

**ACKNOWLEDGEMENT**

I would like to place on record my deep sense of gratitude to **Mr. Sanjeev Patel**, Designation, Jaypee Institute of Information Technology, India for his/her generous guidance, help and useful suggestions.

I express my sincere gratitude to \_\_\_\_\_\_\_\_\_\_\_\_\_\_, Dept. of \_\_\_\_\_\_\_\_\_\_\_\_, India, for his/her stimulating guidance, continuous encouragement and supervision throughout the course of present work.

I also wish to extend my thanks to \_\_\_\_\_\_\_\_\_\_\_\_\_ and other classmates for their insightful comments and constructive suggestions to improve the quality of this project work.

**Signature(s) of Students**

Apurva Agarwal (9915103007)

Gaurav Singh (9915103252)

Kshitij Jain (9915103122)

**ABSTRACT**

Road traffic congestion is one of the problems in large cities. While Twitter has become a major means for communication and message dissemination among Internet users, one question that arises is whether Twitter messages alone can suggest road traffic congestion condition. We propose a method to predict traffic congestion severity level based on the analysis of Twitter messages. Different types of tweets data are used to construct a C4.5 decision tree model for prediction, including tweets from selected road-traffic Twitter accounts, tweets that contain road-traffic-related keywords, and geo-tagged tweets whose volume suggests large crowds in certain areas. In the training that used tweets data of one of the top congested roads in Bangkok, density of tweets has high information gain in classifying congestion severity level. Via a web application, the model can provide a traveller with an estimate of the congestion severity level of the road at every 30 minutes. Such advance congestion information can help the traveller to make a better travel plan. In an evaluation using 10-fold cross validation, the model also shows good performance.

Social network services collaborate millions of people together. These social networks outsource their data via APIs.

Twitter analysis on Road traffic congestion deals with live analysis of twitter data based on keywords related to traffic and current events. We are interested in using this data as a Social tool to predict the current condition of traffic and curb major delays and organize people’s commute. The better the predictions are, more severe our estimations become.

**TABLE OF CONTENTS**

**Page No.**

Acknowledgement i.

Abstract ii.

Gantt Chart iii.

List of Tables iv.

List of Figures iv.

List of Abbreviation v.

Nomenclature vi.

**Chapter 1). Introduction** 1.

**Chapter 2). Background Study** 2.

2.1. Panda Library 2.

2.2 Randam Library 3.

2.3. Tweepy Library 4.

2.4 Pymongo Library 4.

2.5 MondoDb 5.

2.6 Weka 5.

2.7 Decision Tree 6.

**Chapter 3). Requirement Analysis** 8.

3.1. Software 8.

3.2. Hardware 8.

3.5 UML diagram 12.

**Chapter 4). Implementation** 18.

**Chapter 5). Testing Report** 19.

6.1 Unit Testing 19.

6.2 Integration Testing 20.

6.3 Performance Testing 21.

**Chapter 7). Conclusion and Future Scope** 22.

**Chapter 8). References** 23.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Requirement Gathering** |  |  |  |  |  |
| **Design and Analysis** |  |  |  |  |  |
| **Frontend Coding** |  |  |  |  |  |
| **Backend Coding** |  |  |  |  |  |
| **Testing** |  |  |  |  |  |
| **Validation and Implementation** |  |  |  |  |  |
| **Documentation** |  |  |  |  |  |
| **Time Period** | **August** | **September** | **September** | **October** | **November** |

**Gantt Chart**

**List of Tables**

**Table Title Page No.**

3.1 Functions of OCR/ICR.............................................................................. 3

3.2 Functions of OCR/ICR............................................................................. 5

4.2.1 Innovative Ideas of Project........................................................................ 16

6.1.1 Unit Testing............................................................................................... 19

6.2.1 Integration Testing..................................................................................... 20

6.3.1 Performance Testing.................................................................................. 21

**LIST OF FIGURES**

**Figure Title Page No.**

i. Gantt Chart.................................................................................. iii.

3.f.1 User-Case Diagram………………....………............................ 12.

3.f.2 Sequence Diagram……………………....….............................. 13.

3.f.3 Class Diagram……………………………………………......... 14.

**List of Abbreviations**

* **JSON** Java Script Object Notation
* **WEKA** Waikato Environment for Knowledge Analysis
* **OAUTH** Open Authorization
* **API** Application programming interface

**Nomenclature**

|  |  |
| --- | --- |
| **API** | a set of functions and procedures that allow the creation of applications which access the features or data of an operating system, application, or other service. |
| **WEKA** | Waikato Environment for Knowledge Analysis is a suite of machine learning software written in Java, developed at the University of Waikato, New Zealand. It is free software licensed under the GNU General Public License. |
| **TWYTHON** | Actively maintained, pure Python wrapper for the Twitter API. Supports both normal and streaming Twitter API. |
| **DECISION**  **TREE** | A decision tree is a decision support tool that uses a tree-like graph or model of decisions and their possible consequences, including chance event outcomes, resource costs, and utility. |
| **MongoDB** | MongoDB is a free and open-source cross-platform document-oriented database program. Classified as a NoSQL database program, MongoDB uses JSON-like documents with schemas. |
| **OAuth** | OAuth is an authentication method where you approve  applications to act on your behalf without sharing your password. |
| **PYTHON** | Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991. |

**1. INTRODUCTION**

**1.1 DOCUMENT PURPOSE**

This paper has shown that tweets alone can be used to estimate road traffic congestion condition as tweets density and hours of day are useful attributes for building a congestion severity prediction model. In an evaluation, the model shows good performance, and the web application can help with planning a travel by estimating congestion condition 30 minutes in advance. We plan to improve the model by trying to extract more information that can indicate road links from nongeo-tagged tweets and see if that can help better balance the dataset. Construction of a more general model, using more road data, should be further explored. In addition, the traffic condition-related keywords can be refined to better collect tweets that are related to traffic congestion.

**2. BACKGROUND STUDY**

Before starting our project, we referred some research papers and similar existing apps and technologies which gave us fair knowledge and idea and made us clear about the work which we would do try to complete with the best possible outputs.

**2.1) Panda Library**

In computer programming, pandas is a software library written for the Python programming language for data manipulation and analysis. In particular, it offers data structures and operations for manipulating numerical tables and time series. It is free software released under the three-clause BSD license. [2] The name is derived from the term "panel data", an econometrics term for multidimensional, structured data sets.

**Library Features**

Data Frame object for data manipulation with integrated indexing.

Tools for reading and writing data between in-memory data structures and different file formats.

Data alignment and integrated handling of missing data.

Reshaping and pivoting of data sets.

Label-based slicing, fancy indexing, and sub setting of large data sets.

Data structure column insertion and deletion.

Group by engine allowing split-apply-combine operations on data sets.

Data set merging and joining.

Hierarchical axis indexing to work with high-dimensional data in a lower-dimensional data structure.

Time series-functionality: Date range generation and frequency conversion, moving window statistics, moving window linear regressions, date shifting and lagging.

**2.2) Random Library**

This module implements pseudo-random number generators for various distributions.

For integers, there is uniform selection from a range. For sequences, there is uniform selection of a random element, a function to generate a random permutation of a list in-place, and a function for random sampling without replacement.

On the real line, there are functions to compute uniform, normal (Gaussian), lognormal, negative exponential, gamma, and beta distributions. For generating distributions of angles, the von Mises distribution is available.

Almost all module functions depend on the basic function random(), which generates a random float uniformly in the semi-open range [0.0, 1.0). Python uses the Mersenne Twister as the core generator. It produces 53-bit precision floats and has a period of 2\*\*19937-1. The underlying implementation in C is both fast and thread safe. The Mersenne Twister is one of the most extensively tested random number generators in existence. However, being completely deterministic, it is not suitable for all purposes, and is completely unsuitable for cryptographic purposes.

The functions supplied by this module are actually bound methods of a hidden instance of the random.Random class. You can instantiate your own instances of Random to get generators that don’t share state.

Class Random can also be subclassed if you want to use a different basic generator of your own devising: in that case, override the random(), seed(), getstate(), and setstate() methods. Optionally, a new generator can supply a getrandbits() method — this allows randrange() to produce selections over an arbitrarily large range.

The random module also provides the SystemRandom class which uses the system function os.urandom() to generate random numbers from sources provided by the operating system.

**2.3) Tweepy Library**

Tweepy supports oauth authentication. Authentication is handled by the tweepy.AuthHandler class.

OAuth Authentication

Tweepy tries to make OAuth as painless as possible for you. To begin the process, we need to register our client application with Twitter. Create a new application and once you are done you should have your consumer token and secret. Keep these two handy, you’ll need them.

The next step is creating an OAuthHandler instance. Into this we pass our consumer token and secret which was given to us in the previous paragraph:

auth = tweepy.OAuthHandler(consumer\_token, consumer\_secret)

If you have a web application and are using a callback URL that needs to be supplied dynamically you would pass it in like so:

auth = tweepy.OAuthHandler(consumer\_token, consumer\_secret,

callback\_url)

If the callback URL will not be changing, it is best to just configure it statically on twitter.com when setting up your application’s profile.

Unlike basic auth, we must do the OAuth “dance” before we can start using the API. We must complete the following steps:

1.Get a request token from twitter

2.Redirect user to twitter.com to authorize our application

3.If using a callback, twitter will redirect the user to us. Otherwise the user must manually supply us with the verifier code.

4.Exchange the authorized request token for an access token.

**2.4) PyMongo Library**

**PyMongo is a Python distribution containing tools for working with MongoDB, and is the recommended way to work with MongoDB from Python. This documentation attempts to explain everything you need to know to use PyMongo.**

**2.5) Mongo DB**

MongoDB **stores data in flexible, JSON-like documents**, meaning fields can vary from document to document and data structure can be changed over time

The document model **maps to the objects in your application code**, making data easy to work with

**Ad hoc queries, indexing, and real time aggregation** provide powerful ways to access and analyze your data

MongoDB is a **distributed database at its core**, so high availability, horizontal scaling, and geographic distribution are built in and easy to use

MongoDB is **free and open-source**, published under the GNU Affero General Public License.

**2.6) WEKA**

Weka (pronounced to rhyme with Mecca) contains a collection of visualization tools and algorithms for data analysis and predictive modelling, together with graphical user interfaces for easy access to these functions.[1] The original non-Java version of Weka was a Tcl/Tk front-end to (mostly third-party) modelling algorithms implemented in other programming languages, plus data pre-processing utilities in C, and a Makefile-based system for running machine learning experiments. This original version was primarily designed as a tool for analysing data from agricultural domains, but the more recent fully Java-based version (Weka 3), for which development started in 1997, is now used in many different application areas, in particular for educational purposes and research. Advantages of Weka include:

• Free availability under the GNU General Public License.

• Portability, since it is fully implemented in the Java programming language and thus runs on almost any modern computing platform.

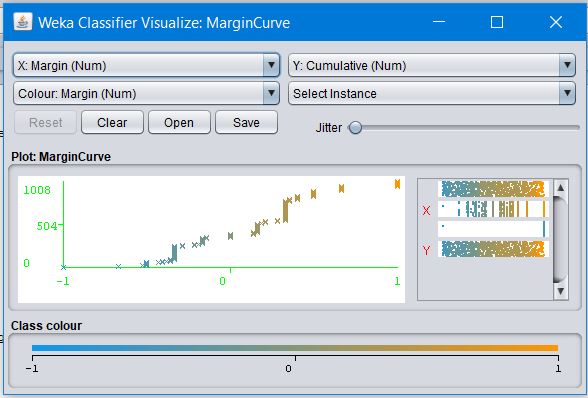
• A comprehensive collection of data preprocessing and modeling techniques.

Ease of use due to its graphical user interfaces.Weka supports several standard data mining tasks, more specifically, data pre-processing, clustering, classification, regression, visualization, and feature selection. All of Weka's techniques are predicated on the assumption that the data is available as one flat file or relation, where each data point is described by a fixed number of attributes (normally, numeric or nominal attributes, but some other attribute types are also supported). Weka provides access to SQL databases using Connectivity and can process the result returned by a database query. Weka provides access to deep learning with Deeplearning4j.[4] It is not capable of multi-relational data mining, but there is separate software for converting a collection of linked database tables into a single table that is suitable for processing using Weka.[5] Another important area that is currently not covered by the algorithms included in the Weka distribution is sequence modelling.

**2.7) Decision Tree**

**Decision Trees (DTs)** are a non-parametric supervised learning method used for classification and regression. The goal is to create a model that predicts the value of a target variable by learning simple decision rules inferred from the data features.

For instance, in the example below, decision trees learn from data to approximate a sine curve with a set of if-then-else decision rules. The deeper the tree, the more complex the decision rules and the fitter the model.



Some advantages of decision trees are:

* Simple to understand and to interpret. Trees can be visualised.
* Requires little data preparation. Other techniques often require data normalisation, dummy variables need to be created and blank values to be removed. Note however that this module does not support missing values.

•The cost of using the tree (i.e., predicting data) is logarithmic in the number of data points used to train the tree.

•Able to handle both numerical and categorical data. Other techniques are usually specialised in analysing datasets that have only one type of variable. See algorithms for more information.

•Able to handle multi-output problems.

•Uses a white box model. If a given situation is observable in a model, the explanation for the condition is easily explained by boolean logic. By contrast, in a black box model (e.g., in an artificial neural network), results may be more difficult to interpret.

•Possible to validate a model using statistical tests. That makes it possible to account for the reliability of the model.

•Performs well even if its assumptions are somewhat violated by the true model from which the data were generated.The disadvantages of decision trees include

•Decision-tree learners can create over-complex trees that do not generalise the data well. This is called overfitting. Mechanisms such as pruning (not currently supported), setting the minimum number of samples required at a leaf node or setting the maximum depth of the tree are necessary to avoid this problem.

•Decision trees can be unstable because small variations in the data might result in a completely different tree being generated. This problem is mitigated by using decision trees within an ensemble.

•The problem of learning an optimal decision tree is known to be NP-complete under several aspects of optimality and even for simple concepts. Consequently, practical decision-tree learning algorithms are based on heuristic algorithms such as the greedy algorithm where locally optimal decisions are made at each node. Such algorithms cannot guarantee to return the globally optimal decision tree. This can be mitigated by training multiple trees in an ensemble learner, where the features and samples are randomly sampled with replacement.

•There are concepts that are hard to learn because decision trees do not express them easily, such as XOR, parity or multiplexer problems.

•Decision tree learners create biased trees if some classes dominate. It is therefore recommended to balance the dataset prior to fitting with the decision tree.

**3. REQUIREMENT ANALYSIS**

**3.1) SOFTWARE REQUIREMENTS**

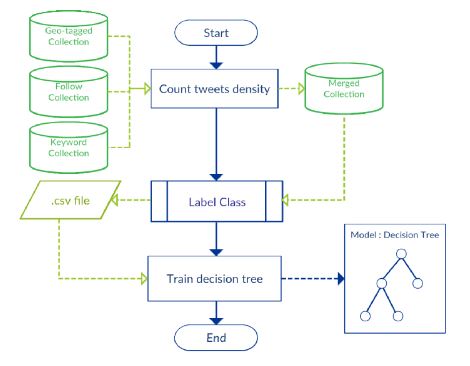
* + **Software:** Python, MongoDB, Weka-Wrapper Library, Twitter API, Twython library.
  + **Python**: Programming language that let you work quickly and integrate systems more efficiently.
  + **MongoDB**: It is an enterprise-wide data management solution. Designed for Big.
  + **Weka-Wrapper Library**: It makes easy to run Weka algorithms and filters from within Python. It offers access to Weka API using thin wrappers around JNI calls using Java bridge package.
  + **Twitter API**: Python package for accessing twitter’s Rest API and streaming API. Supports OAuth 1.0, OAuth 2.0 authentication and it works with latest python 2.x and 3.x branches.
    - **Twython**: Python wrapper of the Twitter API 1.1. It provides an easy way to access twitter data.

**3.2) HARDWARE REQUIREMENTS**

* Minimum Requirement is as follows:
* 4 GB RAM required
* Processor with speed 2 GHz
* Internet Connectivity: The system must run over the internet; all the hardware shall require to connect internet will be hardware internet for the interface.
* An online web server to host the OCR.

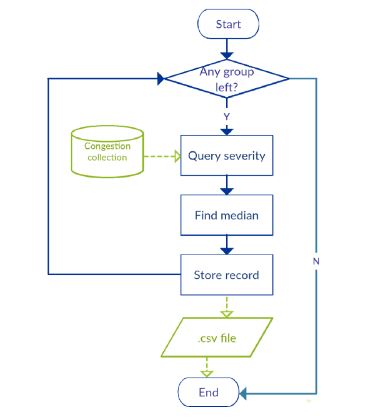
**3.3) UML Diagram**

**3.3.1. User-Case Diagram:**

****

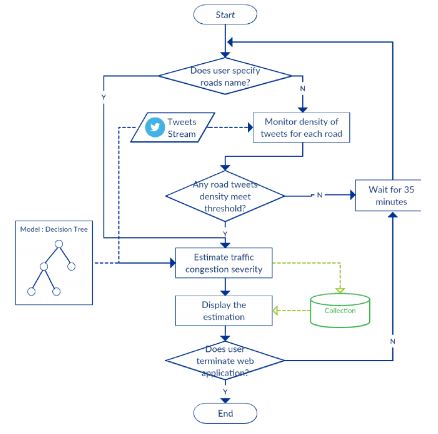
**Fig 3.f.1:** User-Case Diagram

**3.3.2. Sequence Diagram**

****

**Fig 3.f.2:** Sequence Diagram

**3.3.3. Class Diagram**

****

**Fig 3.f.3:** Class Diagram

**5. IMPLEMENTATION**

**A. Data Collection**

We collected data from three sources, i.e. road data, Twitter data, and actual road traffic congestion data. They were processed and stored in a MongoDB.

**1) Road Data**

First, we collected road data from Geofabric. Then, we used QGIS application to select only primary, secondary, and motorway types of roads in Bangkok, using the spatial query tool of QGIS. We exported the selected data to the GeoJSON format and then stored them in a MongoDB collection. Finally, we created 2dsphere index in the geometry field of the collection. This collection is called the Road collection.

**2) Twitter Data**

We wrote three python scripts to collect tweets via Twitter Streaming APIs. We collected data for 30 days. and then processed and stored them in MongoDB collection.

|  |  |
| --- | --- |
| **Types of tweets** | **MongoDB collection Name** |
| Geo-tagged | Geo-tagged collection |
| From the selected accounts | Follow collection |
| keywords related | Keyword collection |

**3) Actual Road Traffic Congestion Data**

We got the actual traffic congestion severity data from iTIC and Longdo Traffic. Congestion severity is categorized into three levels, i.e. L (low), M (medium), and H (high). Note that the actual road data are finer-grained than the road data in our Road collection because the actual congestion data were collected for “links” of roads, whereas the road information in our Road collection were collected for “roads” since traffic information in tweets mostly were associated with roads, rather than links of roads. Also, the actual congestion data for the links were collected at every 30 minutes. We put these actual data in a MongoDB collection called the **Congestion collection.**

**B. Model Construction**

To construct the model, we prepared the training data from all the collected data in Section III.A. An example of the training data to predict congestion severity level (H, M, L) are

shown in Table I. There are four attributes, i.e. 1) day of week 2) hours of the day 3) minutes of the hour, and 4) tweets density. As described previously in Section III.A, we extracted the timestamp\_ms property of each tweet and put the tweets in groups by their timestamps. For example, the first group (or first record) of the training data in Table I refers to the tweets that occurred on Tuesday during 8:00-8:29 (or H8:M0) whereas the third group refers to the tweets that occur on Tuesday during 2:30-2:59 (or H2:M30). Note that “H” was appended to hours of day and “M” to minutes of hour to make them nominal values. The tweets in all three collections (i.e. Geo-tagged, Follow, and Keyword collections) which belonged to each group then were counted, as shown in Fig. 3, to obtain tweets density of the group. To do so, we assigned different weights for the tweets from different collections based on how strong they represent traffic information. That is, tweets from the selected Twitter accounts concern actual traffic condition, whereas the geo-tagged tweets are merely tweets with geo-tags and may not be about traffic condition. Additionally, tweets with traffic-related keywords may or may not give information about congestion condition. We hence assigned the maximum weight of 10 to each tweet in the Follow collection, medium weight of 4 to each tweet in the Keyword collection, and minimum weight of 1 to each tweet in the Geo-tagged collection. Then, we counted tweets density of each group by calculating the total weights of the tweets in the group. For example, if there was one tweet from each of the three collections which belong to the same group Mon H8:M0, tweets density for this group would be 15. We selected a road with the highest tweets density for training, i.e. Pahonyothin Road, in this case. All tweets in each group were merged and stored in the Merged collection. Next, to label a class of congestion severity (H, M, L) to a group of tweets, the process is shown in Fig. 4. For each group of tweets, we queried congestion severity of Pahonyothin Road from the Congestion collection using day of week, hours of day, and minutes of hour of the next 30 minutes after the timestamp of the group, as a query condition. For example, for the group Tue H8:M0 (8:00-8:29), we queried the congestion severity of Tue during 8:30-8:59. Since Congestion collection stored congestion severity level of each link of the road, so the query returned several congestion severity levels for different links of the road during 8:30-8:59. Therefore we had to find a representative value to represent the congestion severity level of the whole road. As the severity level is in a nominal scale, we chose the median value. After that, each group with a labeled class was stored in the .csv file of the training data.

**C. Use of Model**

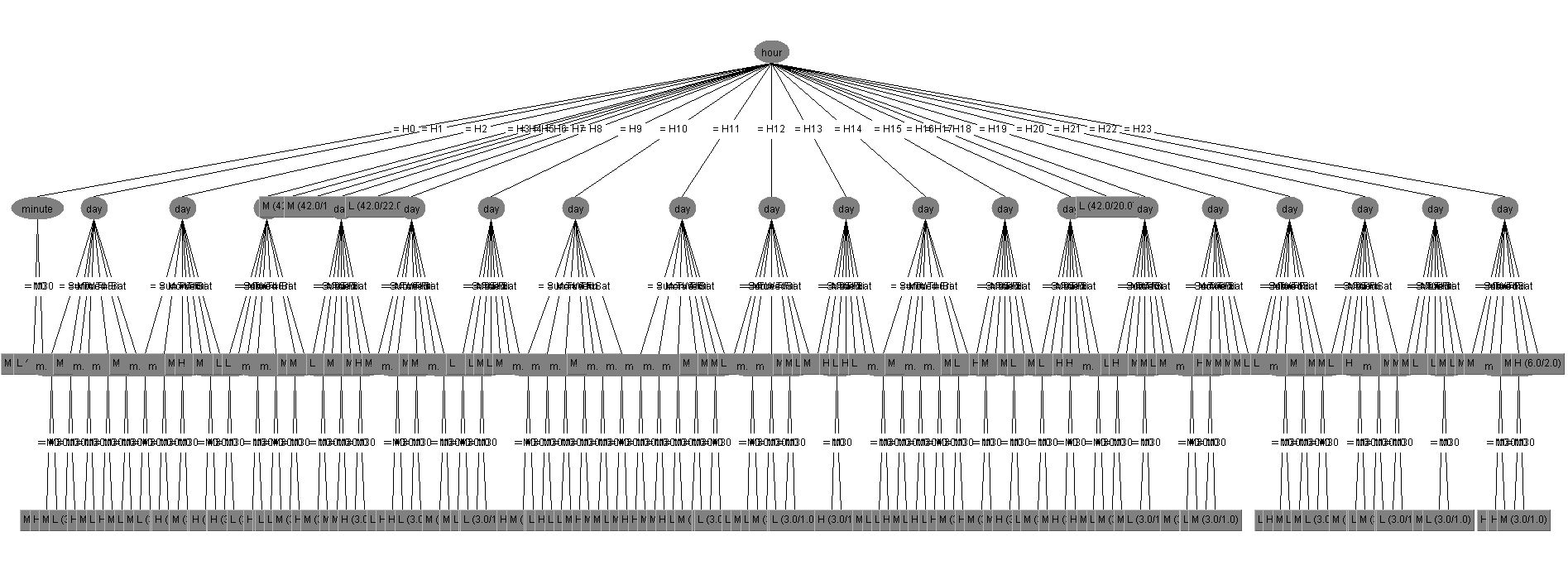
To use the model, we developed a web application that can estimate congestion severity level of the road. The Django web framework [13] was used with the Python-Weka wrapper

library [14] to deal with the model file from the previous step. Fig. 5 describes the process of the web application which comprises two parts, i.e. backend and frontend. • For the backend design, we create scheduled tasks to collect tweets from selected accounts, tweets containing traffic-related keywords, and geo-tagged tweets, and process them in the same way as the training data. While the scheduled tasks work, we inspect tweets density for each road. If any road has its tweets density equal to 10, we will test its attributes with our model and then store the predicted result in a MongoDB collection for the frontend to display. (Note that this tweets density threshold is defined to reduce tweets processing.) We use the Python-Weka-wrapper library to call necessary Weka methods for prediction. The prediction will be executed every 35 minutes because the model can predict the congestion severity in the next 30 minutes, plus 5 minutes for checking tweets density with the threshold.

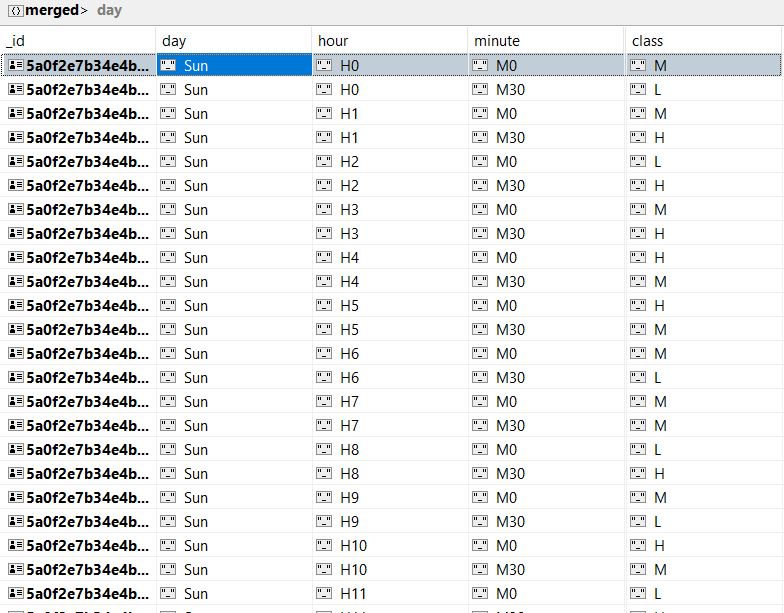
• For the front end design, we offer an option for a user to specify a road name. If a road name is specified, the web application will display only the predicted congestion severity for that road. But if the user does not specify a road name, the application will display congestion severity of all roads whose tweets density values meet the defined threshold described earlier in the backend design part. Also, the web application will be refreshed every 35 minutes if the user does not terminate the application. Fig. 6 shows the UI of the web application. Roads data and their congestion severity estimation results are displayed on map controls whose base map is Google Maps, in the Map section. Different colors represent different severity estimation results, i.e. red for “H”, yellow for “M” and green for “L”. Figure 5. Use of model in web application. Figure 6. User interface of web application. In addition, a user can specify a road name in the Condition section, and the road name will be added to the Watch List section. (The user can also remove it from the watch list.) The Time Information section displays the current time and tells the user which time duration the congestion severity estimation is for. The Result section displays the estimation result.

**6. TESTING REPORT**

Since we aim to study the usefulness of Twitter messages to congestion severity prediction, in the experiment we employed several attributes sets to the C4.5 decision tree in order to test which set could give tweets density a high information gain value. The result was the attributes set shown earlier in Table I. The model from the training.



Several issues were investigated during the construction of the model. In Section III.B, we used the weights of 10, 4, and 1 for the tweets in the Follow, Keyword, and Geo-tagged collections. In fact, we also experimented with other sets of weight values but selected the aforementioned since the resulting model could predict all severity classes H, M, and L. Apart from the C 4.5. decision tree, we also tried other prediction models such as decision table and Naïve Bayes, but the resulting models could not predict the severity class M. As we trained the model using one road and the training dataset contained the imbalanced class M, we further experimented if using more data of other road could help make the dataset more balanced and more attributes become useful to the model. First, we tried to add the data of another road (i.e. Ratchadapisek Road), processed them to obtain a .csv file, and merged this file with that of the Pahonyothin Road to test if the training result could be generic for these two roads. We found that the model has much lower performance than the model based on one road. In addition, we tested with the training data of the Ratchadapisek Road only, and found that using the median to represent the severity class of the whole road made the class L dominate the others. From these experiments, it could be that each road might have its own characteristics, and hence to estimate congestion severity of other roads, we should find a suitable attributes set and method to select the representative severity class for the whole road. Another limitation of using Twitter is that we generally cannot extract fine-grained information, i.e. which link of the road a particular tweet is associated with, except for a geo-tagged tweet. But a geotagged tweet itself may not have congestion-related content. Also, it is found that not a great number of tweets from the selected traffic accounts indicate the links of the roads. Therefore, the model and the web application in the current form can only estimate congestion severity of the whole roads.



**7. CONCLUSION & FUTURE SCOPE**

**8. REFERENCES**

1. 7. Twitter Inc., The Streaming APIs,

https://developer.twitter.com/

2. L. Burks, M. Miller, and R. Zadeh,

“Rapid estimate of ground shaking intensity by combining simple earthquake characteristics with tweets,”

3. F. Lécué, R. Tucker, V. Bicer, P. Tommasi, S. Tallevi-Diotallevi, and M. Sbodio

“Predicting severity of road traffic congestion using

semantic web technologies,”

4. N. Wanichayapong, W. Pruthipunyaskul, W. Pattara-Atikom, and P Chaovalit

“Social-based traffic information extraction and classification”

5. Geofabrik GmbH, OpenStreetMap Contributors, Thailand OpenStreetMap data,

http://www.geofabrik.de/

6. MongoDB, Inc., MongoDB,

https://www.mongodb.com/

8. Weka 3

http://www.cs.waikato.ac.nz/ml/weka/