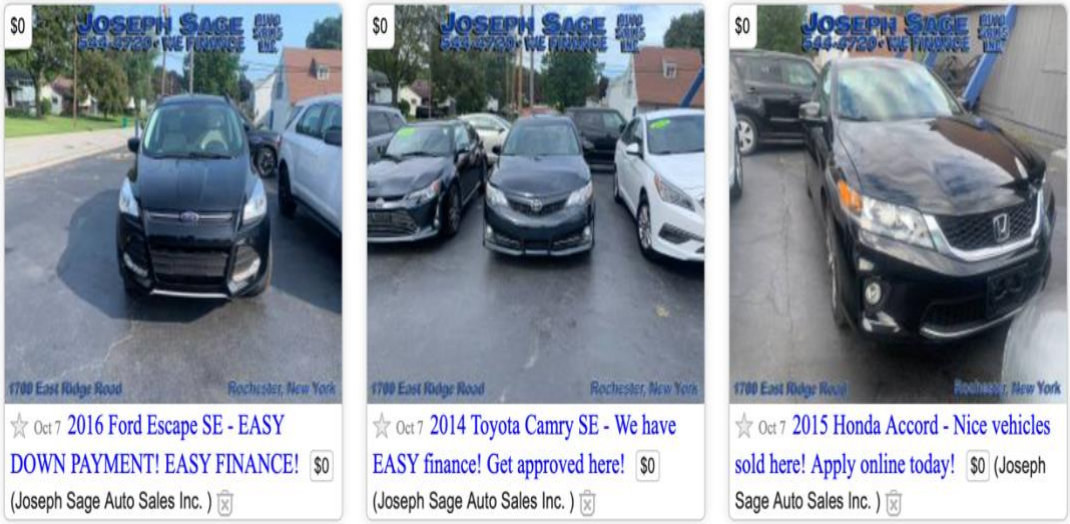


Figure 1.6 Craigslist erroneous price clarification part 2

In addition, from the applications' developers and owners' point of view, they have to implement strict program logic to identify or prevent fraudulent advertisements being

posted (fake images/ spam descriptions) accidentally or deliberately by users of the system, through verification and validation processes internally.

This will require inspecting posts separately and meticulously, even after the ad being posted to the customers and if so, customers will view those fraud ads and lose confidence in using the platform for their needs, enabling loss of application reputation as well. A fraud content submitted example in Craigslist is shown below by Fig. 1.7

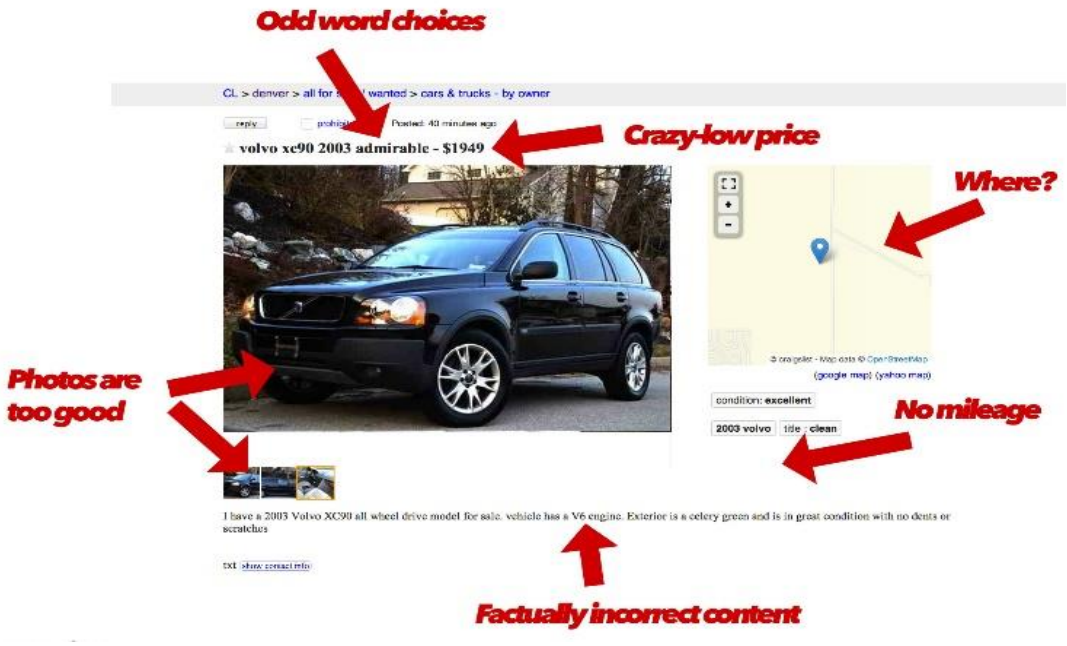


Figure 1.7 Fraudulent content submission in Craigslist

For customers who utilize those platforms to post their relevant advertisements targeting other customers, it can be strenuous when clarifying the accurate price for the product to be sold. Some even fall into misconceptions either accidentally or deliberately.

Then those customers will be doing exhaustive research themselves, to find out what is the exact selling price of other similar level and type products on sale, which takes a huge amount of time out of their schedule again and as a whole, reduce user

satisfaction level and comfortableness provided by the application[2]. User-friendly classified advertising platforms that provide the price predictions for selling products are obscure.

In addition, from the applications' developers and owners' point of view, they have to implement strict program logic to identify or prevent fraudulent advertisements being posted (fake images/ spam descriptions) accidentally or deliberately by users of the system, through verification and validation processes internally.

This will require inspecting posts separately and meticulously, even after the ad being posted to the customers and if so, customers will view those fraud ads and lose confidence in using the platform for their needs, enabling loss of application reputation as well.

Most classified advertising products currently in use do not prioritize detecting and preventing fake advertisement images or characteristic details from being submitted, let alone inspect them internally and inform the respected users who are in the process of submitting the advertisement.

Even more, many implemented online classified advertising systems, simply do not exhibit rich user interfaces for smooth functionality or promote quality user experience and user-friendliness, in consonance with the latest trending technologies.

1.5 Research Objectives

Main Objective

The rudiment objective of this research was providing a SMART, systematic, and rich user-friendly solution to enable interested parties to post their advertisements efficiently and accurately without any inconveniences, in addition to providing an optimized, interactive search facility within the system, to search and find items to buy, according to specific customer requirements, seamlessly and productively. The overall net result from the system is to surpass its predecessor systems in delivering and catering to customer's necessities, using the most optimal, cost-effective, timesaving, human-centric and error preventive approaches that can be carried out and accomplished by an advanced online classified advertising platform.

Sub Objectives

The following specific sub-objectives need to be attained in order to fulfill the above-mentioned primary objective of the research.

1. Developing an optimized item recommendation process, using a convenient machine learning model, to provide personalized content to customers in an interactive manner.
2. Implementing a dynamic price predicting system to recommend a selling price value for an item of a specific type (such as Cars), using a predictive machine learning algorithm.
3. Utilizing machine learning appliances, proactively detecting, and preventing counterfeit advertising content information from being published in the online application.
4. Detection of fake, ill-matched, conflicting or unsuitable images within the advertisement form, and displaying cautious warnings, using image processing techniques.

2 METHODOLOGY

2.1 High-level System Design

Following Fig. 2.1 displays the high-level system diagram having all the crucial components of the Tievs system as a whole.

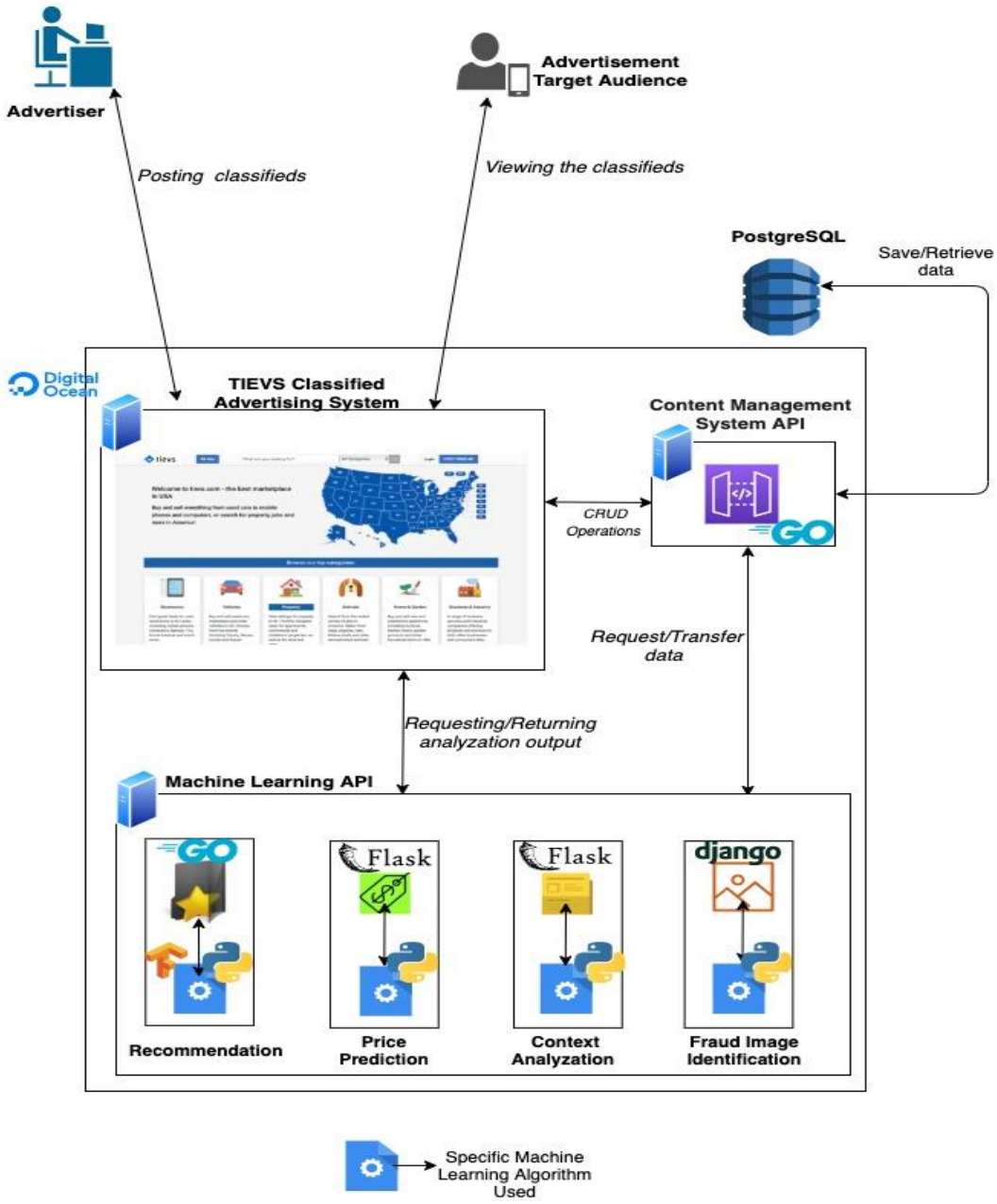


Figure 2.1 High-Level system architectural design

2.2 Data Source, Tools & Technologies

Two openly available Kaggle datasets were utilized for obligatory ML model rudiments. The 'Used Cars Dataset' sustaining 441,396 used car records covering about 40 brands having 25 unique attributes was scraped 2 months before this research inception from an American classified advertisement portal, namely 'Craigslist' since Tievs primitively targeted the US market. The 'Vehicle Detection Image Set' assisted the vehicle image recognition and fraud prevention strategy.

Programming languages and frameworks as Python (Django, Flask/Quart), Angular (Typescript), Golang (Echo framework), and PHP were used with supporting libraries like Pandas, Keras, glob, NumPy, Scikit-learn, TensorFlow, Pyplot, Angular Material, etc. for ML model, front-end app, and model/Content Management System API developments. Jupyter Notebook, Anaconda Navigator, Visual Studio Code, PyCharm, WebStorm, and Nova Editor were used as technical tools while the database server amalgamated PostgreSQL. The Authors incorporated Postman to test REST APIs and the Supervisorctl system to revive those upon hosting failures. Following Fig. 2.2 concisely illustrates the used technologies and tools-related icons.

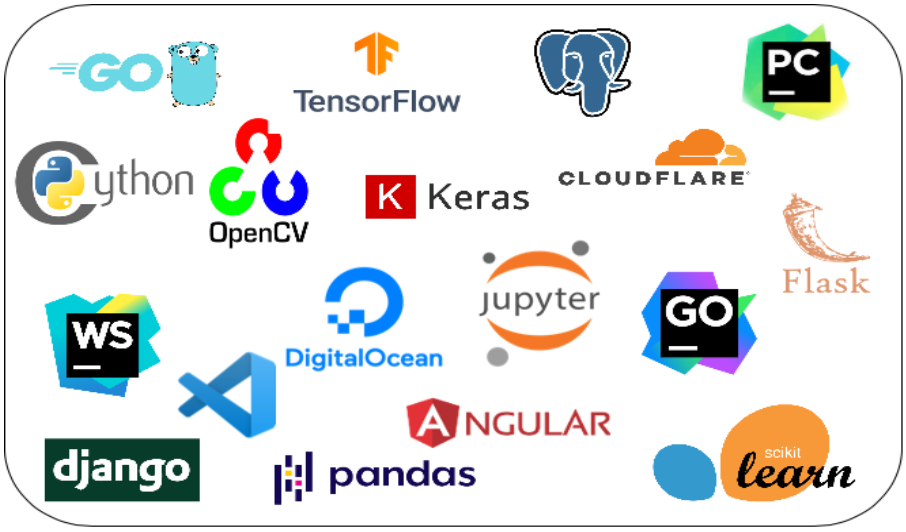


Figure 2.2 Utilized tools & technologies

2.3 Commercialization Aspects of The Product

By utilizing the developed recommendation engine, startup companies who are pursuing e-commerce web app developments would gain benefits since this engine can be used even without comprehensive previously attained user data and information. Additionally, this infers that the cold start problem would be lifted and resolved since user likes, dislikes, comments, feedback, browsing history, ratings, and votes are not accustomed to within the recommendation engine structure. Popular e-commerce giants such as Facebook and Uber have been penalized by various law enforcement for misusing customer personal data for processes such as recommendation itself. In that sense, this constructed recommendation engine would not have similar complications since sensitive consumer data is not being stored for any underlying procedure. Furthermore, real-time recommendation of classifieds will be provided since when a customer submits a new ad, the API executes the new embedding creation related to that, and that ad will instantly be accounted for cosine similarity calculation while paving the way for the overall recommendation process for any other customer who is browsing through the ads.

The incorporation of a price prediction system within an online car classified portal can lead to higher and faster sales of the cars that are being advertised since the tagged price values are much more convenient, reliable, and realistic when comparing with other similar cars within the market and the buyers are more obliged to make a complete purchase consequently. Hence the business revenue of the application as a whole will be increased as well due to the higher number of customer visits and utilization. Customers would not need to do extensive research when clarifying a price value by exiting the application, which results in a decrease in the bounce rate. They would simply receive an approximated price value at the application venue itself and will be thoroughly satisfied with the service that saved their valuable time out of their busy schedules. Moreover, the inclusion of such a price prediction system within the application will give it a towering competitive advantage among other parallel systems

that provide equivalent efforts in satisfying customer necessities. An advertising platform providing its independent item price prediction mechanism is more convenient from an application developer’s point of view due to intrinsic control authority over the price valuation, which is currently obscure since most similar platforms use website crawlers and face unnecessary third party dependencies, heavy network traffic, and future enhancement immobilization.

Fraudulent behaviors have invaded almost every area of internet business, from cybersecurity problems threatening end customers' privacy rights and e-commerce transactions to ransom attacks extorting significant sums of money from organizations. When fraud is not tackled proactively, it can have a negative influence on a company's profitability by diverting funds away from core operations and objectives, causing brand damage, and dissipating revenues. In extreme cases, it might even cost individuals their lives. The inclusion of a proactively recognized fraud detection system within a web-based car categorization platform could potentially lead to significantly higher and much more convenient total revenue of the advertised cars because such a software system has an enhanced reputation and truthfulness, which will be more beneficial and genuine when compared to other nearly equivalent cars on the market. And because of those circumstances, including the high rate of client liaison and utilization, the platform's advertising revenue will rise.

2.4 Recommendation System

2.4.1 Recommender high-level system overview

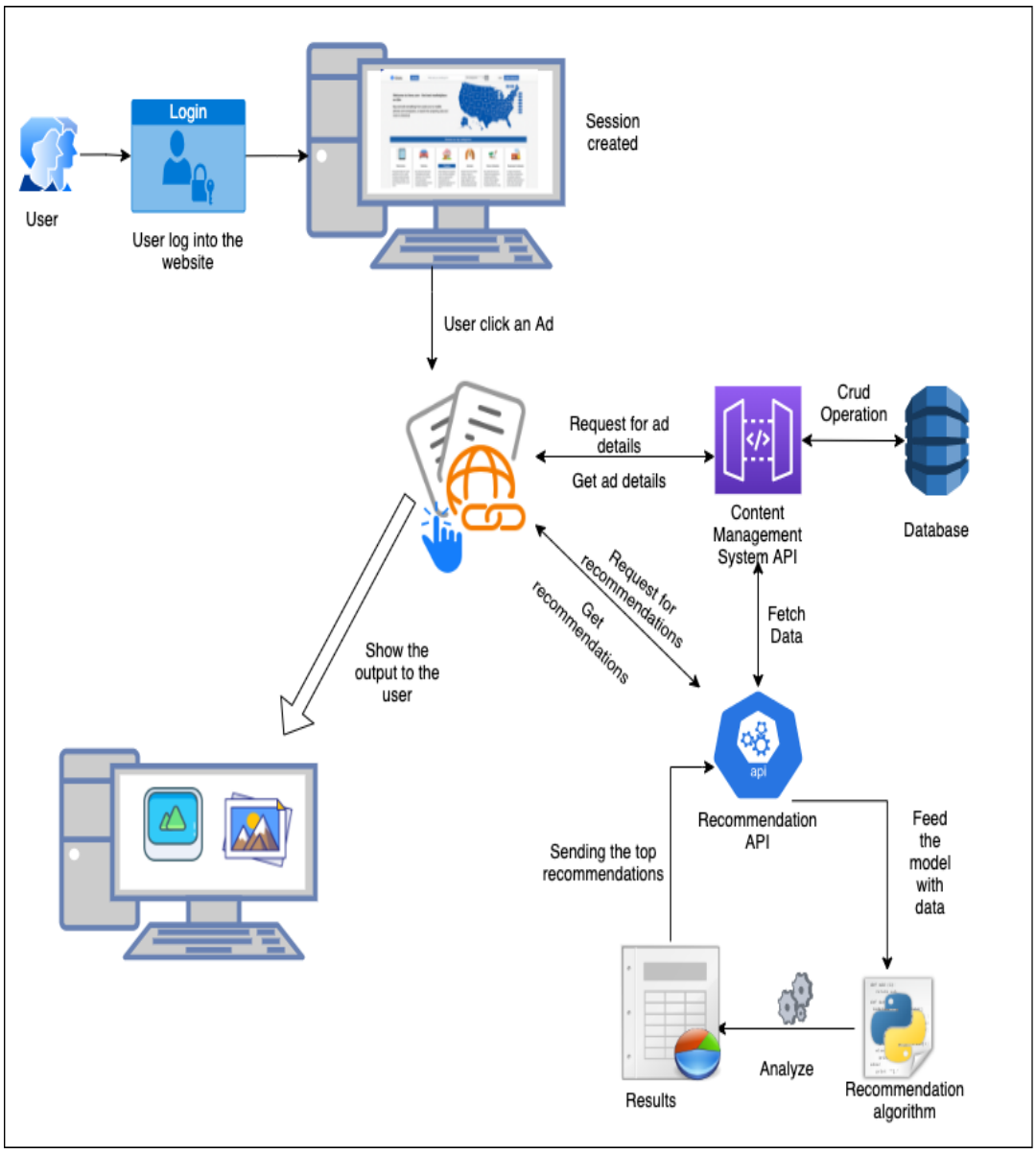


Figure 2.3 System diagram of recommendation system

Fig. 2.3, shown above, is the overall high-level diagram of the novel recommendation algorithm implemented using the TensorFlow imported pre-trained Google USE model is further trained with the incorporated training dataset. The description column is used for this task. The 512-vector embeddings are generated with regard to the

description column included context and the newly produced results are stores in the database table with a new column called ‘embeddings.’ The database system is utilized for the storage of all tables containing the embeddings in PostgreSQL. It is a fast performing, highly expandable, cross-platform database that is appropriate for the considered complex dataset importing. The assistance for various language support and its open-source feature is commendable. The re-trained model is then serialized to be combined with the machine learning (ML) API implemented using the simple, efficient, and reliable open source programming language, which is Go. Go is useful for complex but swift software development, which is exactly what is needed for this research component’s objective attainability. The API implemented handles two major procedures, which are the new embedding creation and storage for the newly published advertisements and the concurrent similarity calculation.

Both tasks are handled by the API parallelly; hence the infrastructure is built using robust Go language routines. The front-end application designed using the Angular framework is integrated with the Go API to provide the customers with the relevant preferred advertisements by filtering them all out quickly. When the customer clicks an ad(hyperlink) from the UI, the application calls the API, and it will request the ad details from the content management system, and also it will asynchronously request recommendations from the recommendation API.

The recommendation API will request other ads in that category from the content management system, and current Ad details will be analyzed using the optimized algorithm for the classifieds ad. The top recommendations will be given to the user as a response to the API request.

2.4.2 Recommender implementation

First, data preprocessing methods were applied to the dataset to make it more suitable and convenient for the concerning algorithm models to perform well. This process is more challenging and should be done with uttermost care if model accuracy is highly valued. Some of the simple processes that were executed in this stage upon the dataset would be,

- Deletion of records having null values as a majority.
- Deletion of the columns which are irrelevant for recommendation purposes.
- Processing and converting value types to match with the algorithm types being considered.
- Applying weights for attributes that contribute the most to the recommendation.

Following Fig. 2.4 shows how the null values are removed from the description column, which is the primary column that is considered for the content-based recommendation engine.

```
oList = oList.dropna(axis=0, subset=['description'])
```

Figure 2.4 Removal of null values from the description column

The main model used in the implementation is the Google universal sentence encoder (USE), which is a pre-trained model that is capable of encoding textual data formats into high dimensional 512 vectors or otherwise known as embeddings, which are basically numerical representations. It aims to feed transfer learning to other Natural Language Processing (NLP) tasks such as similarity calculation, relatedness, classification, and clustering. Google USE is publicly available in TensorFlow-hub. This pre-trained model is fundamentally trained on English sentences extracted from various web pages, Wikipedia's, discussion forums, and questions-answer pages. Variable-length input of textual formats can be entered, and the outcome would always

be a 512 vector embedding, and semantic textual similarity benchmark is phenomenal regarding Google USE. Below Figure. 2.5 extracted from [23] display how the Google USE convert a particular sentence to a 512-dimensioned vector or an embedding which can then be applied upon multiple tasks for learning.

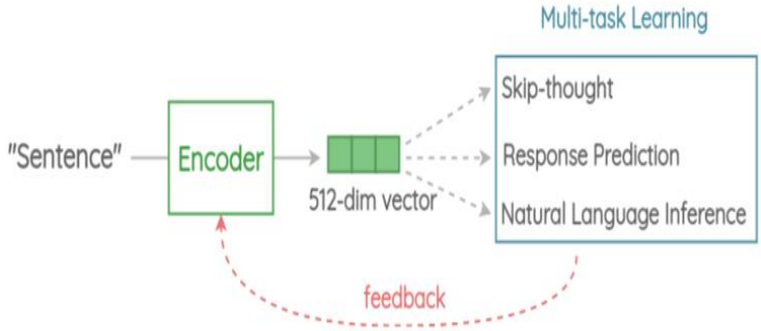


Figure 2.5 USE embedding creation process [23]

To startup with the process of the model formation, in Jupyter Notebook, all relevant python libraries and dependencies were imported using the necessary commands as shown below Fig. 2.6.

```
In [1]:
import numpy as np
import pandas as pd
import pickle
import os
from tqdm import tqdm_notebook
import tensorflow as tf
import tensorflow_hub as hub
from nltk import sent_tokenize
from tqdm import tqdm
from scipy import spatial
from operator import itemgetter
tqdm.pandas()
%matplotlib inline
```

Figure 2.6 Imported dependencies for Google USE model

Next, the id and description columns were extracted separately from the dataset, which is 18,035 car records, after handling null and irrelevant values. Then the description column was transferred through a complex regex function for further cleansing. Cleansing tasks include removing unnecessary string characters ($, # , @, *, %, &, =, -, etc.) from the textual content and converting all letters to lowercase. The regex function is shown in the Fig. 2.7 illustration below.

```python
import re
def clean_plot(text_list):
    clean_list = []
    for sent in text_list:
        sent = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-.:;<=>?@[\]^`{|}~"""), '',sent)
        sent = sent.replace('[]','')
        sent = re.sub('\d+',' ',sent)
        sent = sent.lower()
        clean_list.append(sent)
    return clean_list
```

Figure 2.7 Regex function

After that process, the sentences are then tokenized into tokens using Penn Treebank (PTB) tokenizer. Subsequently, the part where the encoder captures these sentences occur. Upon acquiring the sentences, the DAN encoders go through a set of learning processes and share in order to learn the sentence embeddings. Succeeding the training of the model, it is then capable of generating the appropriate 512-vectors or embeddings for the sentences provided, and this is claimed as the overall inference phase. Following Fig. 2.8 displays the functionality is written in python code to achieve the above-mentioned tasks.

```python
plot_emb_list = []
with tf.Graph().as_default():
    embed = hub.Module("use2")
    messages = tf.compat.v1.placeholder(dtype=tf.string, shape=[None])
    output = embed(messages)
    with tf.compat.v1.Session() as session:
        session.run([tf.compat.v1.global_variables_initializer(), tf.compat.v1.tables_initializer()])
        for plot in tqdm_notebook(cList['description']):
            sent_list = sent_tokenize(plot)
            clean_sent_list = clean_plot(sent_list)
            sent_embed = session.run(output, feed_dict={messages: clean_sent_list})
            plot_emb_list.append(sent_embed.mean(axis=0).reshape(1,512))
cList['embeddings'] = plot_emb_list
cList.head()
```

Figure 2.8 Code for generating embeddings

The new embedding is appended to the existing dataset table as a new column. The generated embeddings of the used car advertisements, based on their description column for recommending purposes, are shown below Fig. 2.9 for abstract referencing.

id	embeddings
18034	[[2.03763898e-02 1.07177123e-02 -2.25664061e-02 -2.61413530e-02 ¶ 2.08641626e-02 -1.50775816e-02 -7.69071106e-04 3.82859930e-02 ¶ -1.17898770e-02 2.03156229e-02 4.3193668.
18033	[[2.06584148e-02 1.27701228e-02 -2.86576021e-02 -2.15150993e-02 ¶ 8.88616312e-03 -1.11251259e-02 1.00778667e-02 1.61405616e-02 ¶ -1.82893742e-02 1.29344007e-02 2.9966147.
18032	[[2.56291702e-02 7.93562178e-03 -2.38398314e-02 -2.71451660e-02 ¶ 1.71665344e-02 -1.17667308e-02 5.49608516e-03 4.23929766e-02 ¶ -9.784886 61e-03 2.72825342e-02 4.2955435.
18031	[[0.03086213 0.01379801 -0.02561954 -0.02335528 0.01328001 -0.0046946 ¶ 0.01190553 0.02065733 -0.01245801 0.01539378 0.03318534 0.01597941 ¶ -0.01352 0.0300649 -0.0043.
18030	[[1.20909335e-02 3.47530544e-02 -4.57419269e-02 2.75885016e-02 ¶ -3.43366456e-03 -1.23503199e-02 -8.64222553e-03 2.04012934e-02 ¶ -4.24633808e-02 4.30987030e-03 4.0786758.
18029	[[2.02120300e-02 2.16376316e-02 -3.44675034e-02 3.17131169e-02 ¶ -2.48793680e-02 2.29074550e-03 -3.50628085e-02 1.67648550e-02 ¶ -3.51545215e-02 1.60991512e-02 2.3017883.
18028	[[2.13250928e-02 4.96837962e-03 -2.38715243e-02 -2.76052151e-02 ¶ 1.59605034e-02 -1.20248310e-02 9.98329744e-03 4.04776298e-02 ¶ -8.12987145e-03 2.20564473e-02 4.2974136.
18027	[[2.78053135e-02 4.91707772e-02 -3.15033533e-02 1.34862494e-02 ¶ -4.11784053e-02 1.90693792e-02 7.28234556e-03 6.61370605e-02 ¶ 5.44080511e-03 6.01768214e-03 5.6420352..
18026	[[1.73602942e-02 3.45736817e-02 -3.77876312e-02 5.01669459e-02 ¶ -3.66957597e-02 -2.76381224e-02 -2.01656185e-02 8.99270736e-03 ¶ -4.84031886e-02 -1.08224228e-02 2.566345..
18025	[[0.04071238 0.00578413 -0.03171218 0.02778312 -0.04102295 0.01951578 ¶ 0.00904648 0.00944059 -0.04779766 -0.02602539 0.02384756 0.01654739 ¶ -0.02107064 0.00534504 -0....
18024	[[3.47324014e-02 3.03216316e-02 -4.69955839e-02 9.70704667e-03 ¶ -5.72442785e-02 -9.71784536e-03 2.63886489e-02 3.20730396e-02 ¶ -4.81603071e-02 -1.84661113e-02 2.1931814.
18023	[[3.47324014e-02 3.03216316e-02 -4.69955839e-02 9.70704667e-03 ¶ -5.72442785e-02 -9.71784536e-03 2.63886489e-02 3.20730396e-02 ¶ -4.81603071e-02 -1.84661113e-02 2.1931814.
18022	[[3.20382155e-02 3.86673957e-02 -3.56526449e-02 1.35018015e-02 ¶ -5.50831035e-02 -4.54134773e-03 3.39207649e-02 4.43015993e-02 ¶ -3.00531890e-02 -1.83886085e-02 2.7248017.
18021	[[3.20382155e-02 3.86673957e-02 -3.56526449e-02 1.35018015e-02 ¶ -5.50831035e-02 -4.54134773e-03 3.39207649e-02 4.43015993e-02 ¶ -3.00531890e-02 -1.83886085e-02 2.7248017.
18020	[[2.19441950e-02 8.59771762e-03 -2.46456247e-02 -2.42545921e-02 ¶ 1.63284764e-02 -1.58562716e-02 9.92128905e-03 3.81903648e-02 ¶ -1.04426835e-02 2.37738211e-02 4.0082812.
18019	[[1.52319577e-02 2.12972593e-02 -2.84270532e-02 -2.03439593e-02 ¶ 6.12772815e-03 8.15529563e-03 -2.51867231e-02 2.33342089e-02 ¶ -1.48283420e-02 1.74199585e-02 3.8538076.
18018	[[3.47324014e-02 3.03216316e-02 -4.69955839e-02 9.70704667e-03 ¶ -5.72442785e-02 -9.71784536e-03 2.63886489e-02 3.20730396e-02 ¶ -4.81603071e-02 -1.84661113e-02 2.1931814.
18017	[[-0.01851632 0.02169145 -0.0467167 0.01609266 -0.02245841 -0.01796459 ¶ -0.01873061 0.04350098 -0.0400321 -0.02683197 0.04239906 0.01767137 ¶ -0.03534538 0.04697414 0.0..
18016	[[9.05754417e-03 7.55660236e-04 -2.82413587e-02 -2.49733124e-03 ¶ -4.03040573e-02 4.28719958e-03 -5.64676430e-03 -1.05990954e-02 ¶ -5.51046133e-02 -3.56607065e-02 5.837405..
18015	[[9.00049321e-03 -2.08281688e-02 -3.02733649e-02 -4.37804461e-02 ¶ -1.98548511e-02 -2.26608478e-04 3.94396624e-03 7.68789323e-04 ¶ 8.07478838e-03 -3.24053466e-02 5.5323228.
18014	[[0.02173067 0.00830612 -0.0230892 -0.02533236 0.02002969 -0.0162549 ¶ 0.00727531 0.04155011 -0.0122468 0.02800902 0.04332618 0.02605746 ¶ -0.02462476 0.02338159 0.015.
18013	[[2.30376385e-02 8.26652907e-03 -2.29314826e-02 -2.39872504e-02 ¶ 1.60423517e-02 -2.06205361e-02 3.47319432e-03 3.76510210e-02 ¶ -1.23902680e-02 1.81864928e-02 4.2343102.
18012	[[4.59751766e-03 -1.15375882e-02 -1.74644999e-02 -1.16414903e-02 ¶ 2.97048857e-04 -1.08487178e-02 3.01315915e-02 2.10997928e-02 ¶ -1.67044532e-02 1.98575179e-03 1.4961152.
18011	[[2.41547227e-02 4.04041167e-03 -2.35498529e-02 -2.70456281e-02 ¶ 1.37750525e-02 -1.15032196e-02 2.94287852e-03 4.07077335e-02 ¶ -1.49807399e-02 1.98525954e-02 4.1863314.
18010	[[2.48058401e-02 9.18141939e-03 -2.45841127e-02 -2.59925034e-02 ¶ 1.65073983e-02 -1.50917899e-02 7.90834054e-03 4.15613092e-02 ¶ -8.72146431e-03 2.71384288e-02 4.2127326.
18009	[[0.02334436 0.0101241 -0.02283718 -0.02101306 0.01790312 -0.01234174 ¶ 0.00623276 0.0407123 -0.01460126 0.02560007 0.04206478 0.02564385 ¶ -0.02524138 0.02286744 0.01.
18008	[[0.02394129 0.00956965 -0.02284844 -0.02105357 0.01763216 -0.01266219 ¶ 0.00652194 0.04059108 -0.01367969 0.02536413 0.04193789 0.02564543 ¶ -0.02531089 0.02285903 0.0.
18007	[[2.34397817e-02 1.06139369e-02 -2.27155592e-02 -2.09383499e-02 ¶ 1.78197008e-02 -1.24642616e-02 6.07001875e-03 4.07716408e-02 ¶ -1.44156534e-02 2.56108623e-02 4.1925635.

Figure 2.9 Generated embeddings

Since the embeddings are successfully created as of this stage, the next step is to export these embedding data into a database. The intention of this upload towards a database is to eliminate the cold start problem and to assist in generating a real-time, dynamic database that is able to create and update advertisement-related embeddings repeatedly as required, all the while adding new embedding entries into the database. Following Fig. 2.10 depicts the real-time dynamic database transactions.

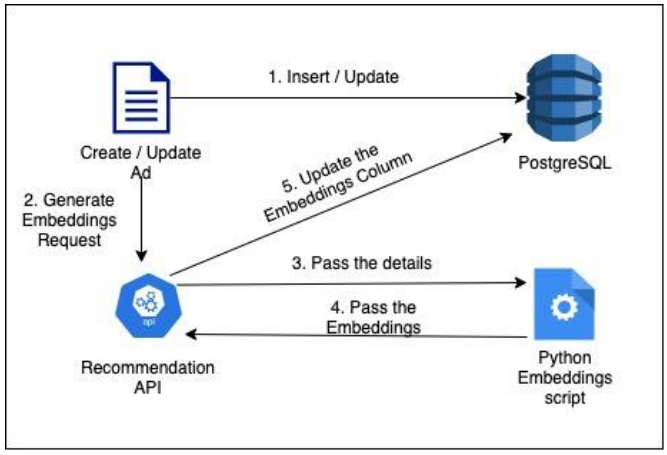


Figure 2.10 Real-time dynamic database

The main purpose of the API that should be incorporated along with the recommendation process was discovered, and it is divided into two key functions. Those two functions are creating new embeddings for the newly published or

submitted classified advertisements and calculation of the cosine similarity using those created and stored advertisement embeddings in order to present the consumers their preferred or otherwise recommended classifieds through the Tievs applications front-end view.

Real-time similarity calculating function

The basic concept behind what is happening when a consumer clicks on a particular ad and how the consequent recommendation process happens is explained below.

The API receives a GET request of the clicked ad, and it procures the specific id of that classified ad from the URL params. According to the id acquired, the state and embeddings of the specific advertisement were be fetched from the database. The next step is to request and obtain the other advertisement ids and embeddings from the database with respect to the previously clicked advertisement state. Those embeddings gained are in a string format.

Therefore, those formatted string embeddings are newly converted into a float array. After that, the major step of cosine similarity calculation was be initiated. In order to do that along with concurrency at a place, a normal Go lang map cannot be used for the task. Nonetheless, as a workaround, a specific map called sync. Map had to be exploited. Below Fig. 2.11 shows the Go language API code for cosine similarity calculation. The equation below (1) depicts the mathematical function which is responsible for calculating the similarity value.

$$similarity = \cos(\theta) = \frac{A \,.B}{||A||\,||B||} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\,\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{1}$$

```go
func Cosine(a []float64, b []float64) (cosine float64, err error) {
    count := 0
    length_a := len(a)
    length_b := len(b)
    if length_a > length_b {
        count = length_a
    } else {
        count = length_b
    }
    sumA := 0.0
    s1 := 0.0
    s2 := 0.0
    for k := 0; k < count; k++ {
        if k >= length_a {
            s2 += math.Pow(b[k], y: 2)
            continue
        }
        if k >= length_b {
            s1 += math.Pow(a[k], y: 2)
            continue
        }
        sumA += a[k] * b[k]
        s1 += math.Pow(a[k], y: 2)
        s2 += math.Pow(b[k], y: 2)
    }
    if s1 == 0 || s2 == 0 {
        return cosine: 0.0, errors.New( text: "Vectors should not be null (all zeros)")
    }
    return sumA / (math.Sqrt(s1) * math.Sqrt(s2)), err: nil
}
```

Figure 2.11 Go lang cosine similarity calculation code

By using the sync.Map, concurrency is applied to a ‘for each’ function to analyze and calculate the similarity value between the selected specific advertisement embedding along with the other embeddings within the array list of the database. The newly calculated similarity value and the id of those ads are then saved in the sync. Map array. At this stage, the concurrency part has been completed successfully. Now, for the further proceedings such as sorting, the sync. The map has to be converted into a conventional map since its primary usage of leveraging the concurrency aspect is done. Simultaneously, the similarity value was converted to a float type value as well. With the help of a specific function (sort. slice), the float converted values are sorted in descending order in the table. The highest or the top most similar records are considered here onwards. The below Fig. 2.12 is the Go language implemented API code that does all the above-mentioned procedures.

```go
go fmt.Printf( format: "DB fetch took %v\n", time.Since(dbStartTime))
go fmt.Printf( format: "All Entries fetch took %v\n", time.Since(timeallads))

start := time.Now()
//Sync Map
var m sync.Map
y := make(map[int]float64)
var wg sync.WaitGroup
wg.Add(len(ads))
for i := range ads {
    //Goroutine
    go func(i int) {
        defer wg.Done()
        //Calculate the cosine value
        cosineValue, _ := Cosine(ads[i].emb,sourceAd.emb)
        //Store it in the sync.map
        m.Store(int(ads[i].id),cosineValue)
    }(i)
}
wg.Wait()
//Converting back to normal map
m.Range(func(key, value interface{}) bool {
    var nValue float64
    nValue = value.(float64)
    y[key.(int)] = nValue
    return true
})
```

Figure 2.12 Cosine value calculation for all ads

Realtime embedding creation

The real-time embedding creation is also a crucial part of the API implemented. Previously, this thesis discussed that the embeddings are being created using the Google USE pre-trained model before deployment. After the production deployment, the same is happening thanks to the novel functionality introduced within the API. Following procedures happen during the new embedding generation.

1. The API calls the exec. command towards the local python file to run a script that assists in generating the embeddings from the generate embeddings function (Fig. 2.13 shows the source code of the API executing this part).
2. The id with other important details, especially the description column data of the specific ad under consideration, is sent.
3. The waiting process starts now for the Go lang API.
4. Meanwhile, the python file (GenerateEembeddings.py) containing the generate embeddings function having the complex regex function starts executing the above-mentioned process in the same way by starting to clean the description column, lowercase it, and tokenize it using Penn Treebank tokenizing.

5. Then the pre-trained TensorFlow model acquired the description sentences to create the embeddings using the DAN architecture.
6. After the new embedding is created, they are updated safely in the database table's specific id assigned embeddings column (Fig. 2.14 displays the GenerateEembeddings.py file's source code for this part with the function itself).
7. The process success or any faults that occurred are subsequently informed to the Go API with a response message.
8. Finally, the Go API's response about the success or failure of the embedding creation process is sent to the response expected party as well.

```go
func generateEmbeddings(c echo.Context) error {

    adID := c.QueryParam( name: "id")
    tableName := c.QueryParam( name: "state")

    cmd := exec.Command( name: "/opt/anaconda3/envs/RecSys/bin/python", arg...: "GenerateEmbeddings.py", "-i", adID, "-o", tableName)
    out, err := cmd.CombinedOutput()
    if err != nil { fmt.Println(err); }
    result := strings.Contains(string(out), substr: "Status::OK")
    var status = "Failed"
    if result == true{
        status = "Updated"
    }
    fmt.Println(string(out))

    return c.String(http.StatusOK, fmt.Sprintf(status))
}

func appendMap(m *[]Ad, ids int64, embS string){
    *m = append(*m, Ad{id: ids, emb: strFloatAr(strings.Fields(embS))})
}
```

Figure 2.13 Go API function for new embedding creation initiation

```python
tqdm.pandas()
import mysql.connector
from mysql.connector import Error


def clean_plot(text_list):
    clean_list = []
    for sent in text_list:
        sent = re.sub('[%s]' % re.escape("""!"#$%&'()*+,-.;:<=>?@[\]^`{|}~"""), '', sent)
        sent = sent.replace('[]', '')
        sent = re.sub('\d+', ' ', sent)
        sent = sent.lower()
        clean_list.append(sent)
    return clean_list


def generate_embed(plot):
    with tf.Graph().as_default():
        embed = hub.Module("use2")
        messages = tf.compat.v1.placeholder(dtype=tf.string, shape=[None])
        output = embed(messages)
        with tf.compat.v1.Session() as session:
            session.run([tf.compat.v1.global_variables_initializer(), tf.compat.v1.tables_initializer()])
            sent_list = sent_tokenize(plot)
            clean_sent_list = clean_plot(sent_list)
            sent_embed2 = (session.run(output, feed_dict={messages: clean_sent_list})).mean(axis=0).reshape(1, 512)
            return sent_embed2
```

Figure 2.14 GenerateEmbeddins.py source code

The recommendation engine consisting of both the Golang API and the Google USE included machine learning model has deployed DigitalOcean cloud server environment for 24/7 hour available access. DigitalOcean is an infrastructure as a service platform (IaaS) that is capable of swift and complex cloud deployments that run on multiple cloud servers simultaneously. The front-end application, which is called Tievs, was fundamentally built based on the Angular framework using Typescript as the programming language managing UI components.

2.4.3 Recommender testing

Initial functional testing was required to be executed upon the Golang implemented concurrency enabled API and imported trained Google USE model, which was deserialized and integrated inside the API itself. For that purpose of API testing, Postman was utilized due to its convenient capability to test HTTP/2 (HTTPS) requests through a graphical user interface having all necessary requirements regarding parameter types, request body types, and request types such as POST, PUT, GET, DELETE. Different types of responses can be acquired, which can further be validated as well. Requests can be saved with collection classification, and metrics such as time, size, status responses can be evaluated. Furthermore, by using Postman as an HTTP client, header, authorizations, and pre-request script management was made possible regarding the API to be tested. Fig. 2.15 displays the test done on the POST request of the API, which triggers the calculation and generation of the new embeddings according to a new advertisement submission sent as a table containing the vehicle type classified ad details. Fig. 2.16 shown below demonstrates how the deployed API server was tested by provoking a GET request for a specific car advertisement. The particular car id, which is assumed to be clicked by a customer in a real scenario, is passed using the URL parameters. The response is displayed as a JSON result indicating all essential attributes of an advertisement that is being recommended, such as price, year, model, manufacturer, and images.

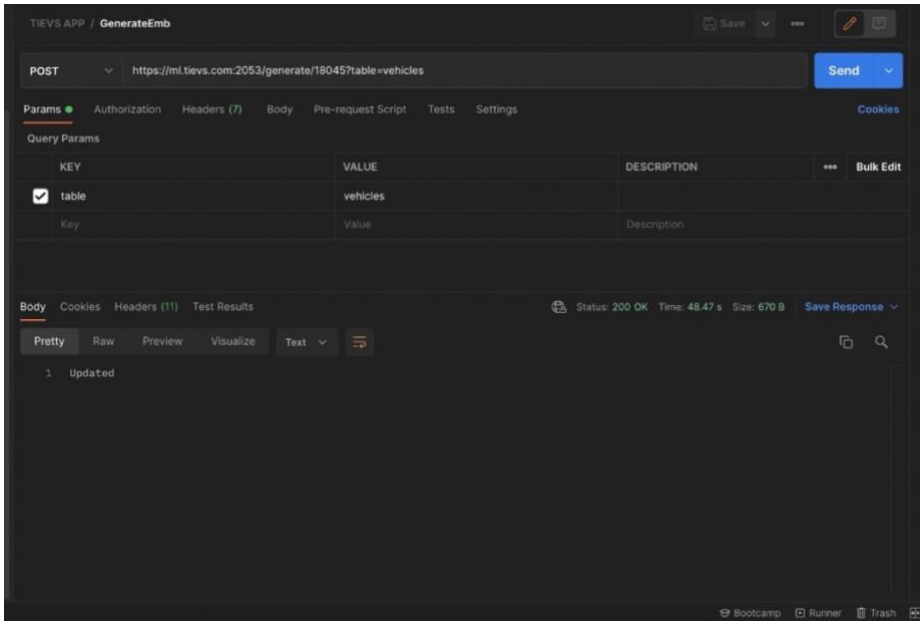


Figure 2.15 Postman generate embeddings testing

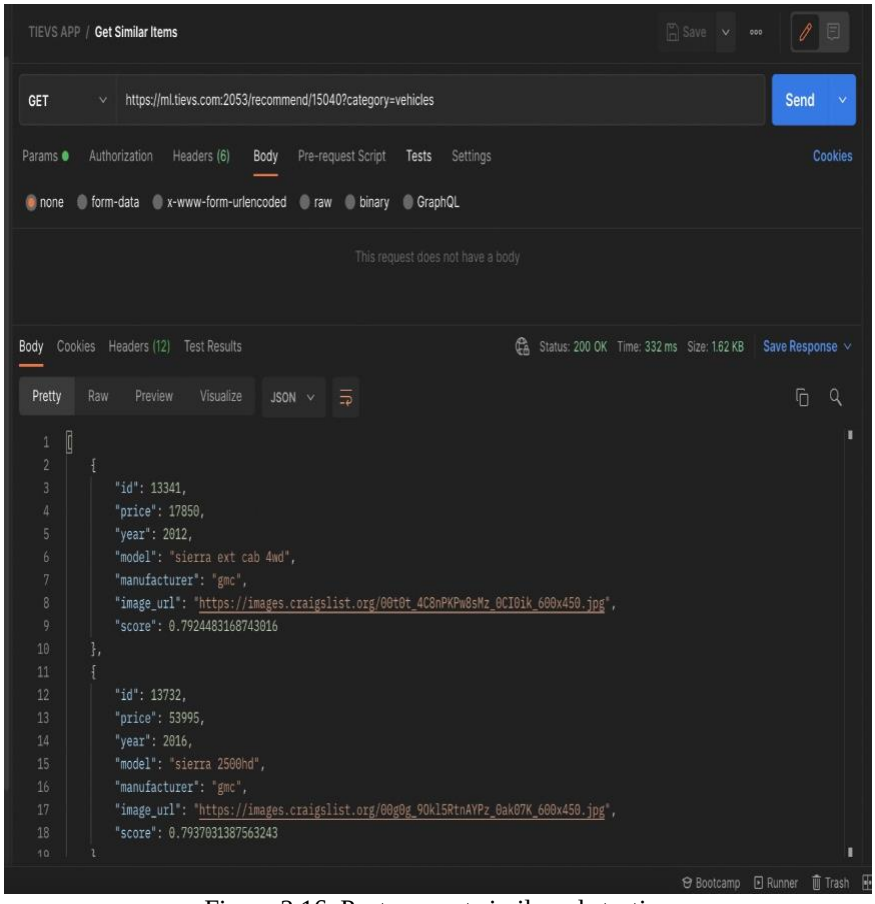


Figure 2.16 Postman get similar ads testing

2.5 Price Prediction System

2.5.1 Prediction model high-level system overview

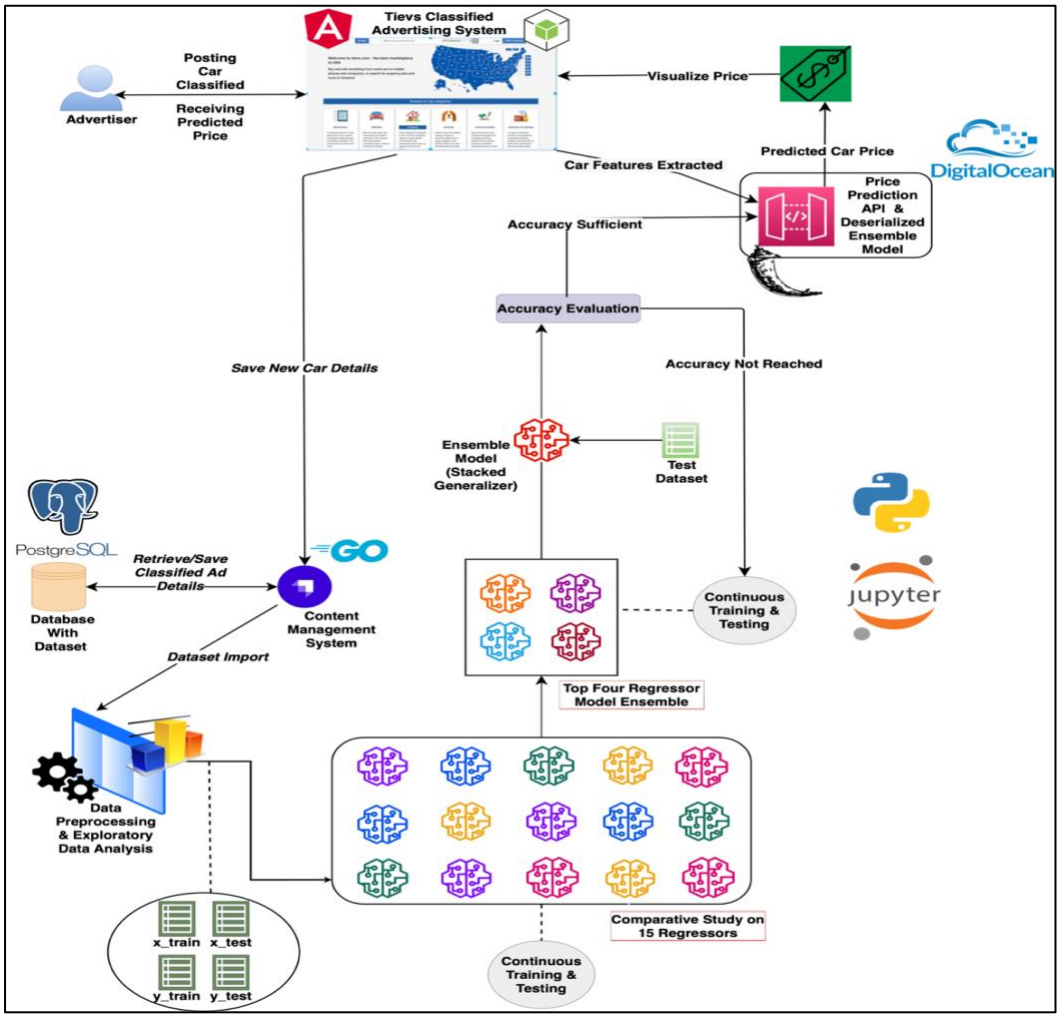


Figure 2.17 System diagram of the price prediction system

The above-shown Fig. 2.17 depicts the overall high-level system diagram of the proposed price prediction system that was built in order to provide a price valuation regarding the used cars that are being published in Tievs as car classifieds. First, the acquired dataset was imported from the PostgreSQL database into the Jupyter Notebook environment, and it underwent exploratory data analysis parallel to various attribute selection techniques, feature engineering techniques, and other data preprocessing methods to understand and cleanse the dataset prior to transferring it into the models. Then, it was split into two major subsets as X and Y, according to the independent variables and the dependent variable (price). Thereafter, it was again split into more sets such as x_train, x_test, y_train, y_test at a 75:25 split ratio for

training and testing. K-fold cross-validation was additionally utilized to assess model effectiveness with regard to overfitting and hyperparameter tuning. A comparative study was conducted targeting 15 different regressor models to perceive their performance with regard to price prediction while utilizing a complex dataset. Those models were trained and tested using the same train and test datasets continuously until satisfactory results appeared. The optimum model hyperparameters were identified and adjusted accordingly to the dataset characteristics using specific methods. The training processes and, likewise, the testing processes were applied repetitively for each of the models for better learning perception. The train and test dataset's accuracy levels and error rates were analyzed for individual models, and afterward, the top four victoriously performing regressor models were identified.

The top four models were amalgamated together to form an ensemble method, which resulted in a hybrid regression model. Using the test dataset, the accuracy of the hybrid model was analyzed, and the surfaced refinements that were necessary to be completed were done, and as a result, the process navigated back to the training of the four exceptional models once more. Simultaneously, an API was built using Flask python micro-framework in order to facilitate the communication between the model execution and result incorporation, along with a client-side application. The API endpoint implemented was a POST since in order for the prediction to happen, first of all, from the client-side, the forecasting car characteristics must be obtained and passed through to the API. Subsequently, succeeding the precision of the hybrid model, it was serialized using the pickle python library and again deserialized inside the API so that it can be integrated with the Flask ML API in order to provide the prediction results for the front-end application. The API containing the model was then deployed in the DigitalOcean cloud environment for admiring scalability and 24/7 availability. Consequently, a cloud-deployed Tievs front-end application was built using the Angular framework. Through the implemented classified advertisement submission form UI, the essential prediction features could be extracted and fastidiously analyzed to facilitate the API endpoint invocation and model price prediction delivery. After the ensembled stacked regressor has given the response through the API, it was able to visualize the final price range repercussion to the customer, within the advertisement form, near the price input section. New car classifieds being submitted into the application, having car attributes and other details, will be saved in the database and will be utilized for the purpose of retraining the prediction model.

2.5.2 Prediction model implementation

Fig. 2.18 depicts the first five rows and columns of the selected dataset, which is visualized through python pandas library commands.

```
In [6]: dataset.head()
Out[6]:
```

price	year	manufacturer	model	condition	cylinders	fuel	odometer	title_status	transmission	VIN	drive	size	type	paint_color
9500	2003.0	chrysler	town & country	excellent	6 cylinders	gas	30376.0	clean	automatic	NaN	fwd	mid-size	mini-van	blue
28990	2018.0	gmc	acadia sle-2 sport utility	good	6 cylinders	gas	20736.0	clean	other	1GKKNLLS2JZ178034	fwd	NaN	other	white
16590	2013.0	lincoln	mkz sedan 4d	good	NaN	gas	61087.0	clean	other	3LN6L2G99DR819673	fwd	NaN	sedan	red
28990	2017.0	gmc	acadia slt-1 sport utility	good	6 cylinders	gas	18041.0	clean	other	1GKKNMLS2HZ305977	fwd	NaN	other	NaN
20590	2017.0	lincoln	mkz premiere sedan 4d	good	NaN	gas	36436.0	clean	other	3LN6L5A97HR666209	fwd	NaN	sedan	NaN

Figure 2.18 Dataset first five rows

First of all, data preprocessing and exploratory data analysis steps were undertaken upon the dataset to gain a deeper understanding and transform the values as necessary for regression-based ML models. Some of the steps undertaken are shown below.

- First of all, filtering out the irrelevant columns that do not affect the price of a car was done using the relevant python commands.
  (Ex: URL, region, region_url, county, image_url, lat, long, VIN, description, state, and id)

- Through this feature engineering or feature selection, the retained impactful features that had at least a slight impact on the price were the year, manufacturer, model, condition, cylinders, fuel, odometer, title status, transmission, drive, size, type, paint color (overall 13 final estimators).
- Rather than eliminating a whole record just because it has some null values is unnecessary. Therefore, filling attribute null values with respective mean/mode values was done next which can also be called imputation. Here, year and odometer column mean values were calculated and then those values were replaced with the null values both year, and odometer columns had since they were numerical variables.

- As for the categorical variables, the mode values were accounted for and categorical column null values were replaced appropriately.

- The records where price = 0, price < $750, and price > $100000 were eliminated.

- Records relating to Ford, Chevrolet, and Toyota manufacturers were considered and F-150 model typed cars were selected to maintain scope and address limited computational resources.

- Next, the year column had to be managed accordingly. It is because the dataset seemed to have year value outliers as in cars made in 1900 and even in 2022, which is extremely unplausible. Hence, for this study, the cars made from 1975 to 2021 seemed ample enough for price prediction model creation and were refined.

- While doing so, another issue arose, which is that the condition labeled on some cars was not matching with their make year. Therefore, tinkering was done to the condition column values for appropriate categorization of the records.

- Finally, the last but not least process appeared at this point. That is the categorical variable conversion into numerical values by using the One-Hot-Encoding method. Categorical encoding is obeyed since categorical variables cannot be used when regression-based ML algorithms are being involved, such as this own research. The numerically standardized or scaled dataset's first five-row view is depicted by Fig. 2.19, which now has 651 columns. It was later exported as a .csv file for model implementation importing purposes.

price	year	odometer	manufacturer_chevrolet	manufacturer_ford	manufacturer_toyota	model_f-150	model_f-150 **1 owner clean	model_f-150 20" fuel wheels fuel	model_f-150 2dr single cab 4x4 v6 5 speed manual long bed	model_f-150 4wd	model_f-150 4wd supercab 145" xlt	model 150 4 superc 163"
20000	2011.0	101000.0	0	1	0	0	0	0	0	0	0	
15500	2004.0	60000.0	0	0	1	1	0	0	0	0	0	
6800	2008.0	202000.0	0	1	0	0	0	0	0	0	0	
3750	1978.0	123456.0	0	1	0	0	0	0	0	0	0	
3900	1997.0	230000.0	0	1	0	0	0	0	0	0	0	
22950	2014.0	166380.0	0	1	0	0	0	0	0	0	0	
10500	1988.0	10200.0	0	1	0	0	0	0	0	0	0	
12800	2000.0	97800.0	0	1	0	0	0	0	0	0	0	
8950	2011.0	164000.0	0	1	0	1	0	0	0	0	0	
31500	2019.0	106054.0	0	1	0	0	0	0	0	0	0	

Figure 2.19 One-Hot-Encoded categorical columns

Then, the next phase was the **15 regressors comparative analysis**. After a comprehensive initial investigation, the most eligible 15 regressor models were identified to continue with the rest of the methodology for creating the optimum price prediction model.

The regressors were namely, **Extra Trees, Bagging, RF, Gradient Boosting, Decision Trees, K-nearest Neighbors (KNN), Support Vector Machine (SVM),**

Lasso, Ridge, Elastic Net, AdaBoost, Multilayer Perceptron (MLP), Linear SVR, Stochastic Gradient Descent (SGD), and Multiple Linear Regressor. Each model was trained using the same training data and evaluated using the same testing data, and the train: test split ratio was 75:25 with 10 fold cross-validation to make the models less biased and account for every observation until satisfactory results emerge.

Moreover, overfitting, which is the level that the model fits exactly to a training dataset and result in poor predictions for unseen future observations while defeating the model's sole purpose, was eradicated to a commendable extent, enhancing generalization when K-fold cross-validation was utilized. In supervised ML, the regression-based method rather than the classification route has proven to be highly reliable and convenient in evaluating a continuous target variable.

When tuning the hyperparameters of the models and obtaining the optimum ones, the three major methods that were followed are **K-fold cross-validation**, **RandomizedSearchCV**, and **GridSearchCV**. By doing that, the model accuracy, speed, and prediction power increased, all the while reducing error rates. Feature scaling (normalization) from StandardScaler() was used to deal with KNN, SVR, linear SVR, and SGD regressors.

Custom arrays and functions were implemented to reuse the code when examining the models each by each and derive the R-squared value (R2), mean absolute error (MAE), relative squared error (RSE), and root mean square error (RMSE) was used as accuracy criterion to determine the top 4 exemplary models out of all those considered. Following pseudo code explains the overall methodology followed when training and testing the precision and error rates of the 15 different regression models individually.

```
PROCEDURE MODEL_TRAIN_TEST(regressor)
    BEGIN
        Import necessary dependencies
        Import Scikit Learn regressor models
        Import cleaned dataset
        Separate independent attributes and dependent attribute
        Dataset split into train_test at 75:25 (x_train, x_test, y_train, y_test)
        FOR each regressor model
        DO
            Model initialization with optimal hyperparameter values
            x_train and y_train fit into created model
            Call function acc_model(model_number, model, x_train, x_test)
        UNTIL all models executed
        END FOR
        Visualize final table having accuracy and error metrics of all models
    END
```

The results acquired and visualized in Jupyter Notebook regarding the models will be revealed in the 3rd chapter, 'Results & Discussion.' The ulterior motive of this phase was to ascertain the top 4 exemplary models for a further innovative approach, which is to apply a heterogenous ensemble learning technique to produce a combined price prediction. Since the individual analysis was done foremostly, a better perception of how the ensemble model's accuracy will differ in comparison to the singular models themselves was proclaimed.

Then, after successful selection of the top 4 most exemplary models out of all 15, the next step was to conduct the **ensemble model development**. This analysis conducted its ensemble experiment using the stacked regressor/stacking generalization method. The reason for this choice was being able to experience the functionality of both bagging and boosting ensembles through the pronounced regressors since the majority consisted of such ensemble-based algorithm structures. To provide an accurate prediction by the assembled model than any of its individual models, the base learners are required to be as distinctive in nature and precise as possible, and the sophisticated

analysis on the 15 regressors provided expediency for that. The stacking method includes base learners and a meta-leaner(meta-regressor), and the feature outputs of the trained (with the thorough training dataset) base learners are obtained by the meta-model for its own training and final forecast render. Following Fig. 2.20 depicts the algorithm used for the stacked regressor ensemble learning process.

Algorithm Stacking

```
Input: training data D = {x_i, y_i}_{i=1}^m
Ouput: ensemble regressor H
Step 1: learn base-level regressor
for t = 1 to T do
    learn h_t based on D
end for
Step 2: construct new data set of predictions
for i = 1 to m do
    D_h = {x'_i, y_i}, where x'_i = {h_1(x_i), ..., h_T(x_i)}
end for
Step 3: learn a meta- regressor
learn H based on D_h
return H
```

Figure 2.20 Stacking regressor algorithm [24]

After the analysis and further refinements, the ensemble model was to finally be serialized (exported from the python file in order to be used by an external API or any other server for storage purposes) using the python library known as 'Pickle'.

The next step of the price prediction system implementation was to develop and integrate the ML API built from the Python microframework Flask. The reason for selecting Flask for the experimentation of building an ML model facilitating API is due to its lightweight structure as minimalistic features without having to be dependent on external libraries or other frameworks. Below illustrated Fig. 2.21 shows all the python related dependencies imported within the priceApi.py file implemented API. The .pkl files containing the stacked regressor model and the column headings were deserialized using pickle.load() function. Below exhibited Fig. 2.22 displays the rest

of the code in the API, which indicates the function targeted by a POST type request towards the hosted server's "/predict-price" route, to acquire the car advertisement attributes as a JSON string, convert it to a data frame and do the necessary encoding by the pd.get_dummies() method prior to being attached with the OHE column headings and final prediction using the stacked regressor model which was deserialized.

```
# Dependencies
from sklearn.base import BaseEstimator, TransformerMixin, RegressorMixin, clone
from flask import Flask, request, jsonify, render_template
import pickle
import sys
import traceback
import pandas as pd
import numpy as np
from sklearn.preprocessing import LabelEncoder
from flask_cors import CORS
app = Flask(__name__)
cors = CORS(app)

#   Load related columns
model_columns = pickle.load(open('model_columns.pkl', 'rb'))

# Load models
stacking_regressor = pickle.load(open('stackingModel.pkl', 'rb'))


@app.route('/')
def home():
    return render_template('predict.html')
```

Figure 2.21 Flask API part 1

```
/ > Users > savandisubhartha > New-2021-195 > ML > PriceP > priceApi.py
priceApi.py
@app.route('/predict-price', methods=['POST'])
def predictPrice():
    if bagging and etr and randomForest and gradientB:
        json_ = request.json
        df = pd.DataFrame(json_, index=[0])
        print(df)
        query = pd.get_dummies(df)
        query = query.reindex(columns=model_columns, fill_value=0)
        print(query)

        model = (stacking_regressor)

        stacked_regression_predictions = np.column_stack([
            model.predict(query) for model in model
        ])

        final_prediction = np.mean(stacked_regression_predictions, axis=1).round(2)
        price_range1 = (final_prediction - 3000).round(2)
        price_range2 = (final_prediction + 3000).round(2)

        # perform conversion
        final_prediction = str(final_prediction)[1:-1]
        price_range1 = str(price_range1)[1:-1]
        price_range2 = str(price_range2)[1:-1]

        return jsonify({'price': str(final_prediction)})

    else:
        print('Train the price prediction model first')
        return 'No price prediction model deployed here to use'


if __name__ == '__main__':
    app.run(host='localhost', port=2096, certfile='cert.pem', keyfile='key.pem', debug=True)
```

Figure 2.22 Flask API part 2

Succeeding the triumphant API implementation with the integration of the developed ML ensemble model, it was necessary for it to be deployed in a cloud server environment for the client application's efficient accessing. For that, the DigitalOcean Infrastructure as a Service (IaaS) cloud service was utilized. The front-end application, dubbed Tievs, was built primarily on the Angular framework, with Typescript serving as the programming language for UI components.

2.5.3 Prediction model testing

Tests were executed in parallel to development work and not only near system completion. Unit testing, smoke testing, regression testing, component testing, integration testing, API testing, UI testing, black-box testing, and production testing were some of the functional testing types conducted during the implementation of the price prediction system and Tievs application as a whole. API testing of the hybrid model integrated Flask API was done almost nearing the end of the cycle to clearly and accurately evaluate the authenticity of predicting a price value.

Below Fig. 2.23 evince the Postman UI of the test case for assessing the POST request sent towards the following deployed API endpoint. The request parameters are sent as raw JSON formatted strings having the necessary key and value pairs relating to a car's features. A total of 13 attributes are being stated and sent. A response is then received indicating the predicted price from the ML ensemble model, that too as a JSON string. The response status is shown as 'OK' with the code of 200, and its' size is stated as 659B, which is considerably low and healthy for the request calls. The overall time from the request to the response by accessing the cloud environment hosted is 3.38 seconds without accounting for caching. With caching, the response time was reduced to 410 milliseconds.

API POST Endpoint – https://ml.tievs.com:2096/predict-price

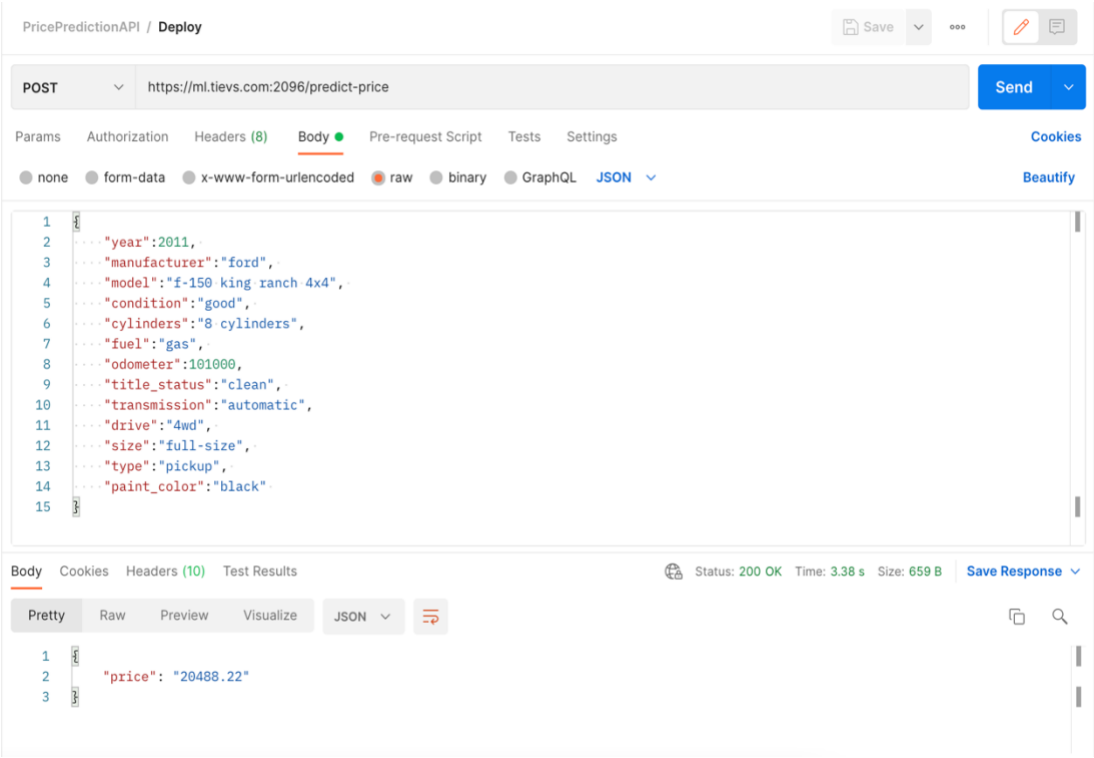


Figure 2.23 Postman API testing

Not only the functional aspect of this research component but also the non-functional requirements were also contemplated by this experiment. Following Fig. 2.24 and Fig. 2.25 depicts the Loadmill load test results presented in an attractive graphical view. The total elapsed time for the test was 2 minutes and 3 seconds. The active simulated user sessions were 5 between each second in the load test throughout. It gained an error rate of just 1.06%, which is an extremely outstanding performance. Normally, it is stated that these types of ML model integrated APIs perform quite poorly when it comes to the error rate statistic, although this price prediction API under test begged to differ in view. Hence it is definitely a remarkable result. The average response time calculated was 501 milliseconds which can be considered as another astounding outcome received with regard to an efficient response time, which many researchers try to attain by their built ML model incorporated APIs. Moreover, the performance over time and the throughput in requests per second (rps) are graphed for convenient visualization.

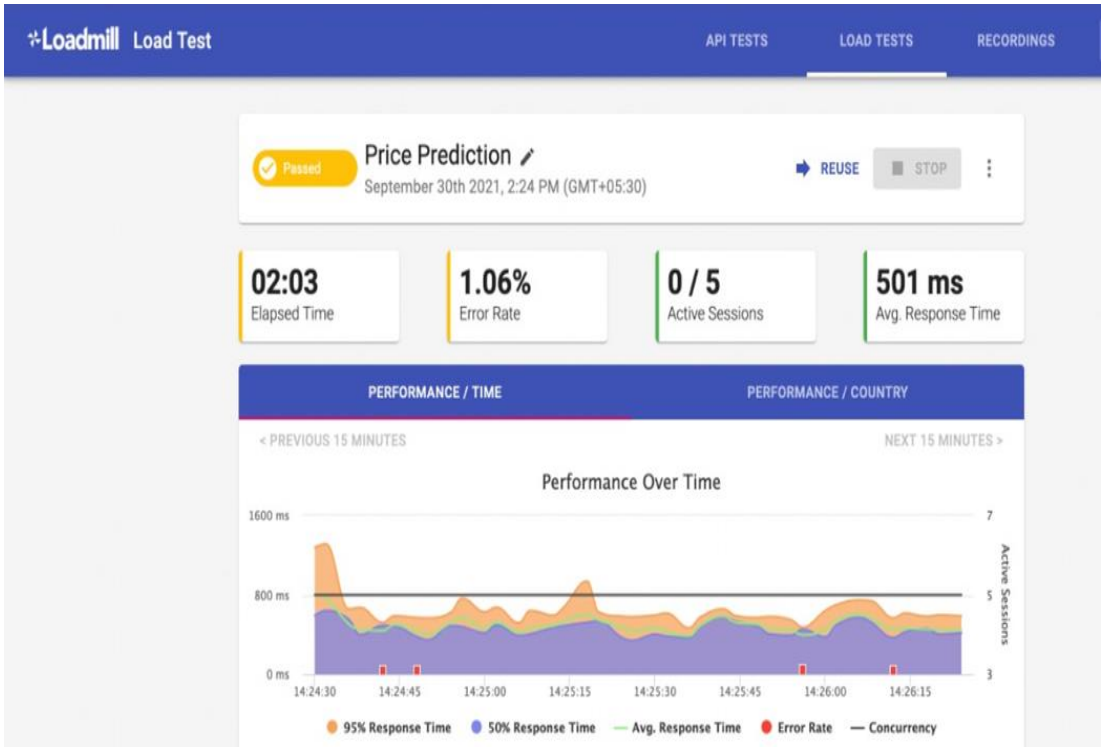


Figure 2.24 Price prediction Loadmill test part 1

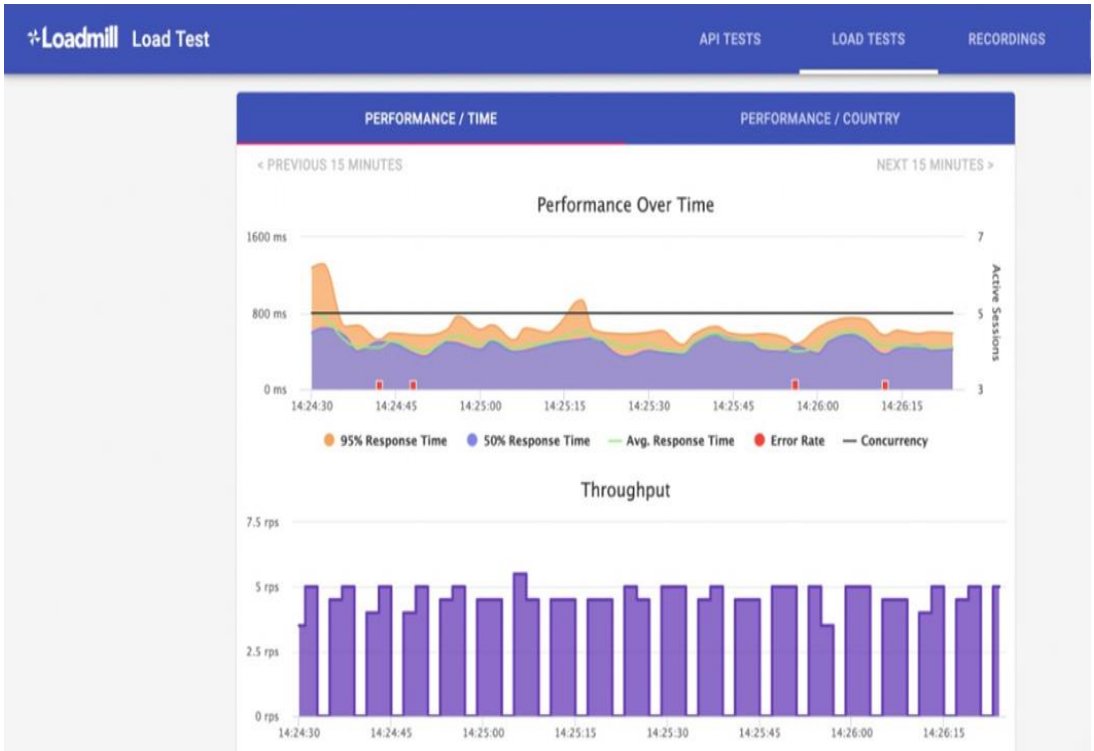


Figure 2.25 Price prediction Loadmill test part 2

2.6 Fraud Content Detection System

2.6.1 Fraud context detection high-level system overview

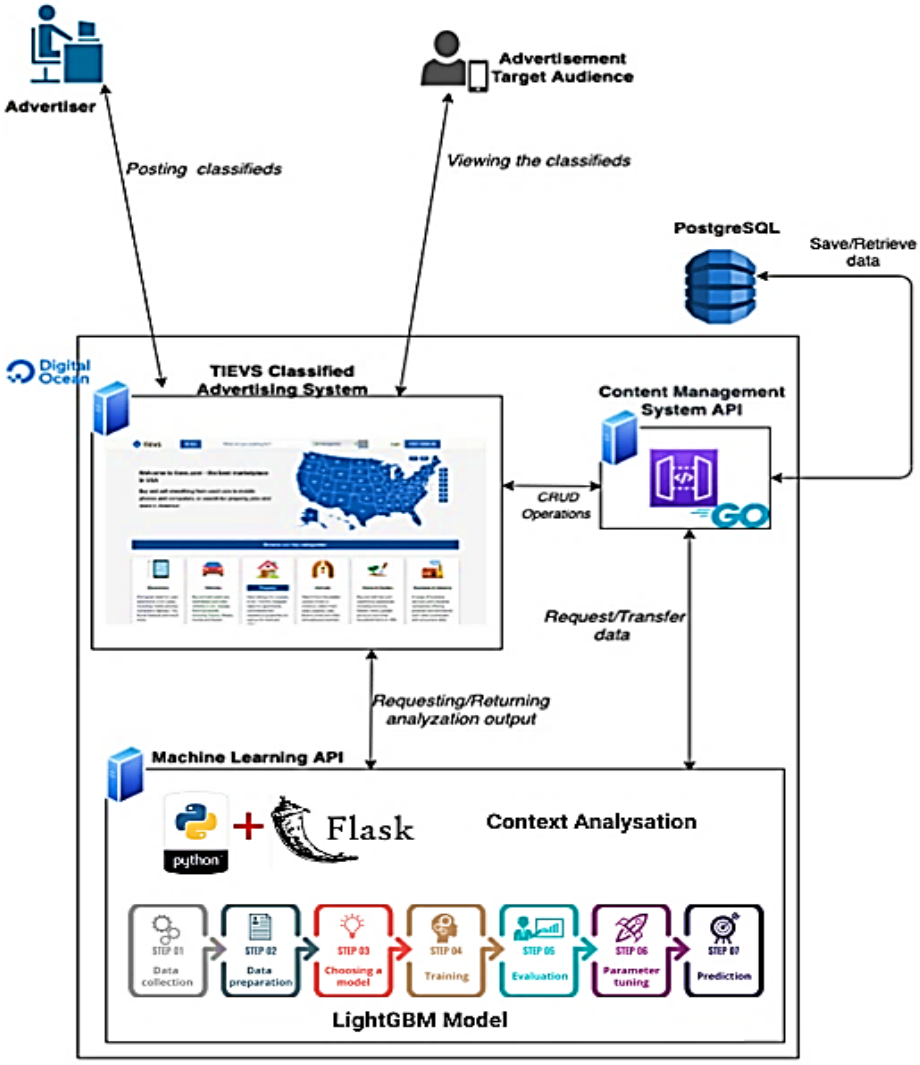


Figure 2.26 Overall high-level system diagram of fraud context detection

The entire high-level system architecture of the newest component architecture built for car classifieds fraud prevention and detection is illustrated in Fig. 2.26. The Flask machine learning API would first extract the dataset from the database, where it may be aggregated and saved using a content management system. Following the acquisition of the dataset, data exploration and different data preparation procedures will be used to purify the dataset before it is transferred into the algorithms. After the

dataset has been properly prepared, it will be divided into subsets and put into the machine learning algorithm model that is thought to be the best for this study concept. The description will be taken from the advertising form in the application interface into the Flask API, which has the model, and the detection result will be generated after rigorous inspection of the characteristics. then, Following the model's accuracy, it will be integrated also with the Angular application to identify fraudulent contents of car advertising that engage with the Frontend. The identification will be exhibited to the consumer as a fraud notice within the advertising form, near the descriptions input section. If a legitimate user posts, it will be stored in the database whereas the user permits the submission to proceed.

2.6.2 Fraud context detection implementation

These contents are persistently dynamic and might vary from one advertisement to the next Consequently, the primary goal of this aspect is to undertake supervised learning and determine the likelihood of spam based on the inputs. Below offers a brief summarization of the above high-level procedures.

- Aiming to foresee the likelihood of fraudulent content before ad submission, an initial dataset record was labeled as legitimate and illegitimate, where the latter was prepared.

- Afterward successfully installing all of the packages (including libraries and dependencies) required for the procedure afterward to obtain a communicative impression in the forms of charts and graphs the exploratory data analysis (EDA) technique was undertaken primarily. Despite knowing, the highest prevalent terms used throughout spams and hams could assist us in properly comprehend the dataset. A word cloud assistance exemplifies exactly what sort of terms are prevalent within every classification.
- Initially, To provoke a word cloud, partition the categories into two panda’s

data frames and implement a word cloud procedure, as indicated below in Fig. 2.27

```
data_ham  = dataset[dataset['classLabel'] == "ham"].copy()
data_spam = dataset[dataset['classLabel'] == "spam"].copy()

def show_wordcloud(dataset, title):
    text = ' '.join(dataset['description'].astype(str).tolist())
    stopwords = set(wordcloud.STOPWORDS)
    fig_wordcloud = wordcloud.WordCloud(stopwords=stopwords, background_color="#ffa78c",
                                        width = 3000, height = 2000).generate(text)
    plt.figure(figsize=(15,15), frameon=True)
    plt.imshow(fig_wordcloud)
    plt.axis('off')
    plt.title(title, fontsize=20)
    plt.show()
```

Figure 2. 27 Implementation of “ham” “spam” word cloud visualization

- Succeeding EDA preprocess technique, which is consists of Label encoding, Handling null values, Regular expressions Handling categorical variables, Removing stop words and stemming, NLP technique TF-IDF, Bag of words often perceived in writing regardless of context was utilized to eliminate all interference from inputs that would otherwise obstruct the potential to distinguish positive and negative classifiers.

- As the stage of preprocessing the stop words from the text content corpus should be taken out. In natural language processing, futile words (information), are alluded to as stop words. Stop words are terms that have been configured to be disregard by search engines mostly when pre-defined for searching and extracting them because of a user’s user's query such as “the”, “a”, “an”, “in”, "but", "because" etc. and so on As displayed underneath, able to be handled substance more effective by eliminating words that not even operate to any future tasks. Below Fig 2.28 depicts the pseudocode for the ‘stop words’ implementation procedure.

```
Define : List of stopword removal
For i=1 to Number_of_words_in_the_document do
        For j=1 to Number_of_words_in_stopwords_list do
                If
                Words(i) == stopwords(j) THEN
                Eliminate Words (i)
                End If
        End For
End For
```

Figure 2.28 Pseudocode stop words removal

- The procedure of the Stemmer algorithm, which eliminates affixes to retrieve the basic form of words, was utilized as a stemming technique due to its computational speed. Extraction of the text base form of a particular word implementation can be illustrated as Fig 2.29

```
#extract the base form of words by removing affixes
ss = nltk.SnowballStemmer("english")
dataset['description'] = dataset['description'].apply(lambda x: ' '.join(ss.stem(term) for term in x.split()))
dataset.head()
```

Figure 2.29 Extraction of the base form of words using snowball stemming

- The raw text was essentially converted into several vectors to perform in ML models due to its impotence to directly operate textual forms.
- Then, a Bag of Words (BOW) model was developed as depicted in Fig 2.30, to extract characteristics from the text before the implementation of the TF-IDF NLP vectorization technique to evaluate the significance of text data and serialize them for further transformation Subsequently, the Scikit-learn library was utilized to convert text content into a matrix of numbers that could be fed to the classifier.

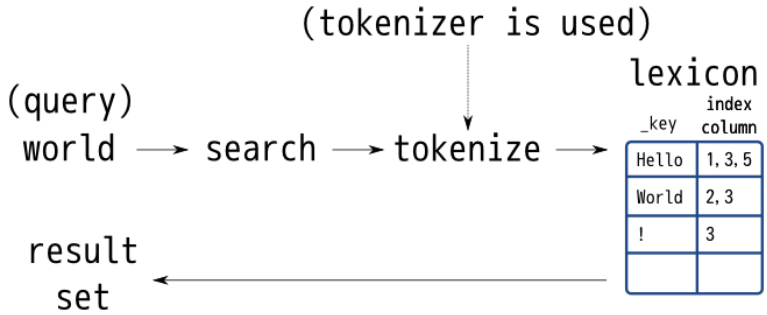


Figure 2.30 Text extraction process using BOW

- This study was initiated to evaluate the expertise of 7 ML algorithms such as Decision Tree, RF, SVM, KNN, XGBoost, XGBRF, and Light Gradient Boosting Machine (LightGBM) classifier in a comparative analysis after vectorization. This is the development's machine learning component. Our current issue is determining if a user is going to submit a post as spam or not. This is a conventional machine learning classification issue in which one class is ham (positive class)) and the other is spam (positive class). Our objective is to be able to create a system that can identify whether a user will submit a posting as spam.

- One strategy to solving this challenge is to utilize supervised learning and is what we have been using in our research In this technique, there is indeed a learning phase in which our algorithm "gets to know" about the features of spamming and innocuous postings after being fed certain tagged posts The highest max_depth obtaining implementation can be shown as below in Fig 2.31. The max_depth parameter provides a huge impact on LightGBM when considering the performance of the model and is time-consuming.

```python
#The performance metric is F1 score
def train_and_test(model, model_name):
    model.fit(X_train, Y_train)
    pred = model.predict(X_test)
    print(f'F1 score is: {f1_score(pred, Y_test)}')

for depth in [1,2,3,4,5,6,7,8,9,10]:
    lgbmodel = lgb.LGBMClassifier(max_depth=depth, n_estimators=1000, num_leaves=40)
    print(f"Max Depth {depth}")
    print(" ")
    print(" ")
    train_and_test(lgbmodel, "Light GBM")
```

Figure 2.31 Highest Max_depth obtained from F1 score metric implementation

Each classifier was trained and tested until the optimal model would be exposed in terms of F1 score (accuracy and recall) and confusion matrix. Based on iteration, proceeding with LightGBM as optimum model disclosure, the highest max depth value was identified as shown above Fig 2.31, from parameter tuning compared with competency to accomplish the most pertinent parameters for the best classifier. After the max_depth iteration, hyperparameter tweaking using RandomizedSearchCV which is most appropriate for our study when compared to GridSearchCV has been implemented for further analysis and obtained the most ideal parameters for the LightGBM model. Obtained parameter values depict the below Fig 2.32.

```python
# let's use the best parameters to the train the model:

best_model = lgb.LGBMClassifier(subsample = 0.5,
 reg_lambda = 0.5722222222222222,
 reg_alpha = 0.2888888888888886,
 num_leaves = 24,
 min_split_gain = 0.0,
 min_data_in_leaf = 60,
 min_child_weight = 0.01,
 min_child_samples = 30,
 max_depth = 5,
 learning_rate = 0.1,
 bagging_freq = 4,
 bagging_fraction = 0.7944444444444444)
best_model.fit(X_train,Y_train)
```

Figure 2.32 Best parameters for LightGBM model

After successful embedding of the trained model in the REST API fraud-detection endpoint, it could classify ad content as legitimate or not and display caution Tievs near advertisement submission when indispensable.

Below Fig 2.33, Fig 2.34 & Fig 2.35 depict the fraudDetection() method implementation in Flask API implementation, textVectosizer() method implementation in Flask API, and lastly UI implementation for obtaining inputs from "Tievs" front-end application, respectively.

```
@app.route('/fraud-detection', methods=['POST'])
def fraudDetection():
    response = flask.jsonify({'some': 'data'})
    response.headers.add('Access-Control-Allow-Origin', '*')

    if request.method == 'POST':
        description = flask.request.json['description']

        prediction = textVectorizer(description)
        return prediction

    else:
        print('Train the model first or input data')
        return 'Nothing to detect'

if __name__ == '__main__':
    app.run()
```

Figure 2.33 Flask API fraudDetection() implementation

```
def textVectorizer(description):
    global my_prediction
    # print("description", description)
    description = [description]
    dataset = {'description': description}
    # print("dataset", dataset)
    data = pd.DataFrame(dataset)
    data["description"] = data["description"].str.replace(
        r'^.+@[^\.].*\.[a-z]{2,}$', 'emailaddress')
    data["description"] = data["description"].str.replace(
        r'^http\://[a-zA-Z0-9\-\.]+\.[a-zA-Z]{2,3}(/\S*)?$', 'webaddress')
    data["description"] = data["description"].str.replace(r'£|\$', 'money-symbol')
    data["description"] = data["description"].str.replace(
        r'^\(?[\d]{3}\)?[\s-]?[\d]{3}[\s-]?[\d]{4}$', 'phone-number')
    data["description"] = data["description"].str.replace(r'\d+(\.\d+)?', 'number')
    data["description"] = data["description"].str.replace(r'[^\w\d\s]', ' ')
    data["description"] = data["description"].str.replace(r'\s+', ' ')
    data["description"] = data["description"].str.replace(r'^\s+|\s*?$', ' ')
    data["description"] = data["description"].str.lower()

    stop_words = set(stopwords.words('english'))

    data["description"] = data["description"].apply(lambda x: ' '.join(
        term for term in x.split() if term not in stop_words))
    ss = nltk.SnowballStemmer("english")
    data["description"] = data["description"].apply(lambda x: ' '.join(ss.stem(term)
                                                                        for term in x.split()
    global my_prediction
    model = pickle.load(open('FraudDetectionModel/fraudDetection.pkl', "rb"))
    tfidf_model = pickle.load(open('FraudDetectionModel/tfidf_model.pkl', "rb"))

    tfidf_vec = tfidf_model.transform(data["description"])
    tfidf_data = pd.DataFrame(tfidf_vec.toarray())
    result = model.predict(tfidf_data)
    my_prediction = jsonify({'prediction': str(result[0])})

    return my_prediction
```

Figure 2.34 Flask API textVectorixer() implementation

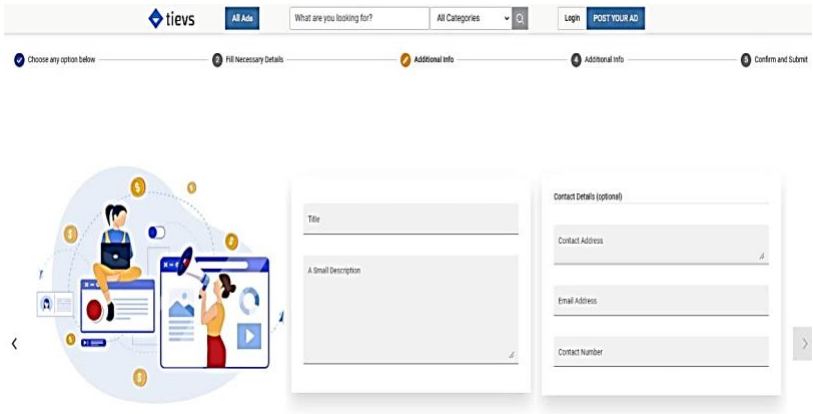


Figure 2.35 User interface implementation for obtaining user inputs

2.6.3 Fraud context detection testing

Testing was carried out to confirm that the model's functions worked as expected, as well as to determine the algorithm's correctness and dependability, as well as its computing efficiency. In development mode, model accuracy, recall, and perhaps the ROC curve and Confusion Matrix were examined. The verified postman findings were displayed, and the model's performance in production mode was successfully analyzed using two data sets with different records.

Using the previously described classification matrix and test case, the primary goal was to identify the most effective model and make the most of the disclosed model. The below Fig 2.36 depicts the postman results in “fraud-detection” endpoint real-world input from the user in a fraudulent situation after integration and could deployment for establishing the production purpose.

API POST Endpoint – https://ml.tievs.com:2087/fraud-detection

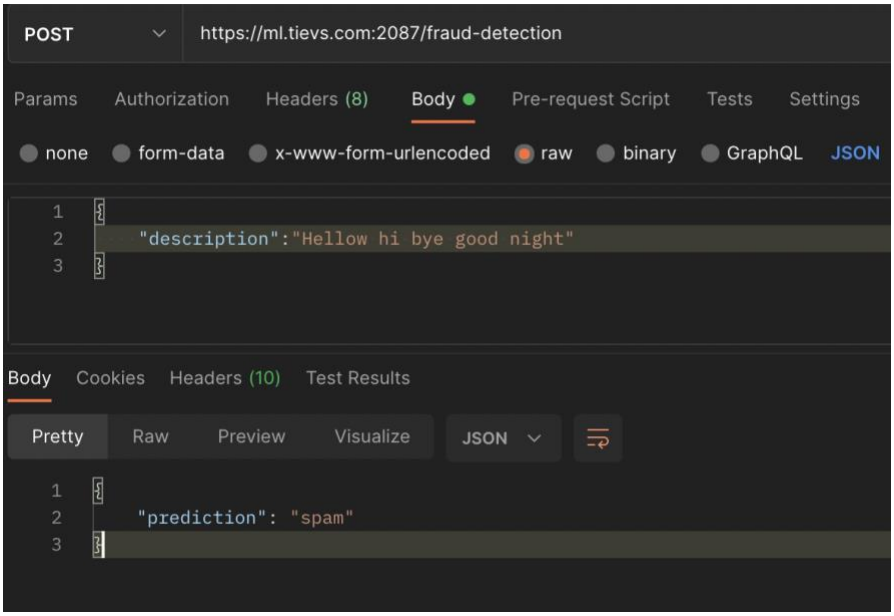


Figure 2.36 Postman of fraudulent input - production mode testing result

Successfully predicted non-fraud (ham) visualization in postman for the local developer testing mode has shown in the below Fig 2.37.

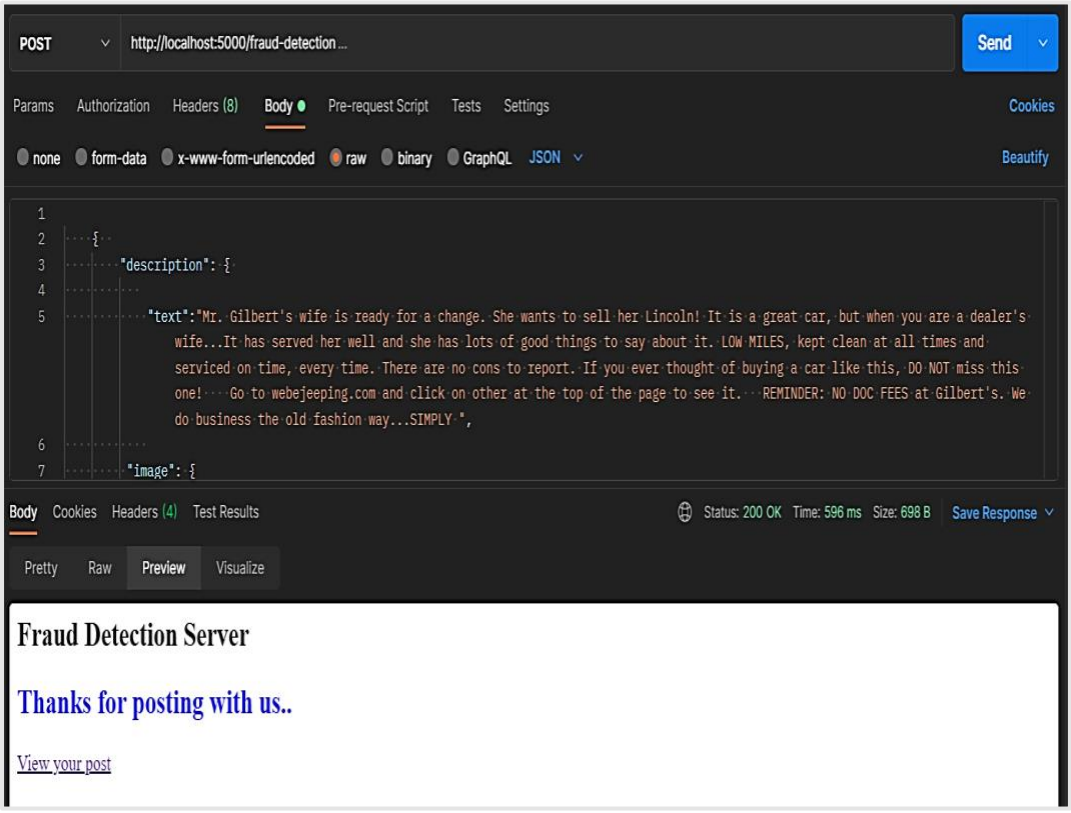


Figure 2.37 Postman of legitimate input - development mode testing result

2.7 Fraud Image Detection System

2.7.1 Fraud image detection high-level system overview

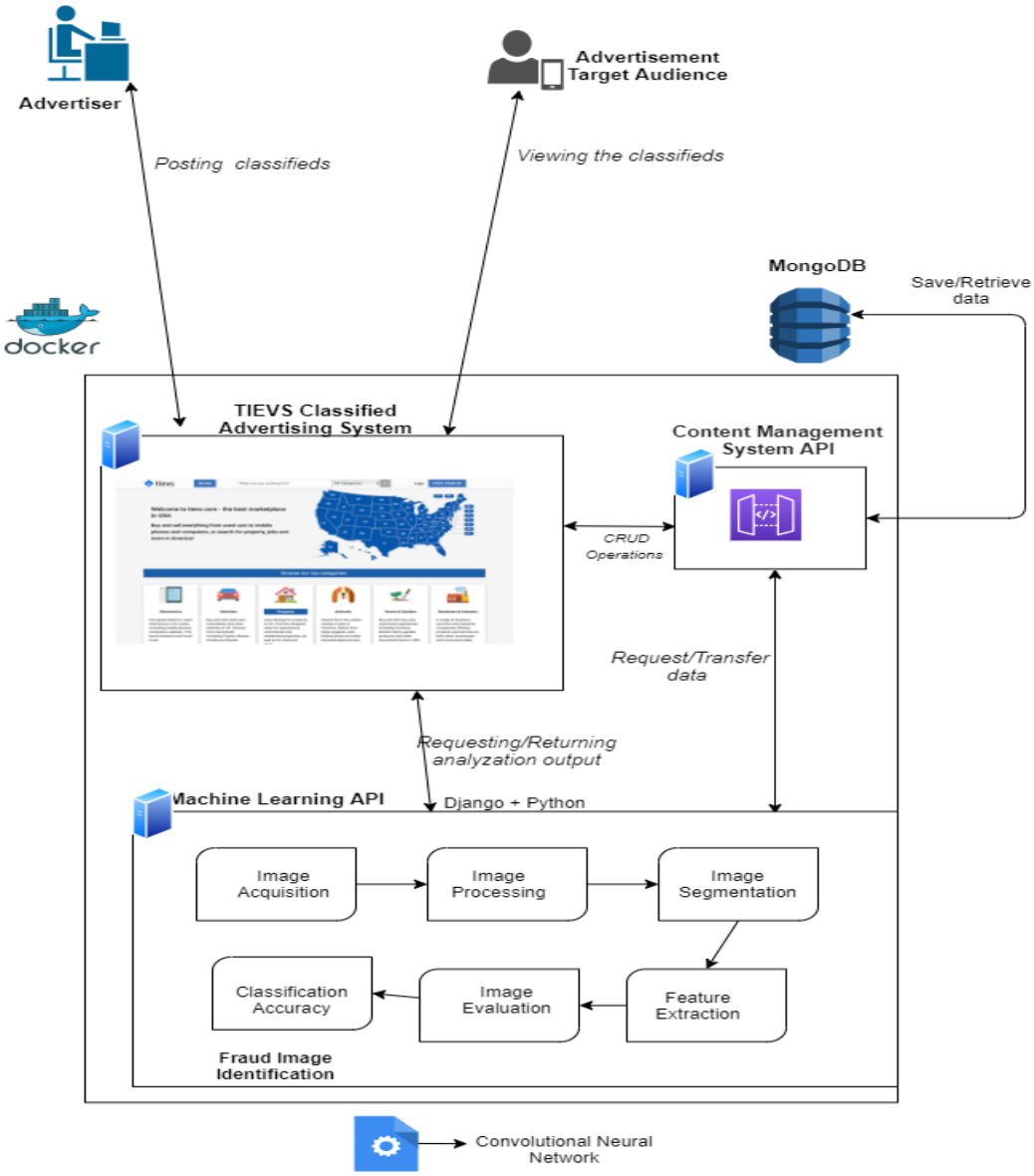


Figure 2.38 Overall high-level system diagram of fraud image detection

The above shown Fig. 2.38 depicts the Overall high-level system architecture of the newest component architecture built for Fraud image detection and prevention in 'Tievs' car classified advertising. The acquired dataset was first imported from the

Django database into the Jupyter Notebook environment, where it was subjected to exploratory data analysis alongside various attribute selection techniques, feature engineering techniques, and other data preprocessing methods in order to understand and cleanse the dataset before transferring it into the models. The data was then divided into two primary subsets, X and Y, based on the independent factors and the dependent variable (vehicle). Following that, it was divided into several sets, such as x train, x test, y train, y test, using a split ratio for training and testing. A comparison study was carried out on Keras models in order to assess their effectiveness in binary prediction while using a complicated dataset. Those models were continually trained and tested using the same train and test datasets until satisfactory results were obtained.

The images will be extracted from the advertising form in the application interface and entered into the Django API, which contains the model, and the detection result will be generated after a thorough analysis of the features. Following the correctness of the model, it will be integrated with the Angular application to identify Fraud images of vehicle advertising that interact with the Frontend. The Advertiser will display the identification as unrelated images within the advertising form, near the images input area.

2.7.2 Fraud image detection implementation

Authors stipulated the DL algorithm, **Convolution Neural Network (CNN),** and in its convolutional layer, import the10 layers, initial image feature learning was done using the vehicle/non-vehicle image set to extract characteristics such as gradient orientation, colors, edges, etc., assign weights and compress them dimensionally as convolved features compared to the original image. GREY scaling the data to execute image preparation procedures such as image rotation, resizing, and part extractions were done using Keras API. Next, the pooling layer reduced spatial size further while extracting superior traits and optimizing processing power. These 2 layers executed continuously and fastidiously trained the CNN model in discerning image qualities.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lambda (Lambda)              (None, 64, 64, 3)         0
_________________________________________________________________
conv1 (Conv2D)               (None, 22, 22, 128)       3584
_________________________________________________________________
dropout (Dropout)            (None, 22, 22, 128)       0
_________________________________________________________________
conv2 (Conv2D)               (None, 8, 8, 128)         147584
_________________________________________________________________
dropout_1 (Dropout)          (None, 8, 8, 128)         0
_________________________________________________________________
conv3 (Conv2D)               (None, 3, 3, 128)         147584
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 1, 1, 128)         0
_________________________________________________________________
dropout_2 (Dropout)          (None, 1, 1, 128)         0
_________________________________________________________________
dense1 (Conv2D)              (None, 1, 1, 128)         1048704
_________________________________________________________________
dropout_3 (Dropout)          (None, 1, 1, 128)         0
_________________________________________________________________
dense2 (Conv2D)              (None, 1, 1, 1)           129
=================================================================
Total params: 1,347,585
Trainable params: 1,347,585
Non-trainable params: 0
```

Figure 2.39 Train the best parameters to the CNN model

The above Fig. 2.39 demonstrates the best parameters selected for the CNN model. The final output was flattened and column vectorized before classification through a fully connected feed-forward neural network layer. With backpropagation applied iterative training, the model could discriminate high/low-level image features preparatory to effectuating **Softmax classification**. After the model was thoroughly tested on the accuracy, it was capable of identifying ad permissible car images by the manufacturer as well as prohibited images. The specific ML API was created using the Django framework and integrated with the trained model before being implied into the front-end application to detect inappropriate images and display warning signs necessarily. CNN differentiates from the traditional model by using only a few parameters to reduce overfitting probability and considering neighbor context (images in this scenario) information to note similarities. Below Fig. 2.40 shows the importing of the vehicle/non-vehicle dataset.

```
In [7]: from sklearn.utils import shuffle

        # shuffle dataset
        dataset = shuffle(dataset)
        dataset.head()
```

Out[7]:

	path	is_vehicle	image
2028	dataset/vehicles\2827.png	car	[[[19, 18, 13], [18, 17, 14], [16, 20, 16], [1...
10510	dataset/non-vehicles\extra2856.png	non_car	[[[69, 72, 85], [70, 73, 84], [65, 69, 79], [6...
12933	dataset/non-vehicles\extra5311.png	non_car	[[[176, 179, 184], [174, 177, 182], [172, 175,...
8007	dataset/vehicles\middle (368).png	car	[[[86, 94, 70], [83, 91, 67], [83, 92, 68], [8...
506	dataset/vehicles\1455.png	car	[[[19, 17, 13], [15, 15, 14], [14, 14, 12], [1...

Figure 2.40 Shuffle the dataset

In order to aim to predict the potential of fraudulent image submission prior to ad submission, an initial dataset record was categorized as related and non-related, with the latter being prepared. After successfully installing all of the packages (including libraries and dependencies) necessary for the operation, the exploratory data analysis (EDA) approach was largely used to get a communicative impression in the form of charts and graphs. Below shown Fig. 2.41 displays how the dataset is being divided into vehicle and non-vehicle sets.

```
In [4]: import pandas as pd

        # add card to df
        car_df = pd.DataFrame(cars, columns=['path'])
        # add label 'car'
        car_df['is_vehicle'] = 'car'
        non_car_df = pd.DataFrame(non_cars, columns=['path'])
        non_car_df['is_vehicle'] = 'non_car'

        dataset = pd.concat([car_df, non_car_df], ignore_index=True, sort=False)

        dataset.shape

Out[4]: (17760, 2)
```

Figure 2.41 Implementation of vehicle and non-vehicle

2.7.3 Fraud image detection testing

During the implementation phase, all parts, from data engineering to front-end application deployment, underwent continuously and repeated testing. Tests were carried out concurrently with development activity and not simply at the end of the system's life cycle. Some of the functional testing types used throughout the construction of the fraud image detection system and Tievs application as a whole included unit testing, smoke testing, component testing, integration testing, API testing, UI testing, black-box testing, and production testing. API testing of the CNN model integrated Django API was performed after the cycle to clearly and correctly evaluate the accuracy of image prediction. The purpose of this black-box testing process is to confirm the research components' compliance to their initial functional specifications and requirements.

Afterward, the confirmation of the model's function worked as expected, as well as determine the correctness of the algorithms and efficiency of the model and accuracy of the model Softmax classification and GRAY scaling were examined. The postman verified results are Showed. The Fig 2.42 below shows the postman results in "non-fraudulent image detection" endpoint real-world input from the user after integrating and deployment for defining the purpose on the customer.

API POST endpoint - https://catsy.org/check-vehicle/

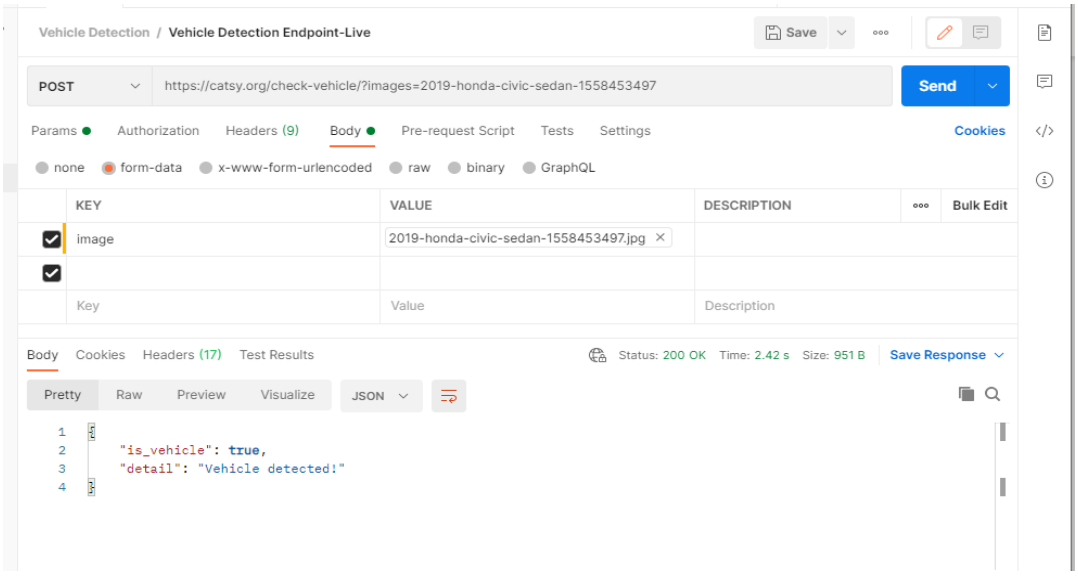


Figure 2.42 Postman of non-fraudulent image detection

The below Fig 2.43 shows that successfully detected the fraudulent images in the system.

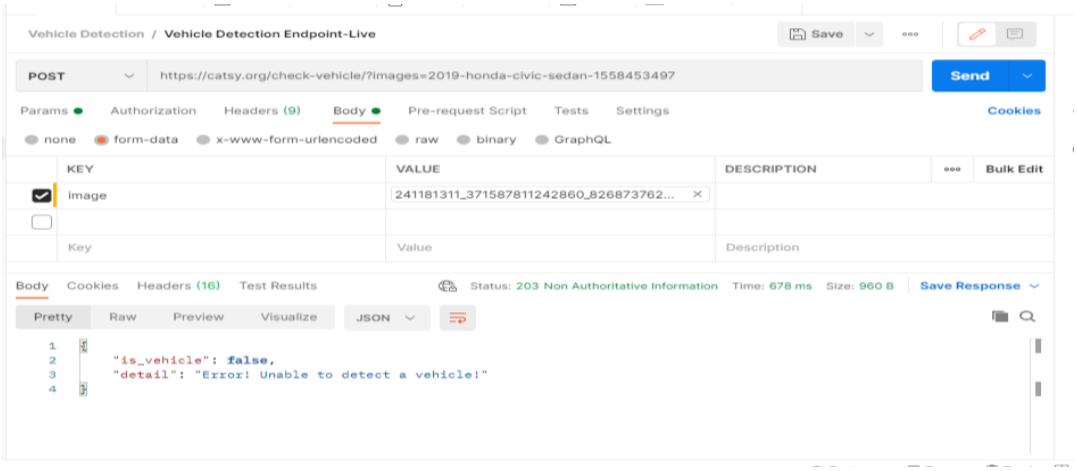


Figure 2.43 Postman of fraudulent image detection

3 RESULTS & DISCUSSION

3.1 Results

- **Recommendation system**

The introduced system has been able straightforwardly to attain the set objectives, making it a far superior recommender system than existing competitor systems. Following revealed Fig. 3.1 exhibit few matrices have astounding values that were used to attest the Tievs recommendation system's potentiality in achieving goals. The dashboard is dispensed for the API's registered domain under the load testing software known as 'Loadmill.

Tested API - https://ml.tievs.com:2053/recommend/15040?category=vehicles

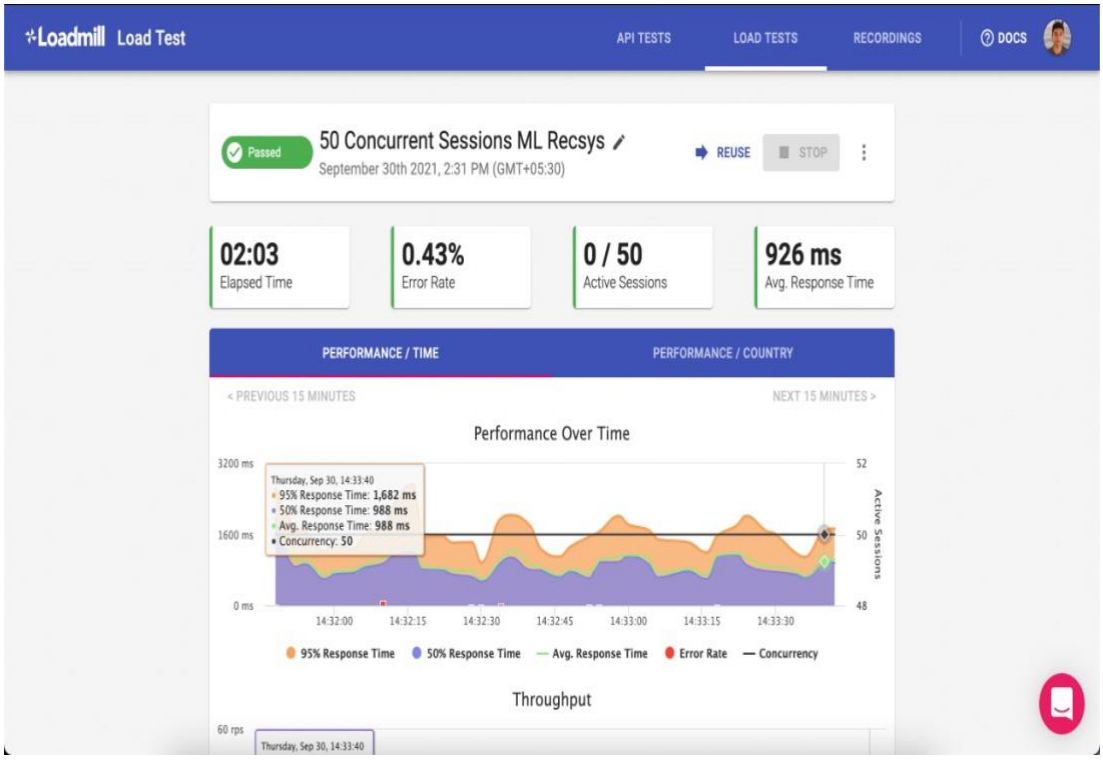


Figure 3.1 Loadmill load test

The performance over time graph is displayed from the below Fig. 3.2 more clearly about the average response time 988ms, 50% average response time 988ms, and 90%

response time 1682ms. The throughput graph illustrates the gradually fluctuating request per second (rps) graph, and the currently pointed throughput value remains at 36.5 rps, which can be reckoned as a similarly outstanding result procured.

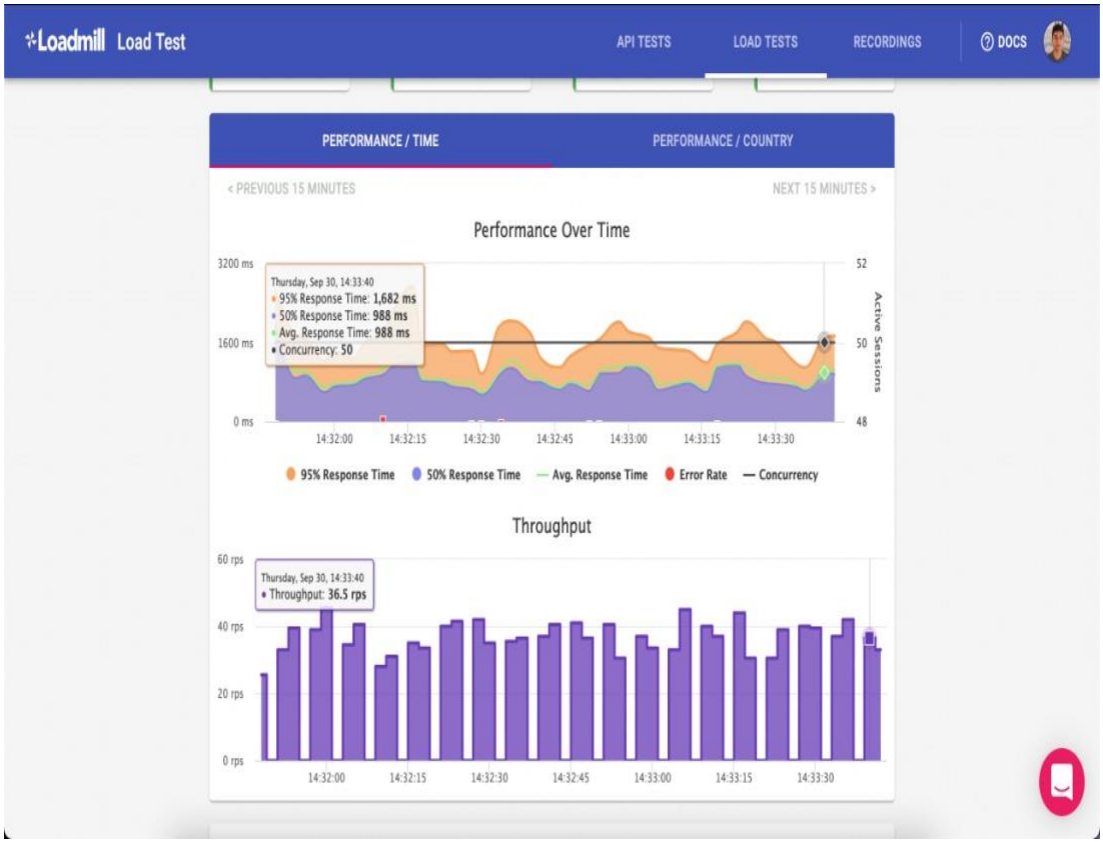


Figure 3.2 Performance graphs

Fig. 3.1 shows that there are 50 sessions concurrently running to make an impact on the server load. Concurrency implemented through Golang is a major aspect that has been researched and analyzed thoroughly through this proposed recommendation system. The Loadmill platform initially had allowed for only five different user sessions to be run within the load testing conducted for the allocated elapsing time. However, such a low amount of sessions was not adequate to test the overall power of the recommendation engine that could handle the concurrency commendably. Moreover, that is a critical quality that should be maintained by this research component being built, and that was another major characteristic that fills out the gap within the existing research done in the same domain and also what keeps other competitive systems at bay or at a lower level than this system.

To rectify the matter, the Golang API was redefined and reconstructed at a code level to allow for 50 sessions to be established during the load test conducted by Loadmill. By that, the necessary requirements for the concurrency handling capability were regulated.

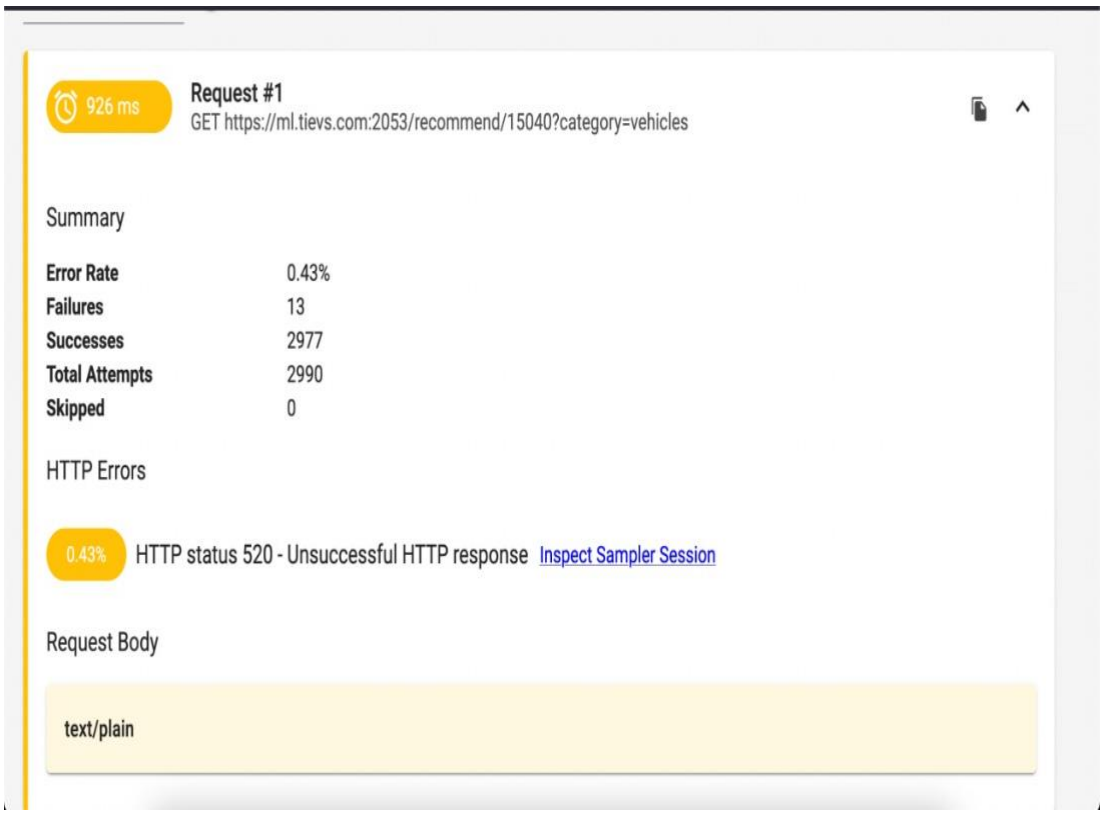


Figure 3.3 Loadmill test alternative view

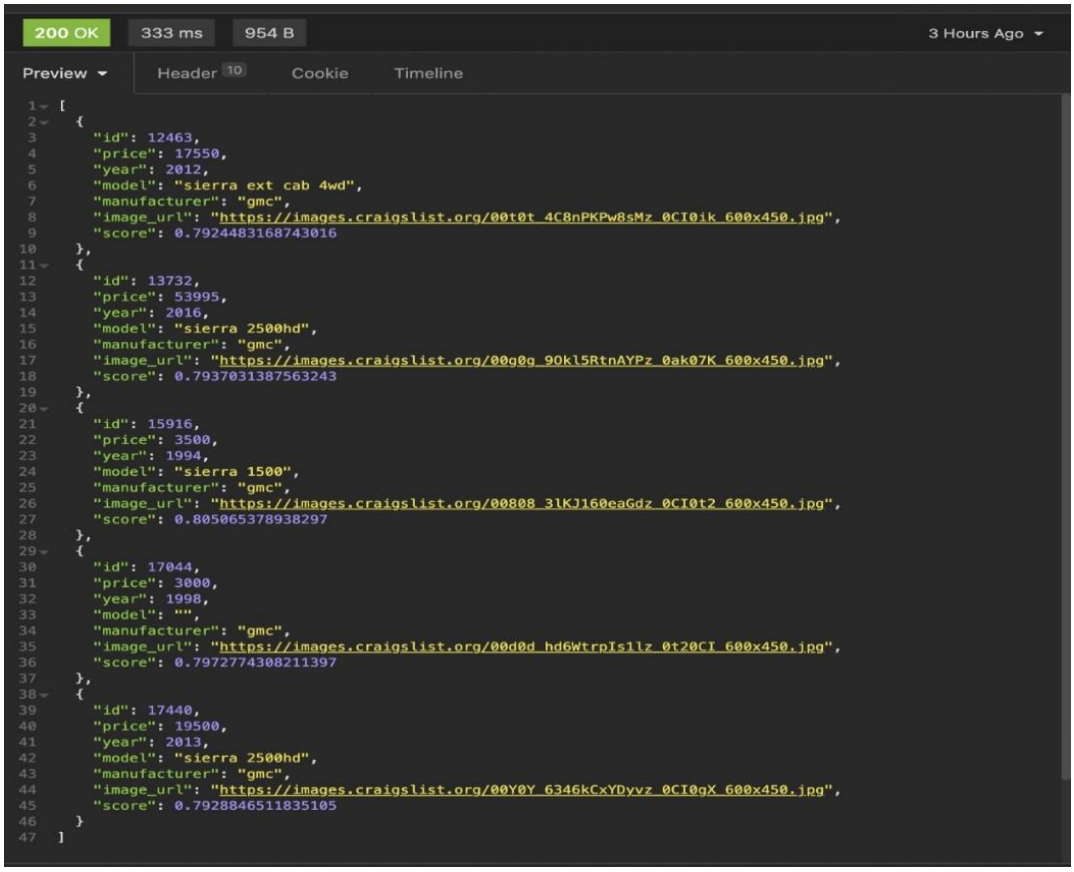
Similarly and additionally, the above shown Fig. 3.3 displays a different UI of the Loadmill load testing application, where it shows the total attempts made at the GET request towards the API endpoint shown below for the particular car advertisement id number 15040.

- https://ml.tievs.com:2053/recommend/15040?category=vehicles

The total attempts made is 2990, and from those attempts, 2977 were successful in receiving the desired response. The number of requests that ended up in errors is only

13. Hence, by analyzing those statistics, it is indisputably apparent that the implemented API call and model functionality is executing accordingly without any faults. If an error were to occur, which is extremely rare in this sense, the error would be of the type HTTP 520 status code, which is the code for an unsuccessful HTTP response. The request body is of type plain text.

However, the main takeaway here is the fact that the error occurrence probability is vastly depleted. For more clarification, below Fig. 3.4 shows the Postman result acquired in JSON format of the classified advertisements that have a higher similarity score from the cosine similarity calculation. The request was successfully executed as shown by the 200 status code, and the distinctive quality of it is that the recommendation is provided within a swift time period as only 333 milliseconds. That can be known as another major objective of implementing the discussed recommendation system for the Tievs application.




```
[
  {
    "id": 12463,
    "price": 17550,
    "year": 2012,
    "model": "sierra ext cab 4wd",
    "manufacturer": "gmc",
    "image_url": "https://images.craigslist.org/00t0t_4C8nPKPw8sMz_0CI0ik_600x450.jpg",
    "score": 0.7924483168743016
  },
  {
    "id": 13732,
    "price": 53995,
    "year": 2016,
    "model": "sierra 2500hd",
    "manufacturer": "gmc",
    "image_url": "https://images.craigslist.org/00g0g_9Okl5RtnAYPz_0ak07K_600x450.jpg",
    "score": 0.7937031387563243
  },
  {
    "id": 15916,
    "price": 3500,
    "year": 1994,
    "model": "sierra 1500",
    "manufacturer": "gmc",
    "image_url": "https://images.craigslist.org/00808_3lKJ160eaGdz_0CI0t2_600x450.jpg",
    "score": 0.805065378938297
  },
  {
    "id": 17044,
    "price": 3000,
    "year": 1998,
    "model": "",
    "manufacturer": "gmc",
    "image_url": "https://images.craigslist.org/00d0d_hd6WtrpIs1lz_0t20CI_600x450.jpg",
    "score": 0.7972774308211397
  },
  {
    "id": 17440,
    "price": 19500,
    "year": 2013,
    "model": "sierra 2500hd",
    "manufacturer": "gmc",
    "image_url": "https://images.craigslist.org/00Y0Y_6346kCxYDyvz_0CI0gX_600x450.jpg",
    "score": 0.7928846511835105
  }
]
```


Figure 3.4 Postman top similar ad result

Next, the UI view designed for an already selected car classified advertisement is being elicited by Fig. 3.5. In that view, only the necessary car characteristics are displayed along with a concise description provided regarding the vehicle from the own words of the person selling it. The seller's contact details are also being unveiled as well. Nonetheless, the UI design is being continuously reconditioned since the production goal is to realistically deploy the web application among its target market. Fig. 3.6 is the exhibit of how similar advertisements, which were recommended to the customers in a user-centric manner through the necessary API function invocations and model executions, are presented.

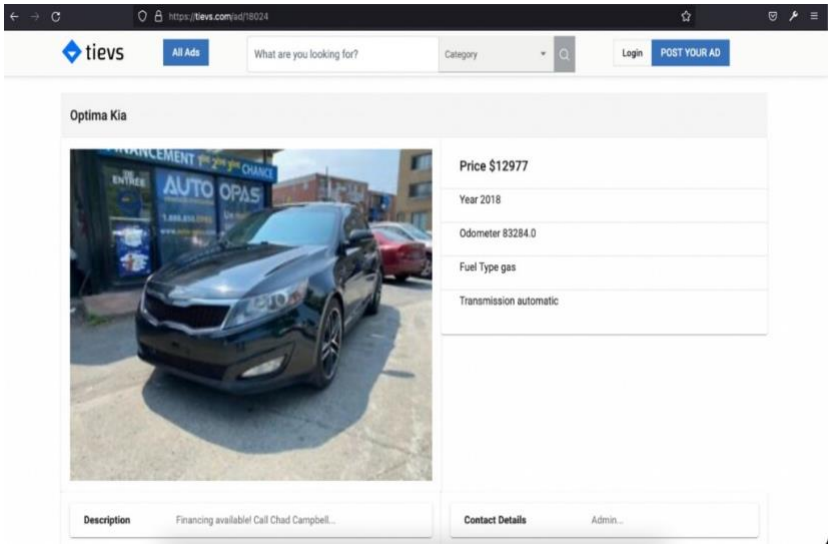


Figure 3.5 Single classified ad view

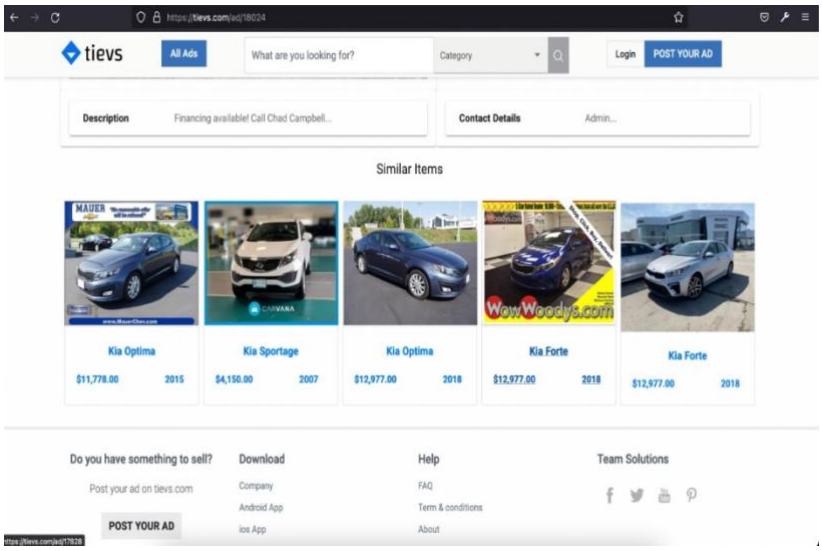


Figure 3.6 Top similar classified advertisement view

- **Price prediction system**

The adjacent Fig. 3.7 depicts the produced table, visualizing all 15 regressor results in the specific forecasting fruition order. The prediction accuracy and error rates with respect to performance criterion of the above-fitted models such as R2, RSE, and RMSE were evaluated, and a decision was made on which models performed in the most valiant and robust manner with healthy accuracies and comparatively low faulty predictions.

Out[26]:

	Model	r2_train	r2_test	d_train	d_test	rmse_train	rmse_test
0	Extra Trees Regressor	99.84	89.61	0.23	8.81	56,364.77	447,910.22
2	BaggingRegressor	98.00	89.31	4.07	10.07	199,277.80	454,298.66
10	RandomForestRegressor	93.79	87.90	9.80	13.42	350,777.32	483,407.64
4	Gradient Boosting Regressor	86.72	85.03	13.62	14.57	513,048.82	537,649.30
3	DecisionTreesRegressor	99.84	84.50	0.23	10.52	56,363.16	547,212.05
5	K Nearest Neighbours Regressor	99.84	84.19	0.23	11.60	56,983.67	552,620.97
12	Support Vector Machines Regressor	80.27	77.35	15.87	17.72	625,419.62	661,497.12
6	Lasso Regression	75.15	74.34	20.98	20.98	701,890.38	703,993.84
11	RidgeRegressor	75.16	74.32	20.88	20.96	701,707.10	704,348.04
9	MultipleLinearRegression	75.17	74.30	20.85	20.97	701,660.45	704,610.79
14	ElasticNet Regression	73.49	73.63	22.20	21.75	724,939.81	713,656.48
1	AdaBoost Regressor	72.86	72.03	22.56	22.53	733,530.31	735,074.02
7	MLP Regressor	48.96	48.04	31.52	31.51	1,005,903.24	1,001,804.14
13	LinearSVR	-59.32	-60.65	60.23	59.92	1,777,185.95	1,761,560.67
8	SGDRegressor	-52,114,148,702,837.68	-47,200,749,697,672.50	6,892,406.59	6,126,466.12	1,016,427,455,113.17	954,830,568,327.06

Figure 3.7 Regressor comparison table

When contemplating on r2_train, r2_test, d_train, d_test, rmse_train, and rmse_test values shown in the above table, it is indisputable evidence that the triumphant models from the rest of the regressors are Extra trees regressor, Bagging regressor, Random Forest regressor, and Gradient Boosting regressor. Fundamentally, r2_test, d_test, and rmse_test were the criterion strongly considered since the training set may have led to overfitting, and those values could be misleading.

In the comparison table, the R^2 value gave an implication of the goodness of regressor model fit or the variance ratio of the price values with respect to its independent estimators. However, this research decided to further analyze the R^2value's validity of the final selected four models, and for that mission, the modified version of R^2, which is adjusted R^2metric interpretation, was preferred and exploited. Thereby, a more precise view of the independent metric correlation can be provided and can certify the predictor suitability and variability as well. Following depicted calculation equations (2) and (3) are respectively of R-squared value and adjusted R-squared value.

$$R - squared \quad = \quad 1 - \frac{\sum_i (y_i - \hat{y}_i)^2}{\sum_i (y_i - \bar{y})^2} \qquad (2)$$

$$\text{Adjusted R} - \text{squared} \quad = \quad 1 - \frac{(1 - R^2)(N - 1)}{N - P - 1} \qquad (3)$$

The adjusted R-squared values of the outstanding regressors are listed below, indicating a better model fit.

Bagging Regressor – 0.956
Extra Trees Regressor – 0.972
Random Forest Regressor – 0.920
Gradient Boosting Regressor - 0.872

In addition to that, the predicted price and the actual price dispersed graphs of the four regressor models and the best fit line is visualized below by Fig. 3.8. When examining the graphs, it is again obvious that the models have performed well when predicting the price value in comparison to the actual value since the majority of the dots in the four graphs are condensed together, where the predicted and actual value representing dots are overlapping on one another while forming the linear progressing path. Only a very few dots are perceptible as scattered though it is extremely negligible in the current circumstanced having a complex dataset.

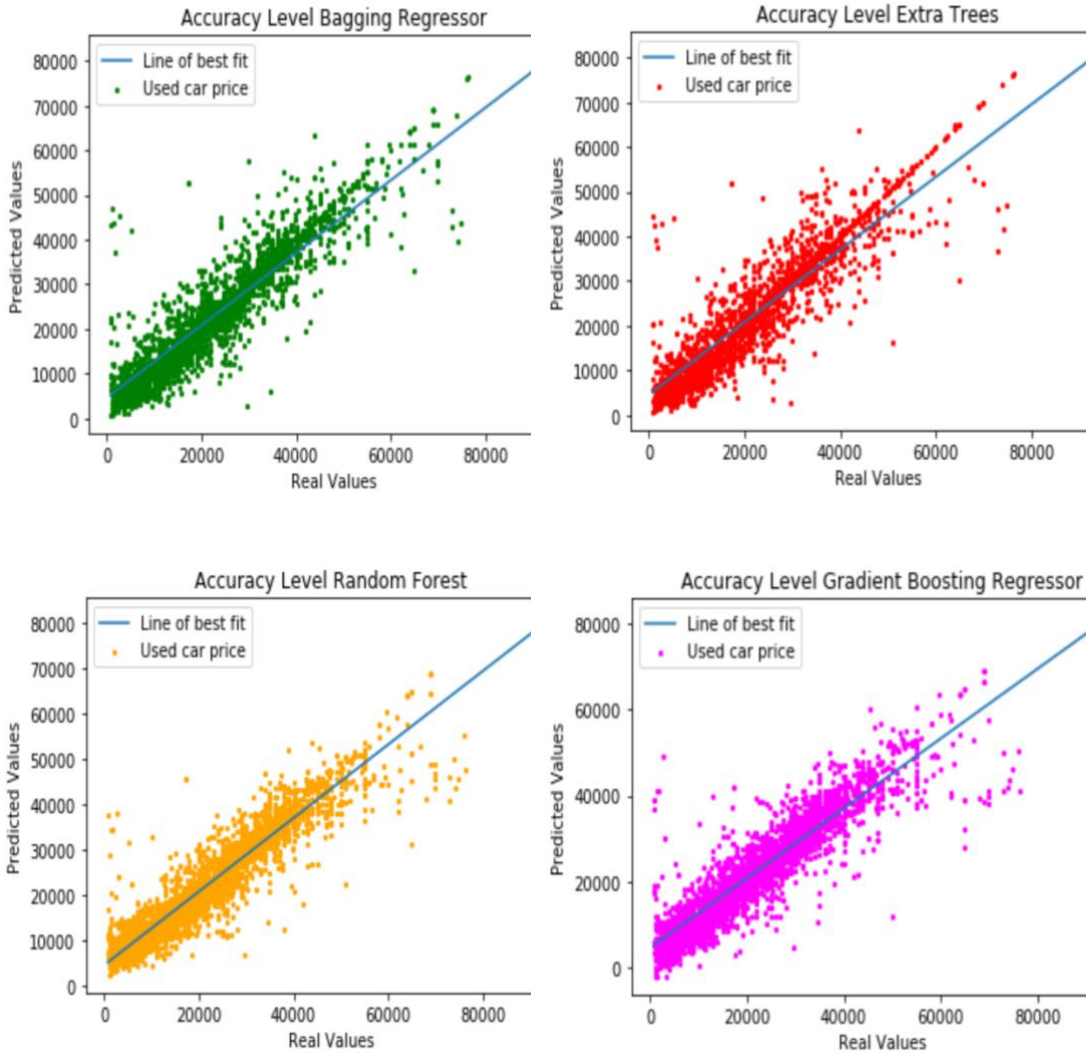


Figure 3.8 Top 4 regressor predicted values vs. actual values

Moving on, the stacked regressor for an ensemble generalization was created using the above eloquent models, and its efficiency was grandly noticeable in comparison to the solitary four models discussed above. For the base learners (level_0), Bagging, Extra Trees, and Random Forest regressors were used since those base-level learners should have the premium performance out of all four. Hence, the Gradient Boosting regressor having the marginally low prediction power was (adj. R^2 0.872) chosen as the meta-learner (level_1) for the stacking model. Following Fig. 3.9 shows the difference between the mean and standard deviation values of the four models' and the stacked regressor model's prediction scores.

Additionally, the below Fig. 3.10 provides a perspicuous view of the above mean score values graphically by using a boxplot to illustrate the marginal difference between the price predictions scores of all four models individually and the ensemble stacked regressor.

```
In [25]:
from sklearn.model_selection import RepeatedKFold
from numpy import mean
from numpy import std
# get the models to evaluate
models = get_models()
# evaluate the models and store results
results, names = list(), list()
for name, model in models.items():
    scores = evaluate_model(model, X, y)
    results.append(scores)
    names.append(name)
    print('>%s %.3f (%.3f)' % (name, mean(scores), std(scores)))
```

```
>bagging -2220.329 (82.770)
>extraTree -1920.138 (102.622)
>randomForest -3090.368 (90.082)
>gradientBoosting -3161.644 (98.659)
>stacking -1850.050 (103.634)
```

Figure 3.9 Individual and ensemble model accuracy comparison

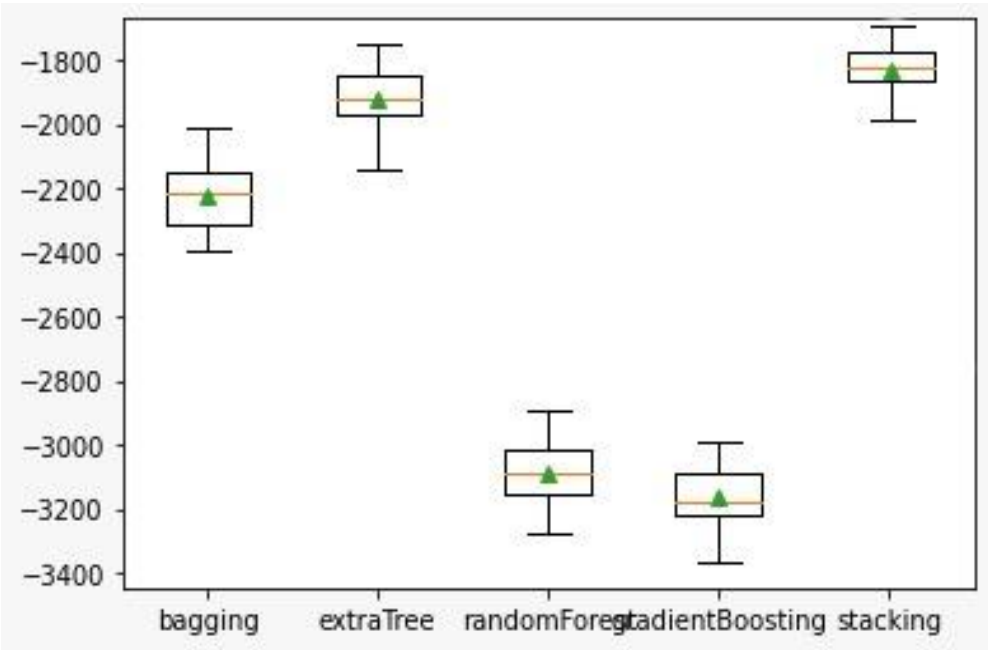


Figure 3.10 Individual and ensemble model mean score boxplot comparison

Due to the overly satisfying results received from the stacked regressor ensemble model, it was then serialized, and again deserialization with the API implemented using Flask framework, for it is utilized in the future by the client-side application when the user prompts the price prediction of a specific car being sold as a classified advertisement. All API implementation and testing details were stated in the previous chapter 'Methodology.'

After both model and API integration was done, it was deployed into the cloud server environment, and a specific domain Uniform Resource Locator (URL) was assigned to the API endpoint as well for convenient accessing by the front-end Tievs application. The price prediction facility served page is depicted by Fig. 3.11, which is a stepper page in the advertisement submission form, where the user enters all the essential characteristics of the vehicle being sold, which in turn facilitates the ML ensemble model's price prediction execution by invoking the API call from the client-side application, and finally the predicted price visualization. Thus, these results culminate in the substantial breakthroughs of this research.

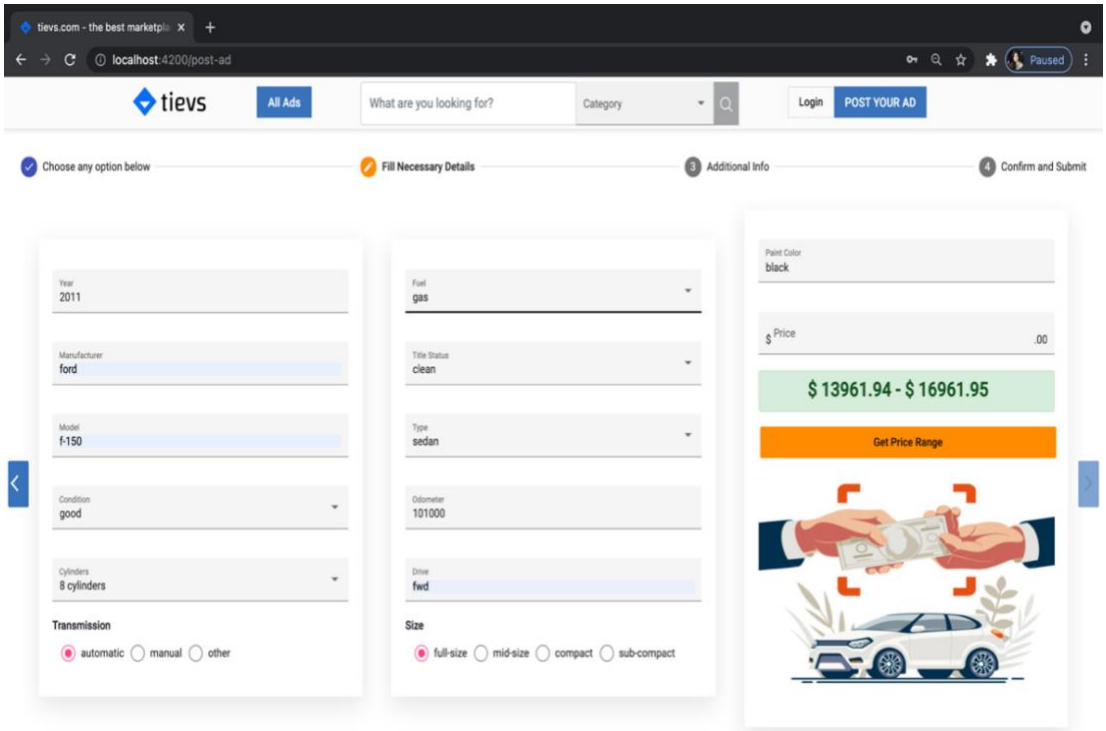


Figure 3.11 Predicted price visualization in the ad submission form

- **Fraud content detection system**

The heuristics utilized to detect and evaluate non-legitimate advertising were adequately precise. The non-legitimate made up 14.6% of the data, whereas legitimate composed 85.4% of the dataset. Although most classifiers appeared to be highly accurate, based on the F1 score, the LightGBM classifier was proven optimal, having 95.79% of precision, whereas the K-fold validation average accuracy was 97.8%.

From iteration, max-depth of 5 had the highest F1 score of 0.8 according to random grid selection of the most pertinent parameters for the LGBM classifier. Pursuant to the confusion matrix shown in Fig. 3.12, the model properly classified 2200 records as legitimate and 4 records as fraudulent.

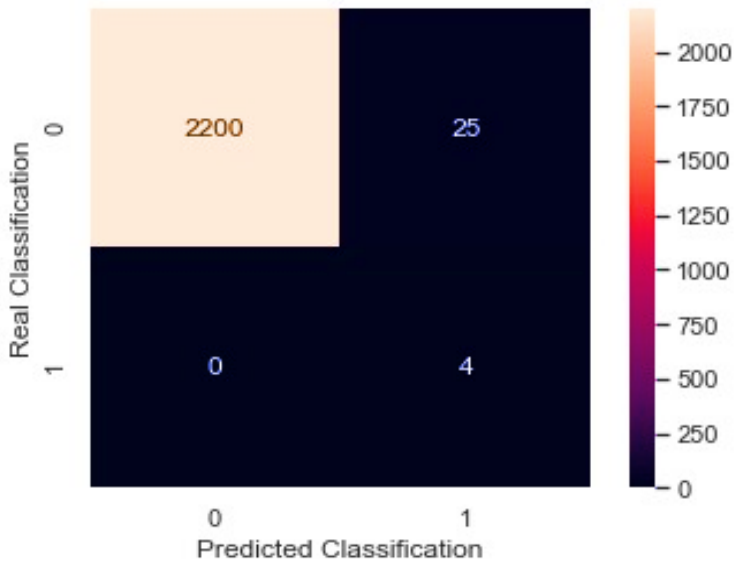


Figure 3.12 Confusion matrix visualization for LightGBM model

	Accuracy
KNeighborsClassifier	0.431579
DecisionTreeClassifier	0.894737
XGBClassifier	0.926316
XGBRFClassifier	0.926316
RandomForestClassifier	0.936842
Support Vector Machine	0.947368
LightGBM classifier	0.957895

Figure 3.13 Accuracy levels of classifiers

The Fig. 3.13 depicts the accuracy result of classifiers which are compared in a comparative analysis to obtain the best model out of seven algorithms and the Fig. 3.14 depicts the final classification reports which indicate the F1 score accuracy of the final optimum model before parameter tweaking using max_depth.

```
Classifier : LGBMClassifier
Accuracy of the model :  0.9578947368421052
Percentage of the Accuracy : 95.79%

              precision    recall  f1-score

   Not Fraud       1.00      0.95      0.98
       Fraud       0.67      1.00      0.80

    accuracy                           0.96
   macro avg       0.83      0.98      0.89
weighted avg       0.97      0.96      0.96
```

Figure 3.14 Classification report of LightGBM classifier

Conversely, it has misidentified fraudulent content as legitimate content 0 times and 25 times misidentified legitimate content as fraudulent. An important aspect of this could be discerned in Table 3.1.

Table 3.1 Classification Report of LightGBM classifier

Content Classification	F1 score	Recall	Precision	Support
Not Fraud	0.98	0.95	1.00	2200
Fraud	0.80	1.00	0.67	29

After reaching the optimum model successfully predict the likelihood of the ad being spammed is shown as illustrates in the below Fig. 3.15.

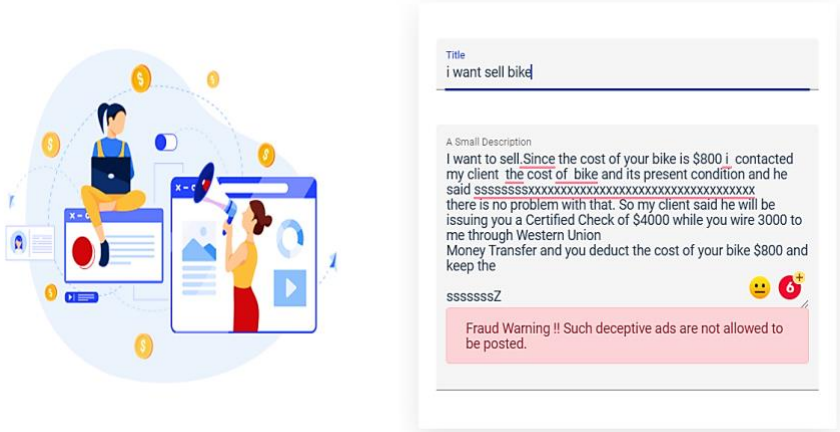


Figure 3.15 Visualization of fraud warning as the ad posting

- **Fraud image detection system**

The image recognition system was successfully able to detect 'fake' and 'not fake' images with minimal processing power and restrain inapposite images from being submitted into the Tievs application through deterrents. As given in Table 3.2, the CNN model accuracy rate is flattering, being above 90% with an adequate loss rate as well.

Table 3.2 Convolution neural network Statistics

Image Classification	Accuracy Rate	Loss Rate
Convolution Neural Network	51.16%	0.8363

Accuracy level has obtained as shown in the Fig. 3.16, after training the model using preprocessing data.

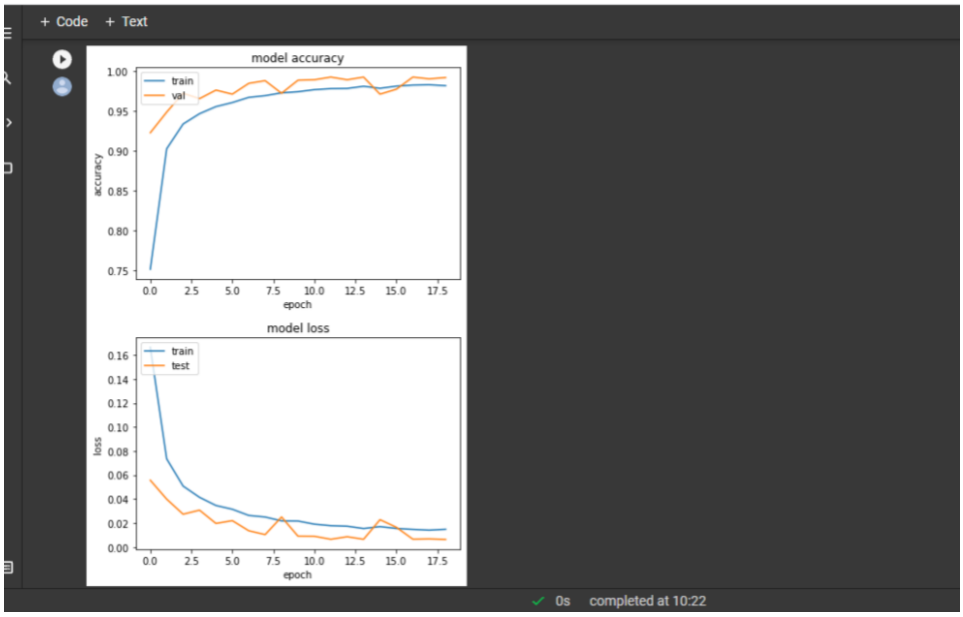


Figure 3.16 Model accuracy rate

- **Overall system results**

All of the built models, APIs, and the front-end application itself were being deployed in a cloud environment residing droplet, which was purchased through DigitalOcean IaaS cloud service provider. Below Fig. 3.17 and Fig. 3.18 display the dashboard view having all the necessary specifications.

Additionally, below shown Fig. 3.19 describes the SSH command-line interface within the Ubuntu operating system-induced droplet. In addition, the Supervisorctl tool was mainly used to restart and redeploy the API servers upon any occurred failures, and it could be verified through the following shown command in Fig. 3.20 as well, showing the API server statuses.

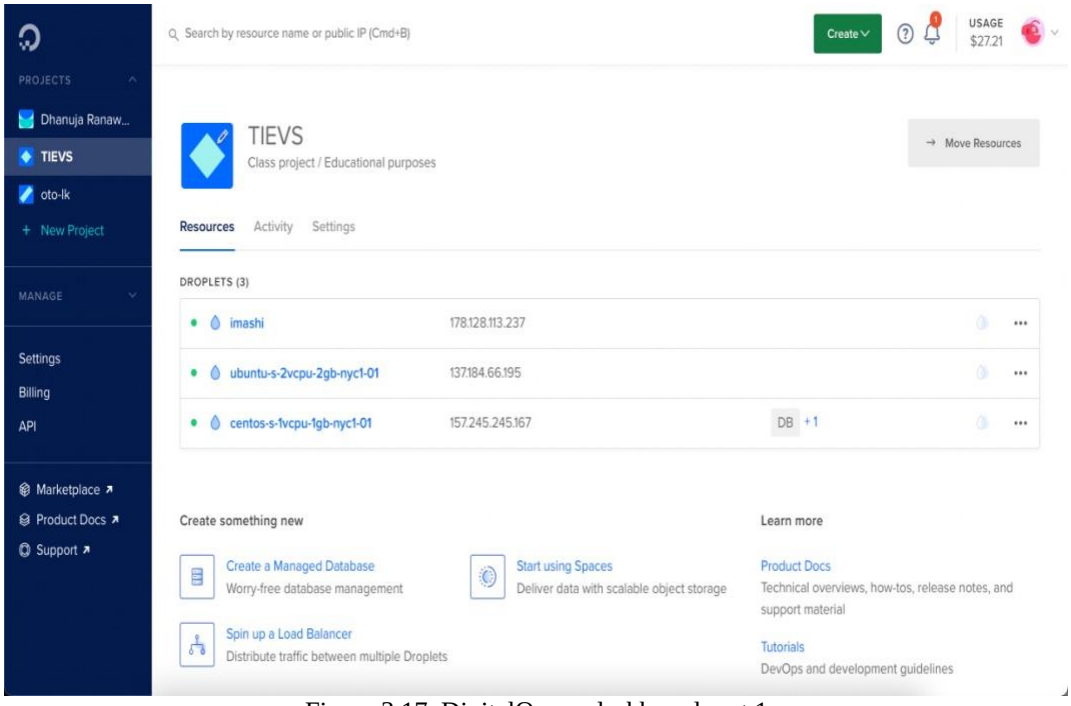


Figure 3.17 DigitalOcean dashboard part 1

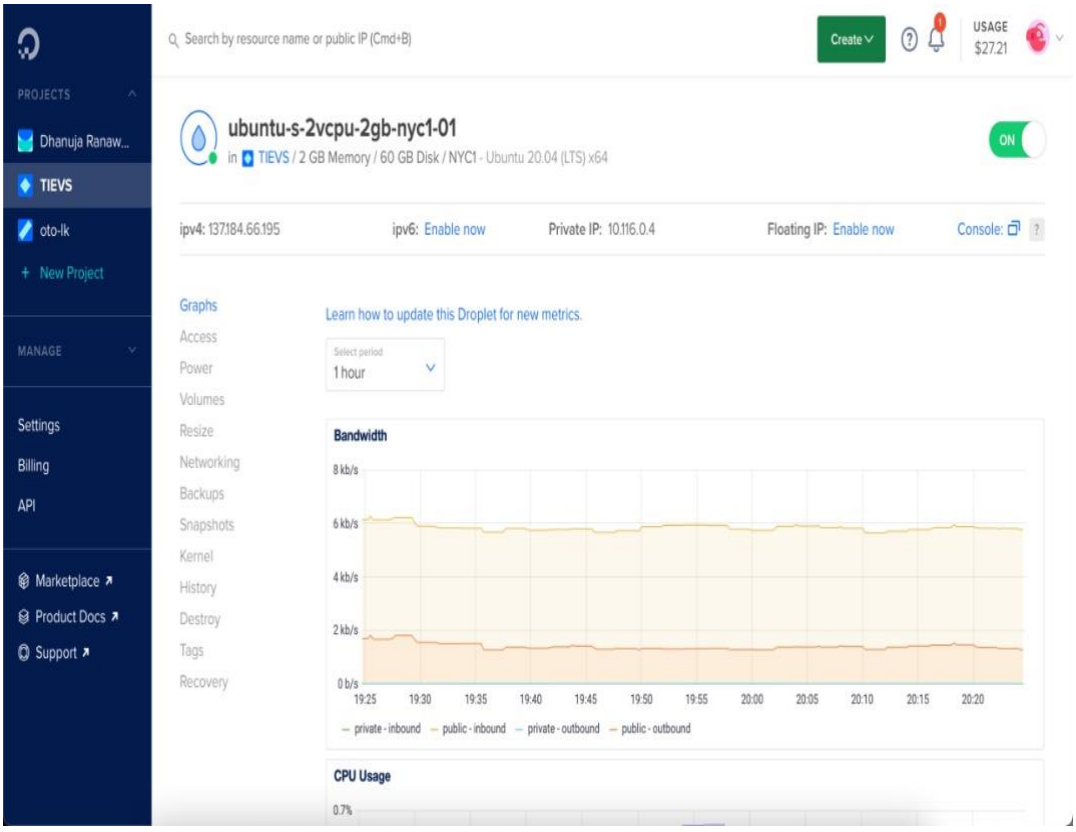


Figure 3.18 DigitalOcean dashboard part 2

```
Welcome to Ubuntu 20.04.2 LTS (GNU/Linux 5.4.0-84-generic x86_64)

 * Documentation:  https://help.ubuntu.com
 * Management:     https://landscape.canonical.com
 * Support:        https://ubuntu.com/advantage

  System information as of Thu Sep 30 14:55:22 UTC 2021

  System load:  0.0               Users logged in:       0
  Usage of /:   47.9% of 57.98GB  IPv4 address for eth0: 137.184.66.195
  Memory usage: 47%               IPv4 address for eth0: 10.10.0.7
  Swap usage:   0%                IPv4 address for eth1: 10.116.0.4
  Processes:    134

83 updates can be applied immediately.
To see these additional updates run: apt list --upgradable


*** System restart required ***
Last login: Mon Sep 27 18:41:35 2021 from 112.134.175.138
root@ubuntu-s-2vcpu-2gb-nyc1-01:~#
```

Figure 3.19 Droplet SSH tunnel

```
PriceP — root@ubuntu-s-2vcpu-2gb-nyc1-01: ~ — ssh root@137.184.66.195 — 122×39

root@ubuntu-s-2vcpu-2gb-nyc1-01:~# sudo supervisorctl status
fraudtxt-api                     RUNNING   pid 112502, uptime 2 days, 20:06:45
km-api                           RUNNING   pid 112503, uptime 2 days, 20:06:45
price-api                        RUNNING   pid 112504, uptime 2 days, 20:06:45
recsystem-api                    RUNNING   pid 145659, uptime 5:57:22
root@ubuntu-s-2vcpu-2gb-nyc1-01:~#
```

Figure 3.20 Supervisorctl status

3.2 Research Findings

Below are the major research findings from the overall study and analysis executed, with relevance to all four research components.

- The cold start problem is an important common problem that needed to be addressed within the prevailing systems, and this implemented solution was far more capable of eliminating it since it is built based on content-based filtering but ability-wise enhanced using the collaborative functionality of cosine similarity calculation and intelligent Google USE model.

- Real-time recommendations could be provided to the consumers without much trouble due to that reason. Parallelly submitted advertisements were examined, and the necessary embeddings were created promptly to facilitate the dynamic, updated suggestive mechanism, which happens continuously according to user requests received. This result gained by this research component, which is similar to an implicit model retraining circumstance, is truly a remarkable found that distinguishes the engine from other competitors.

- Moreover, due to discontinued usage of customer personal information, it was found that the Tievs application incorporating this recommendation engine was free of any possible user privacy-related allegations made by particular law enforcement. It was firmly evident that such penalties would not be able to cause any ructions.

- The concurrency within the recommendation engine itself and the speed it delivers is immensely astonishing and a great find for this research analysis. Even from the start of this experimentation procedure, one of the main focused aspects was the speed of the recommended procedures considering the fact that it handles the calculations of top similar ads from a complex dataset having many records. That high speed was effortlessly explored and encountered at the end.

- Since the solution is deployed in the cloud (DigitalOcean) as droplets, there is a high chance of scalability when required, which can be known as a tempting advantage for such classified advertising applications. Due to this scalability architecture, it was discovered that fault tolerance and disaster recovery mechanisms could also be enacted, which could further improve the recommendation system's stability and continuous availability for the betterment of its consumers.

- Furthermore, it was encountered that utilizing a classified application's own ML-based procedure for the evaluation of a price value for a particular selling item was quite advantageous for the application's internal functionality and the system developers' expediency. Though some existing classified platforms utilize website crawlers or similar third party appliances to provide their customers the price evaluation of an item they are selling, it is troublesome due to ambivalent network traffic originating to and from the servers providing the price evaluation service, sudden server failures, incompatible or rigid attributes under consideration and future system augmentation immobilization. Developers won't have the intrinsic authority and control of the price evaluation procedures, and if faults were to occur from the dependent third parties, the application would lose price evaluation functionality altogether, making it incapacitated and risk losing reputation among consumers as well.

- Out of all of the independent estimators of the price value, the make year and the mileage value were found the most influential from the study of the dataset. The other 11 attributes considered (mentioned in the 'Methodology') had only an adequate impact on price though those weren't omitted completely from applying into the prediction model. In addition to that, it was found that when using the K-fold cross-validation (CV) method, the model tendency to overfit was significantly decreased to a lower level. Training and testing the models with a different number of folds and 'K' times lead to analyzing the ML models' ability to predict unseen or new data. Due to that, the models were not overly or exactly matching against the

training dataset, and they were performing poorly when it came to the test dataset. Without using the cross-validation method, the training accuracy seemed excessively large, indicating overfitting error.

- The regression-based supervised machine learning method was found highly appropriate for this research since the dependent variable is a continuous value to be predicted. When contemplating on the results obtained, it was conceivable that most of the selected 15 regressors models and even all of the finally chosen winning models were all of decision tree type algorithms rather than being linear regression or neural network type. By analyzing the comparison table with the final values of 15 regressor performance criteria, it is apparent that decision trees led to better prediction capabilities. It was found that the intuitive performance and the fewer data preprocessing necessity, along with not requiring scaling or normalization of values to a significant extent, made those decision trees models function well withstanding minor insignificance and being harshly affected. However, excessive resource consumption and higher time for the training were identified as a considerable drawback as well.

- This research incorporated ensemble learning under the rudiment hypothesis that it will eminently perform exceptionally well when comparing with individual algorithm models. For this purpose, stacked regressor generalization was utilized. It is usually tedious to encounter a model which performs best on a particular dataset. Hence, through an alternative combination of such good models, a sublime performance was established, which outperforms solidarity models. All in all, the hypothesis was ultimately proven and found true when examining the results generated from this research regarding the ensemble model.

- Nonetheless, as stated in the prior sections, the greatest efficiency was attained by employing a classification matrix. As a result, the LightGBM has the highest precision, and the f1 score is one of the best metrics for measuring efficiency,

according to a review of the research literature. Finally, we focused our efforts on obtaining a high f1 score; as previously indicated, the f1 score has a maximum value of 1 and a minimum value of 0. Following several iterations, it was discovered that max depth 5 delivers the best F1 score.

- The extant literature's contents detection studies prompted the exploration of various types of classification tasks utilizing various databases. Using a literature review, it was possible to investigate how detection works in the early stages of several machine learning algorithms such as Random Forest, Decision Tree, and Support Vector Machines. Despite the fact that the background of fraud detection investigations assisted in developing the best solutions using these algorithms, the boosting algorithms such as LightGBM, XGBRF, and XGB for fraudulent content identification in classifieds advertisement matters have never been extensively evaluated.

- The literature review assisted in the choice of stemming over lemmatization since lemmatization yields more exact results. As a result, Snowball is more voracious than the Porter stemmer. Many of the Snowball stemmer's characteristics were inspired by problems experienced with the Porter stemmer.

- Many existing classified platforms provide their customers with a red flag option if they encounter fraud. However, this is not ideal since it increases the likelihood of the website being spammed and leads to higher reputation maintenance fees. It is fairly traumatic since the website's reputation may decrease if the claimed fraud count was relatively large, then programmers have to maintain the danger transpired, which is time-consuming. Concerning the problem at hand, our system allows for proactive detecting fraud behavior, which helps to minimize the likelihood of the website being spammed

- This research, use deep learning methods to classify the two classes: vehicle and no vehicle. This technique allows the development of discriminant functions

immediately on the basis of data without any prior knowledge of the feature extraction procedure. Utilize convolutional neural networks (CNN), which have obtained extremely good results in numerous computer vision applications over the last several years, including image classification, object identification, image segmentation, and many more.

- This enables us to essentially tackle two problems: the first is a lack of data for training. Second, it enables us to adjust the size of the input data in accordance with the related and unrelated, even without augmentation methods or extra geometric deformations. Furthermore, because the dataset lacks a segmented annotation, a simple and efficient approach is utilized to determine the mask of the false section based on the information included in the fake images. Model is a convolutional neural architecture.

3.3 Discussion

The presence of the Cold Start problem, which has been prevailing in most collaborative-based recommendation systems, has been omitted through the methodology used in this study since content-based recommendation procedures were undertaken. When the user data such as likes, dislikes, feedback, comments, etc., are not accounted for, the Cold Start problem implicitly moves out of the question. Besides, classified advertising platforms, unlike most other e-commerce applications, do not save customer-related data such as their likes, dislikes, comments, feedback, reviews, and other private preferences, hence the use of collaborative recommendation was not that significant to proceed with this study's major goal. Through the discontinued usage of these user details, some of which could be tagged confidential, Tievs was made immune to possible penalties or fines that would have been effectuated by various law enforcement regarding potential customer data and information disclosing.

The real-time recommendation was designed and developed for the customers of the Tievs application by incorporating concurrency routines that were implemented through the Golang API as Goroutines. In usual recommendation scenarios, when a new advertisement is submitted into the system, and different customers access the application from another session, they would not receive newly published classified included recommendations instantly. It may take a few minutes or hours for all calculations to happen and the respective suggestions to arise. That is why the real-time recommendation is preferred, especially in online classified platforms.

The deployment of the model included APIs, and all their dependent files within a cloud environment were done at the end of all critical component implementation. The droplet, which was purchased from the IaaS platform DigitalOcean, provided high availability, scalability, and flexibility for the APIs and the Tievs front-end application as well. Such courses of action were not regulated or organized by existing studies that focusing recommendation, price prediction, fault context detection, or even fault

image detection. Not only about the engine establishing factor, but also the deployment aspects must be addressed without negligence if all those components need to satisfy its main tasks without availability-wise interruptions. Otherwise, those models and APIs themselves would be worthless, and that is why this research hoped to attempt on that notion and consequently succeeded.

It was skeptical initially regarding the attainability of the comparative study of 15 different regressors for implementing the price prediction system, which seemed quite challenging when considering this research's scope. However, the goal was accomplished by the smooth but rigorous execution of the procedures. Previous studies [9]-[16] had done similar comparative studies on multiple ML algorithms, although they were limited to a smaller number, not exceeding 6. In that sense, this research advances further in the strategy undertaken.

Another rationale on why this research is significant and innovative when comparing with similar existing research is that most of them have only evaluated the algorithm prediction performances by analyzing them individually with the relevant hyperparameters. Ensemble learning is rarely implored, especially in car price prediction [9]-[15]. The study [16], being divergent, employed only a classification-based ensemble model. Hence, investigations on the regressors' capacity to work as a group for prediction are further asserted obscure. Contrastingly, this research expanded the regressor ensemble research area by filling those gaps. Since ensemble learning was appointed as one of the main targets of this research, the above four regressors were combined together using a heterogeneous ensemble learning technique called stacked generalization. To build the stacking regressor, Extra Trees, Bagging, RF were used as the base learners since those three were at the top from all four, and Gradient Boosting regressor was chosen as the meta-learner, and it gave the final predictions according to the prediction results obtained from the base learners. Although the top four regressors were commendable when performing individually, it was found while proving the initial hypothesis at the beginning of this research, the

stacked or the ensemble model performed even more exceptional when predicting the price value according to the car features provided. This hybrid prediction power having commendable generalization, as depicted in the 'Results' subsection, unquestionably escalates novelty and is a huge find while being an asset to this research results.

From the perspective of an application programmer, rigorous program logic must be implemented to identify and prevent fraudulent advertising from being published, either accidentally or intentionally, utilizing verification and validation approaches. If those advertising were uploaded, buyers may eventually see incorrect ads and lose trust in utilizing the program for their requirements, causing the platform's image to suffer. Current apps do not prioritize, much alone monitor and avoid the aforementioned events. Furthermore, boosting algorithm learning has never been employed, especially in automotive fraud classifieds ad content. Furthermore, Angular-integrated functions handle databases and API service calls from the front-end application, while the components handle the UI elements and their colorful, correct replies even without frequent page refreshing. It effectively serves as a back-end replacement, needing no additional code. Because the model backend implementation was appropriately built with flask subsystem.

The authors of this research distributed a questionnaire among potential classified advertising platform users and received 1005 responses, where the majority was declared Male, aging from 20 to 60 years. The main goal of this survey was to validate Tievs system’s potential in delivering the customer’s classified advertising necessities efficiently and cost-effectively, without causing any disturbances along the flow. Below Fig. 3.21 depicts the responders’ preference as a ratio to recommend Tievs portal to others in future. Fig. 3.22 is responsible for displaying the effectiveness of each main components having ameliorated functionalities that are crucial for the overall system’s proficiency. Lastly, the Fig. 3.23 illustrated the all-around review for the Tievs system, given by its prospective users.

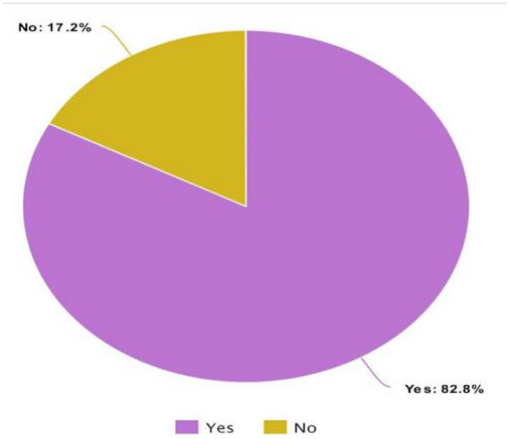


Figure 3.2121 User preference for Tievs web-application recommendation

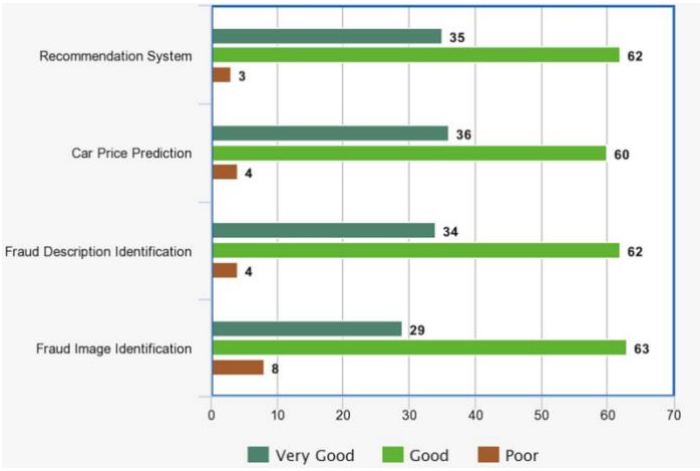


Figure 3.2222 Effectiveness of each main component in Tievs

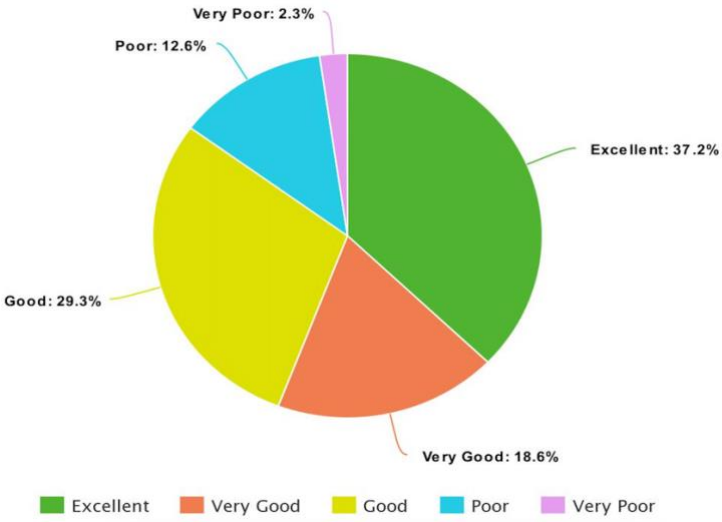


Figure 3.2323 Overall user review for Tievs system

3.4 Summary of Each Student's contribution

Table 3.3 IT18122060 Contribution Summary

Member	Component	Tasks
R. A. D. Prathapa	Optimized Classified Advertisement Recommendation System	• Investigate suitable Machine Learning and Natural Language Processing algorithm types available with regard to recommendation systems. • Exploring and collection of the most convenient dataset that could be used for model training. • Preprocessing the dataset necessarily prior to model utilization • Finding the most suitable model for recommendation engine development. • Identifying and importing the necessary dependencies to import the model and training of the model. • Successful model serialization and deserialization. • Meticulous implementation of the API using Golang while providing for concurrency and handling of all essential tasks through function invocations. • Build relevant client and server-side components to visualize the analyzed and predicted customer-specific search results and retrain the model using new information acquired from the application database (new embeddings). • Smooth integration between front-end and back-end services.

Table 3.4 IT18113532 Contribution Summary

Member	Component	Tasks
K.S.S. Bandaranayake	Used Car Ensemble Price Prediction Model	• Analyze existing literature to find out what has and has not been done with regard to attempts on used car price prediction. • Identify the most eligible ML algorithms that can perform well with regard to price prediction. • Accumulate an adequate dataset having all essential characteristics that will prompt a higher accuracy during the ML model implementation. • Perform necessary feature engineering and data preprocessing methods along with exploratory data analysis to get a better depth of the dataset being utilized. • Conduct a comparative study on 15 different algorithm models, train and test them using refined hyperparameter to obtain higher prediction accuracy and low error rates through specific performance metrics (R^2, adj. R^2, MAE, MSE, RMSE, RSE). • Selection of the top four exemplary models out of all 15 models through metric analysis and creation of an ensemble model following the stacked generalization architecture for a powerful combined prediction. • Implement the ML API to allow the ensemble model to be integrated and utilized through an HTTP POST endpoint and deployment of the price prediction API in a cloud environment. • Front-end application necessary component design and development. • Integration of price prediction API and Tievs front-end application's advertisement submission form to accept predict price request and final predicted price range visualization.

Table 3.5 IT18089400 Contribution Summary

Member	Component	Tasks
Ravihari J.M.S	Intelligent Classified Fraud Context Detection System	• Congregate the appropriate data set using external sources, or survey using craigslist region and an initial data records should be labeled as fraud and non-fraud inability. • Exploring and collection of the most convenient dataset that could be used for model training. • Preprocessing the dataset is required, as is exploratory data analysis, in order to successfully complete the data preparation process and feed the model. • Explore the most suitable machine learning algorithm considering the accurateness of the output speed or training time. • Comparative analysis using classification matrix and exposed the optimum model. • Hyperparameter tweaking for optimization for the revealed optimum model. • Fraud content detection Flask API implementation, Embedding the Mode importing the model pickles with the application without inconveniences. • Build client and server-side components and extract the data from the advertisement form, prior to posting. • Visualization on the front end with a warning message prior to posting, if the advertisement is consisting of inappropriate content.

Table 3.6 IT18082548 Contribution Summary

Member	Component	Tasks
Madhushani S.K.A.K.I	Fraud Image Detection System	• Collection of most appropriate data records • Accomplished the data preprocessing and exploratory data analysis technique for having an effective dataset for training and validation process • Training and testing model on data • Evaluate the car detection prediction model on trained data • The JPEG characteristics algorithm and GRAY scaling inclusion for effectively identify the image fraudulent activities. • Successfully Predict the car brand And image detection server implementation. • Successful front-end and back-end integration and cloud deployment. • Implementation an API using Django Framework and Django REST. Both are the python base. • Identifying and blocking fraudulent/conflicting pictures from being submitted in the application on a proactive basis. • Notify the user of cautious messages if they repeatedly attempt to upload such irrelevant/misleading images. • Visualization on Tievs front-end application

4 CONCLUSION

Tievs is an ameliorated, systematic, and highly user-friendly system for classified advertising. As for the final conclusion of Tievs with respect to the whole system's performance metrics, it is distinctly apparent that the application is functioning remarkably well in terms of speed, website vitals, performance grades, etc. as shown in the below Fig. 4.1 and Fig. 4.2, which were primarily extracted from GTmetrix online performance testing tool.

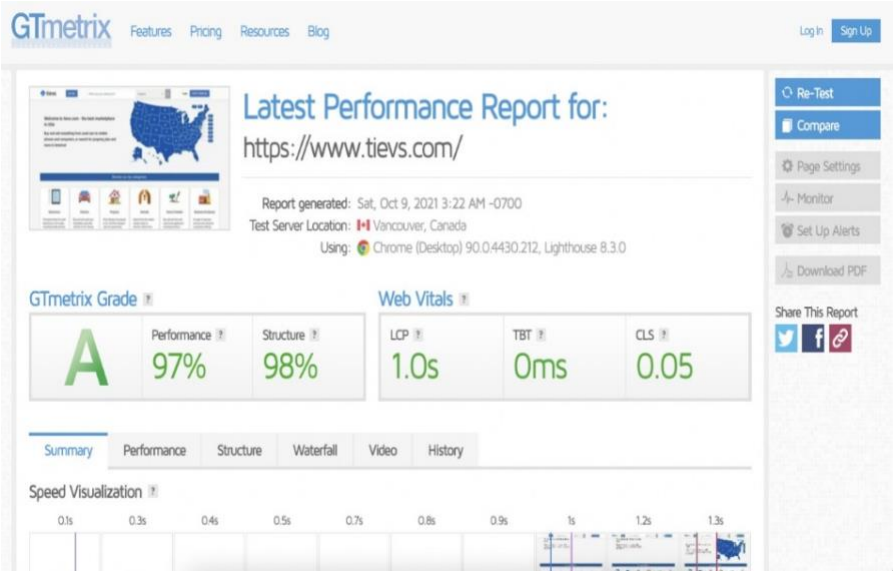


Figure 4.1 GTmetrix performance view part 1

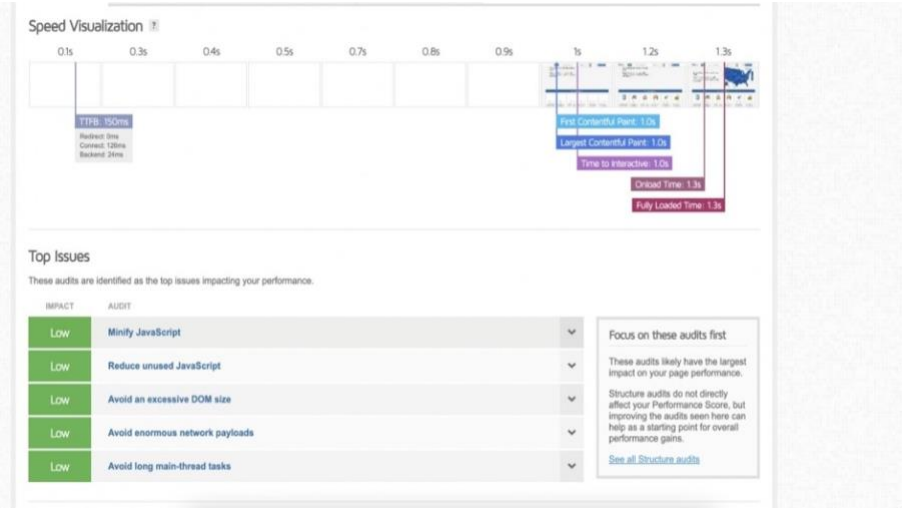


Figure 4.2 GTmetrix performance view part 2

Tievs consists of an innovative recommendation, dynamic price prediction, and an intelligent content/image identification and prevention system to gratify consumers' requirements when browsing or publishing advertisements. The novel recommendation engine pertaining to Google USE with DL and NLP techniques and cosine similarity calibrates relatedness of classifieds according to user interacted ad details while reducing the cold start problem and making the process more simple, scalable, and cost-effective forging the application trendier as opposed to traditional techniques. By collaborative filtering enhancements done hereafter where user data is acquired after application launch will provide more superlative classified advocacy .

The regression-based ensemble model, consisting of 4 impenetrable regressors, provides strong price prediction for used cars with an accuracy of above 85% and even 90% in some instances, relieving the consumers from their intensive price value establishment when posting ads. Authors intend to tune model parameters and increase training data while experimenting on more sophisticated algorithms as genetic, fuzzy logic, and neural networks, leading to online model retraining. Optimal classifier LightGBM achieved pertinent parameters out of analyzed models through supervised learning to identify fraud advertisement context prior to ad submission. Perhaps some tweaking to the training phase with a larger fraud content dataset, presuming to augment recall rate precisely while reducing precision loss, could be done in the foreseeable future.

Identifying fraudulent images was accomplished satisfactorily by giving attention to car image features using the CNN model, where it certified image legitimacy explicitly to prevent hoaxes. Furthermore, the authors anticipate determining the color of car images and their originality against submission as well. Nonetheless, this classified advertising web application was heightened with beneficial qualities to assuredly emphasize online advertising stature and score competitive advantages against other prevailing systems. As a future avenue for the overall system, authors will devote themselves to expand the services presently orchestrated by embracing other types of

classifieds such as real estate, electronics, pets, jobs, etc. Hence broadening the target consumer audience altogether is plausible. The regression-based ensemble model, consisting of 4 impenetrable regressors, provides strong price prediction for used cars with an accuracy of above 85% and even 90% in some instances, relieving the consumers from their intensive price value establishment when posting ads. Authors intend to tune model parameters and increase training data while experimenting on more sophisticated algorithms as genetic, fuzzy logic, and neural networks, leading to online model retraining.

Optimal classifier LightGBM achieved pertinent parameters out of analyzed models through supervised learning to identify fraud advertisement context prior to ad submission. Perhaps some tweaking to the training phase with a larger fraud content dataset, presuming to augment recall rate precisely while reducing precision loss, could be done in the foreseeable future.

Identifying fraudulent images was accomplished satisfactorily by giving attention to car image features using the CNN model, where it certified image legitimacy explicitly to prevent hoaxes. Furthermore, the authors anticipate determining the color of car images and their originality against submission as well. Nonetheless, this classified advertising web application was heightened with beneficial qualities to assuredly emphasize online advertising stature and score competitive advantages against other prevailing systems.

As a future avenue for the overall system, authors will devote themselves to expand the services presently orchestrated by embracing other types of classifieds such as real estate, electronics, pets, jobs, etc. Hence broadening the target consumer audience altogether is plausible.

REFERENCES

[1] Business Insider. 2021. CHART OF THE DAY: Newspaper Billions Become Craigslist Millions. [online] Available at: https://www.businessinsider.com/chart-of-the-day-craigslist-vs-newspaper-2009-6 [Accessed: 23-Feb-2021].

[2] S. Jones, “Online classifieds,” Pew Research Center: Internet, Science & Tech, 30-May-2020. [Online]. Available: https://www.pewresearch.org/internet/2009/05/22/online-classifieds/. [Accessed: 06-Oct-2021].

[3] “Online classified ads: Digital, dynamic, and still evolving,” McKinsey & Company, 03-Apr-2018. [Online]. Available: https://www.mckinsey.com/industries/technology-media-and-telecommunications/our-insights/online-classified-ads. [Accessed: 23-Feb-2021].

[4] R. Singla, S. Gupta, A. Gupta, and D. K. Vishwakarma, “FLEX: A Content Based Movie Recommender,” in 2020 International Conference for Emerging Technology (INCET), 2020, pp. 1–4, doi: 10.1109/INCET49848.2020.9154163.

[5] T. Badriyah, S. Azvy, W. Yuwono and I. Syarif, "Recommendation system for property search using content based filtering method," 2018 International Conference on Information and Communications Technology (ICOIACT), 2018, pp. 25-29, doi: 10.1109/ICOIACT.2018.8350801.

[6] A. Pal, P. Parhi, and M. Aggarwal, “An improved content based collaborative filtering algorithm for movie recommendations,” in 2017 Tenth International Conference on Contemporary Computing (IC3), 2017, pp. 1–3, doi: 10.1109/IC3.2017.8284357.

[7] Cer, Daniel & Yang, Yinfei & Kong, Sheng-yi & Hua, Nan & Limtiaco, Nicole & John, Rhomni & Constant, Noah & Guajardo-Cespedes, Mario & Yuan, Steve & Tar, Chris & Sung, Yun-Hsuan & Strope, Brian & Kurzweil, Ray, “Universal Sentence Encoder”, 2018.

[8] S. Shaikh, S. Rathi and P. Janrao, "Recommendation System in E-Commerce Websites: A Graph Based Approached," 2017 IEEE 7th International Advance Computing Conference (IACC), Hyderabad, India, 2017, pp. 931-934, doi: 10.1109/IACC.2017.0189.

[9] Ganesh, Mukkesh & Venkatasubbu, Pattabiraman, “Used Cars Price Prediction using Supervised Learning Techniques”, *International Journal of Engineering and Advanced Technology*, vol. 9, Dec., pp. 216-223, 2019.

[10] Kuiper, Shonda. "Introduction to Multiple Regression: How Much Is Your Car Worth?", *Journal of Statistics Education 16.3*, 2008.

[11] N. Kanwal and J. Sadaqat, “Vehicle Price Prediction System using Machine Learning Techniques,” *International Jounal of Computer Applications*, vol. 167, no. 9, pp. 27–31, 2017.

[12] N. Monburinon, P. Chertchom, T. Kaewkiriya, S. Rungpheung, S. Buya and P. Boonpou, "Prediction of prices for used car by using regression models," 2018 5th International Conference on Business and Industrial Research (ICBIR), 2018, pp. 115-119, doi: 10.1109/ICBIR.2018.8391177.

[13] N. Pal, P. Arora, S. Sumanth Palakurthy, D. Sundararaman, P Kholi, " How much is my car worth? A methodology for predicting used cars prices using Random Forest," Future of Information and Communications Conference (FICC), 2018, pp. 1-6.

[14] Xinyuan Zhang , Zhiye Zhang and Changtong Qiu, "Model of Predicting the Price Range of Used Car", 2017.

[15] S. Pudaruth "Predicting the Price of Used Cars Using Machine Learning Techniques," *International Journal of information & Computation Technology,* vol. 4, no. 7, pp. 753-764, 2014.

[16] E. Gegic, B. Isakovic, D. Keco, Z. Masetic, J. Kevric, "Car Price Prediction using Machine Learning Techniques," *International Burch University, Sarajevo, Bosnia and Herzegovina,* vol. 8, no. 1, pp. 113-118, 2015.

[17] V. Garg and S. Nilizadeh, "Craigslist Scams and Community Composition: Investigating Online Fraud Victimization," in *2013 IEEE Security and Privacy Workshops,* San Francisco, CA, USA, May 2013, pp. 123–126. doi: 10.1109/SPW.2013.21.

[18] D. Fetterly, M. Manasse, and M. Najork, "Spam, damn spam, and statistics: using statistical analysis to locate spam web pages," in Proceedings of the 7th International Workshop on the Web and Databases colocated with ACM SIGMOD/PODS 2004 – WebDB 04, Paris, France, 2004, p. 1. doi: 10.1145/1017074.1017077.

[19] H. Tran, T. Hornbeck, V. Ha-Thuc, J. Cremer, and P. Srinivasan, "Spam detection in online classified advertisements," ACM Int. Conf. Proceeding Ser., vol. 11, Apr. 2011, doi: 10.1145/1964114.1964122.

[20] Y. Guo, X. Cao, W. Zhang, and R. Wang, “Fake Colorized Image Detection,” *IEEE Trans. Inf. Forensics Secur.*, vol. 13, no. 8, pp. 1932–1944, Aug. 2018, doi: 10.1109/TIFS.2018.2806926.

[21] J. Xingteng, W. Xuan, and D. Zhe, “Image matching method based on improved SURF algorithm,” in 2015 IEEE International Conference on Computer and Communications (ICCC), Oct. 2015, pp. 142–145. doi: 10.1109/CompComm.2015.7387556.

[22] S. Al-Rousan, A. Abuhussein, F. Alsubaei, L. Collen, and S. Shiva, “Ads-Guard: Detecting Scammers in Online Classified Ads,” in *2020* IEEE Symposium Series on Computational Intelligence (SSCI*)*, Dec. 2020, pp. 1492–1498. doi: 10.1109/SSCI47803.2020.9308544.

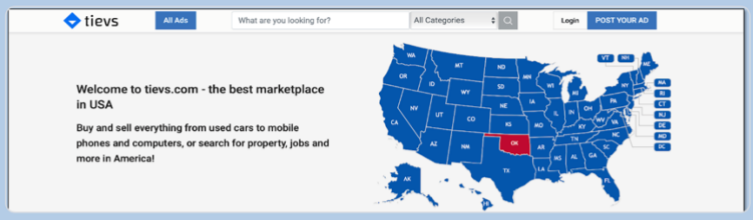
[23] Chaudhary, A., *Universal Sentence Encoder Visually Explained*. [online] Amit Chaudhary. 2021. Available at: <https://amitness.com/2020/06/universal-sentence-encoder/> [Accessed 4 October 2021].

[24] KDnuggets, “Ensemble Learning to Improve Machine Learning Results - KDnuggets,” *KDnuggets*, 2021. [Online]. Available: https://www.kdnuggets.com/2017/09/ensemble-learning-improve-machine-learning-results.html/2 [Accessed 4 October 2021].

APPENDICES

Appendix A: User Feedback Questionnaire for Tievs

Online Survey Link - https://forms.gle/TyCEopchFp7UsR4A7



"Tievs" Classified Advertising System's User Feedback Questionnaire

This questionnaire is used to gather customer feedback for the purpose of a SLIIT final year research project "Tievs: Classified Advertising Enhanced Using Machine Learning Techniques", which is being supervised by Ms. Manori Gamage and co-supervised by Ms. Suriyaa Kumari. The main expertise and reference person is Mrs. Wasanthi Bandaranayake.

Tievs is simply an ameliorated classified advertising platform that ventures on further to cater to the customer requirements of classified advertisement submission and exploration. The functionalities being provided by this system are being advanced through Machine Learning, Deep Learning, and Natural Language Processing concepts. The major components of Tievs are the optimised recommendation system, the independent classified advertised item price prediction system, fraudulent content detection, and prevention system, and finally erroneous image identification and avoidance system. Due to scope limitations, car classifieds are being effectuated for the time being although expansion for other classified types are underway.

Please visit the below URL to redirect to the Tievs domain where you are free to navigate through the application as you wish. Simultaneously, you can provide your valuable feedback by filling out this survey, with respect to your experience and perspective of Tievs classified advertising application.

https://tievs.com/

We ensure the data will remain anonymous as well as data will only be used for research purposes and will not be used for any commercial purpose.

By choosing to continue you will be giving your consent to participate in the study. Please confirm your consent below. *

- [ ] I give my consent to participate in this study

Please answer the following questions to the best of your knowledge.

1) Enter your age. *

Your answer

2) State your gender. *

- ( ) Female
- ( ) Male
- ( ) Prefer not to say
- ( ) Other: _____

3) From the following list, which are the applications that are more familiar with you with respect to classified advertising? *

- [ ] Craiglist
- [ ] Quikr
- [ ] Olx
- [ ] Carwale
- [ ] CarDheko
- [ ] Pakwheels
- [ ] ikman.lk
- [ ] KSL.com
- [ ] Oodle
- [ ] Hoobly
- [ ] Geebo
- [ ] Locanto
- [ ] FaceBook Marketplace

4) Which feature(s) are the most important to you in a classified advertising system? *

- [ ] Convenient and appropriate classified ad recommendation
- [ ] Price prediction and visualisation for an item prior for it to be advertised
- [ ] Fraud content identification and prevention from being submitted into the portal
- [ ] Fraud image detection and elimination to protect platform reliability and authenticity

5) Do you find the graphical user interface of Tievs attractive? *

- ( ) Yes
- ( ) No
- ( ) A little
- ( ) Not much

6) Do you think it is easy to navigate through the sections within the Tievs web application? *

- ( ) Yes
- ( ) No
- ( ) Maybe
- ( ) I'm not sure

7) Did you encounter any inconveniences while browsing or utilising the functionalities provided by Tievs? *

- ( ) Yes
- ( ) No
- ( ) Maybe

8) If the answer you gave is "Yes" for the previous question (7), please state the nature of the inconvenience which you faced. (Else, you can ignore this question)

Your answer

9) How do you assess the effectiveness of the recommendation system in Tievs? *

- ◯ Very Good
- ◯ Good
- ◯ Poor

10) How do you assess the effectiveness of the car price prediction and visualisation system in Tievs? *

- ◯ Very Good
- ◯ Good
- ◯ Poor

11) How do you assess the effectiveness of the fraud content identification and prevention system in Tievs? *

- ◯ Very Good
- ◯ Good
- ◯ Poor

12) How do you assess the effectiveness of the erroneous image identification and prevention system in Tievs? *

- ◯ Very Good
- ◯ Good
- ◯ Poor

13) Did you discover any major faults in any of the above main component functionalities? *

- ◯ Yes
- ◯ No

14) If the answer you gave is "Yes" for the previous question (14), please choose the component(s) where the fault(s) was discovered.

- ☐ Recommendation System
- ☐ Price Prediction System
- ☐ Fraud Content Identification System
- ☐ Erroneous Image Identification System

15) If the answer you gave is "Yes" for the previous question (14), please state the fault(s) encountered.

Your answer

16) Please provide your overall system review for our Tievs classified advertising system. *

- Excellent
- Very Good
- Good
- Poor
- Very poor

17) Would you give your preference to recommend Tievs for others in future? *

- Yes
- No

18) If you may, you are free to provide some feedback regarding future refinements for Tievs.

Your answer

Thank you for your valuable contribution. Have a nice day!

Submit

Clear form

Appendix B: Plagiarism Report from Turnitin

Group Final Report

ORIGINALITY REPORT

17%	3%	2%	16%
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Sri Lanka Institute of Information Technology Student Paper	15%
2	A Kuznetsov. "Digital image forgery detection using deep learning approach", Journal of Physics: Conference Series, 2019 Publication	<1%
3	dl.lib.mrt.ac.lk Internet Source	<1%
4	machinelearningmastery.com Internet Source	<1%
5	www.dselva.co.in Internet Source	<1%
6	doctorpenguin.com Internet Source	<1%
7	www.diva-portal.org Internet Source	<1%
8	www.pewresearch.org Internet Source	<1%

9	Submitted to University of Wales, Bangor Student Paper	<1 %
10	Submitted to University of Westminster Student Paper	<1 %
11	acp.copernicus.org Internet Source	<1 %
12	gigadom.in Internet Source	<1 %
13	Submitted to University of Edinburgh Student Paper	<1 %
14	docplayer.es Internet Source	<1 %
15	ecommons.usask.ca Internet Source	<1 %
16	fs.ogm.utah.gov Internet Source	<1 %
17	open.metu.edu.tr Internet Source	<1 %
18	Forrest Jehlik, Alvaro Blazquez de Mingo. "A Deep Learning Approach to Modeling a Complex Multi-variate, Temporal Thermal Problem", 2019 18th IEEE International Conference On Machine Learning And Applications (ICMLA), 2019 Publication	<1 %

19	www.mdpi.com Internet Source	<1 %
20	"Advances in Information and Communication Networks", Springer Science and Business Media LLC, 2019 Publication	<1 %
21	"Deep Learning in Natural Language Processing", Springer Science and Business Media LLC, 2018 Publication	<1 %
22	dokumen.pub Internet Source	<1 %
23	etd.astu.edu.et Internet Source	<1 %
24	"Proceedings of International Conference on Recent Trends in Machine Learning, IoT, Smart Cities and Applications", Springer Science and Business Media LLC, 2021 Publication	<1 %
25	"Ubiquitous Intelligent Systems", Springer Science and Business Media LLC, 2022 Publication	<1 %