Chapter 1: INTRODUCTION

1.1 Overview of Project

Various Business sectors and government agencies and individual travellers require precise and appropriately traffic flow information. It helps the riders and drivers to make better travel judgement to alleviate traffic congestion, improve traffic operation efficiency, and reduce carbon emissions. The development and deployment of Intelligent Transportation System (ITSs) provide better accuracy for Traffic flow prediction. It is deal with as a crucial element for the success of advanced traffic management systems, advanced public transportation systems, and traveller information systems. [1]. The dependency of traffic flow is dependent on real-time traffic and historical data collected from various sensor sources, including inductive loops, radars, cameras, mobile Global Positioning System, crowdsourcing, social media. Traffic data is exploding due to the vast use of traditional sensors and new technologies, and we have entered the era of a large volume of data transportation. Transportation control and management are now becoming more data-driven. [2], [3].However, there are already lots of traffic flow prediction systems and models; most of them use shallow traffic models and are still somewhat failing due to the enormous dataset dimension.

Recently, deep learning concepts attract many persons involving academicians and industrialist due to their ability to deal with classification problems, understanding of natural language, dimensionality reduction, detection of objects, motion modelling. DL uses multi-layer concepts of neural networks to mining the inherent properties in data from the lowest level to the highest level [4]. They can identify massive volumes of structure in the data, which eventually helps us to visualize and make meaningful inferences from the data. Most of the ITS departments and researches in this area are also concerned about developing an autonomous vehicle, which can make transportation systems much economical and reduce the risk of lives. Also, saving time is the integrative benefit of this idea. In current decades the lots of attention have made towards the safe automatic driving. It is necessary that the information will be provided in time through driver assistance system (DAS), autonomous vehicles (AV)and Traffic Sign Recognition (TSR) [5].

Chapter 2: LITERATURE SURVEY

1) Accelerated Incident Detection across Transportation Networks using Vehicle Kinetics and Support Vector Machine in Cooperation with Infrastructure Agents

AUTHORS: Ma, Yongchang,Chowdhury, Mashrur,Jeihani, Mansoureh,Fries, Ryan This study presents a framework for highway incident detection using vehicle kinetics, such as speed profile and lane changing behavior. This approach was envisioned in the vehicle-infrastructure integration (VII, also known as IntelliDrive) model in which vehicles and infrastructure communicate with each other to improve mobility and safety. The framework uses an in-vehicle intelligent module, based on a support vector machine (SVM), to determine the vehicle's travel experiences with autonomously generated kinetics data. Roadside infrastructure agents (also known as RSUs: roadside units) detect the incident by compiling travel experiences from several vehicles and comparing the aggregated results with the pre-selected threshold values. The authors report on their evaluations of this system in two different environments: a previously calibrated and validated simulation network in rural Spartanburg, South Carolina; and an urban freeway network in Baltimore, Maryland. Their evaluation found no significant differences in the detection performance between the original network and a new network that the VII-SVM system has not seen before. They

conclude that the generic VII-SVM system is valuable and applicable across transportation networks.

2)A Decentralized Approach for Anticipatory Vehicle Routing Using Delegate Multiagent Systems

AUTHORS: Qin Yu, Tao Jiang, Aiyun Zhou, Lili Zhang, Cheng Zhang & Pan Xu

Advanced vehicle guidance systems use real-time traffic information to route traffic and to avoid congestion. Unfor- tunately, these systems can only react upon the presence of traffic jams and not to prevent the creation of unnecessary congestion. Anticipatory vehicle routing is promising in that respect, because this approach allows directing vehicle routing by accounting for traffic forecast information. This paper presents a decentralized approach for anticipatory vehicle routing that is particularly use- ful in large-scale dynamic environments. The approach is based on delegate multiagent systems, i.e., an environment-centric co- ordination mechanism that is, in part, inspired by ant behavior. Antlike agents explore the environment on behalf of vehicles and detect a congestion forecast, allowing vehicles to reroute. The approach is explained in depth and is evaluated by comparison with three alternative routing

strategies. The experiments are done in simulation of a real-world traffic environment. The experiments indicate a considerable performance gain compared with the most advanced strategy under test, i.e., a traffic-message-channel-based routing strategy.

3)Dedicated Short-Range Communications Technology for Freeway Incident Detection: Performance Assessment Based on Traffic Simulation Data

AUTHORS: Xuehu Wang ,Yongchang Zheng ,Lan Gan,Xuan Wang,Xinting Sang,Xiangfeng Kong,Jie Zhao

An assessment of the use of dedicated short-range communications technology to perform travel time monitoring and automated incident detection on a segment of rural freeway is described. The assessment used the CorSim traffic simulation tool to simulate traffic and incidents on a segment of rural freeway. Output data from the simulation was subjected to postprocessing to convert it to probe and beacon data. An incident detection algorithm was developed by using a travel time threshold and a counter. An alarm was generated when a counter reached a preselected level. This algorithm was tested on selected data files, and the results were used to identify the optimum values of the travel time threshold and the counter alarm level. With these optimum values, the algorithm was applied to the probe and beacon data to determine how quickly the system could detect various traffic incidents. The analysis showed that the system could provide rapid and reliable detection of incidents. During the simulation and analysis, several parameters were varied so that their effect on the system performance could be observed. Each parameter proved to have a significant effect on the detection time, and the observed effects were consistent with logical expectations. In general, the time to detect an incident was reduced in response to (a) increased traffic volume, (b) increased incident severity, (c) increased transponder population, (d) reduced reader spacing, and (e) reduced distance from incident to next downstream reader.

4)FREEWAY INCIDENT DETECTION USING KINEMATIC DATA FROM PROBE VEHICLES

AUTHORS:Qin Yu, Tao Jiang, Aiyun Zhou, Lili Zhang, Cheng Zhang & Pan Xu

This paper presents an incident detection algorithm based on the speed and acceleration profiles of probe vehicles as they travel along a freeway. It is based on the assumption that when a probe vehicle approaches a detectable incident, it will decelerate from its normal speed and then accelerate back to the normal speed after passing the incident. The incident detection performance of the algorithm, at various percentages of probe vehicles in the traffic stream, has been tested on a set of incident data generated by a calibrated microscopic traffic simulation model. The results are compared with a multi-layer feed-forward neural network incident detection techniques that uses volume, speed and occupancy measured at fixed locations as inputs. It is found that when there are 30% probe vehicles in the traffic stream, the new probe vehicle algorithm can achieve comparable detection rate and mean time to detection against the neural network model.

5)LSTM network: a deep learning approach for short-term traffic forecast

AUTHORS:Joanna Jaworek-Korjakowska 1 and Paweł Kłeczek 1.

Short-term traffic forecast is one of the essential issues in intelligent transportation system. Accurate forecast result enables commuters make appropriate

travel modes, travel routes, and departure time, which is meaningful in traffic management. To promote the forecast accuracy, a feasible way is to develop a more effective approach for traffic data analysis. The availability of abundant traffic data and computation power emerge in recent years, which motivates us to improve the accuracy of short-term traffic forecast via deep learning approaches. A novel traffic forecast model based on long short-term memory (LSTM) network is proposed. Different from conventional forecast models, the proposed LSTM network considers temporal-spatial correlation in traffic system via a two-dimensional network which is composed of many memory units. A comparison with other representative forecast models validates that the proposed LSTM network can achieve a better performance

2.2 Present System

A method of detection of specific disease in grapes is proposed in. Downy mildew pest/disease can be detected by the real time system with weather data. The central sever provide forecast service of weather condition and disease. Another kind of solution related of monitoring traps which are used to capture pest is with the help of image sensors. In, he authors designed and implemented a low power consumed system which is based on wireless image sensors and powered by battery. The frequency of capturing and transferring trap images of sensors can be set and remote adjusted by trapping application.

2.3 Overview of Language (PYTHON)

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++or Java. It provides constructs that enable clear programming on both small

and large scales. Python interpreters are available for many operating systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation. Python features a dynamic type system and automatic memory management. It supports multiple programming paradigms, including object-oriented, imperative, functional and procedural, and has a large and comprehensive standard library.

Interactive Mode Programming

Invoking the interpreter without passing a script file as a parameter brings up the following prompt −

```
$ python
Python 2.4.3 (#1, Nov 11 2010, 13:34:43)
[GCC 4.1.2 20080704 (Red Hat 4.1.2-48)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Type the following text at the Python prompt and press the Enter −

```
>>> print "Hello, Python!"
```

If you are running new version of Python, then you would need to use print statement with parenthesis as in print ("Hello, Python!");. However in Python version 2.4.3, this produces the following result −

```
Hello, Python!
```

Script Mode Programming

Invoking the interpreter with a script parameter begins execution of the script and continues until the script is finished. When the script is finished, the interpreter is no longer active.

Let us write a simple Python program in a script. Python files have extension .py. Type the following source code in a test.py file −

Live Demo

```
print "Hello, Python!"
```

We assume that you have Python interpreter set in PATH variable. Now, try to run this program as follows −

$ python test.py
This produces the following result −

Hello, Python!
Let us try another way to execute a Python script. Here is the modified test.py file −

Live Demo
#!/usr/bin/python

print "Hello, Python!"
We assume that you have Python interpreter available in /usr/bin directory. Now, try to run this program as follows −

$ chmod +x test.py # This is to make file executable
$./test.py
This produces the following result −

Hello, Python!

Python Identifiers

A Python identifier is a name used to identify a variable, function, class, module or other object. An identifier starts with a letter A to Z or a to z or an underscore (_) followed by zero or more letters, underscores and digits (0 to 9).

Python does not allow punctuation characters such as @, $, and % within identifiers. Python is a case sensitive programming language. Thus, Manpower and manpower are two different identifiers in Python.

Here are naming conventions for Python identifiers −

Class names start with an uppercase letter. All other identifiers start with a lowercase letter.

Starting an identifier with a single leading underscore indicates that the identifier is private.

Starting an identifier with two leading underscores indicates a strongly private identifier.

If the identifier also ends with two trailing underscores, the identifier is a language-defined special name.

Reserved Words

The following list shows the Python keywords. These are reserved words and you cannot use them as constant or variable or any other identifier names. All the Python keywords contain lowercase letters only.

and exec not
assert finally or
break for pass
class from print
continue global raise
def if return
del import try
elif in while
else is with
except lambdayield

Lines and Indentation

Python provides no braces to indicate blocks of code for class and function definitions or flow control. Blocks of code are denoted by line indentation, which is rigidly enforced.

The number of spaces in the indentation is variable, but all statements within the block must be indented the same amount. For example −

```
if True:
   print "True"
else:
   print "False"
```

However, the following block generates an error −

```
if True:
print "Answer"
print "True"
else:
print "Answer"
print "False"
```

Thus, in Python all the continuous lines indented with same number of spaces would form a block. The following example has various statement blocks −

Note − Do not try to understand the logic at this point of time. Just make sure you understood various blocks even if they are without braces.

```
#!/usr/bin/python

import sys

try:
  # open file stream
  file = open(file_name, "w")
except IOError:
  print "There was an error writing to", file_name
  sys.exit()
print "Enter '", file_finish,
print "' When finished"
while file_text != file_finish:
  file_text = raw_input("Enter text: ")
  if file_text == file_finish:
    # close the file
    file.close
    break
  file.write(file_text)
  file.write("\n")
```

```
file.close()
file_name = raw_input("Enter filename: ")
if len(file_name) == 0:
  print "Next time please enter something"
  sys.exit()
try:
  file = open(file_name, "r")
except IOError:
  print "There was an error reading file"
  sys.exit()
file_text = file.read()
file.close()
print file_text
```

Multi-Line Statements

Statements in Python typically end with a new line. Python does, however, allow the use of the line continuation character (\) to denote that the line should continue. For example −

```
total = item_one + \
      item_two + \
      item_three
```

Statements contained within the [], {}, or () brackets do not need to use the line continuation character. For example −

```
days = ['Monday', 'Tuesday', 'Wednesday',
      'Thursday', 'Friday']
```

Quotation in Python

Python accepts single ('), double (") and triple (''' or """) quotes to denote string literals, as long as the same type of quote starts and ends the string.

The triple quotes are used to span the string across multiple lines. For example, all the following are legal −

```
word = 'word'
```

```
sentence = "This is a sentence."
paragraph = """This is a paragraph. It is
made up of multiple lines and sentences."""
```

Comments in Python

A hash sign (#) that is not inside a string literal begins a comment. All characters after the # and up to the end of the physical line are part of the comment and the Python interpreter ignores them.

Live Demo

```
#!/usr/bin/python

# First comment
print "Hello, Python!" # second comment
```

This produces the following result −

```
Hello, Python!
```

You can type a comment on the same line after a statement or expression −

```
name = "Madisetti" # This is again comment
```

You can comment multiple lines as follows −

```
# This is a comment.
# This is a comment, too.
# This is a comment, too.
# I said that already.
```

Following triple-quoted string is also ignored by Python interpreter and can be used as a multiline comments:

```
'''
This is a multiline
comment.
'''
```

Using Blank Lines

A line containing only whitespace, possibly with a comment, is known as a blank line and Python totally ignores it.

In an interactive interpreter session, you must enter an empty physical line to terminate a multiline statement.

Waiting for the User

The following line of the program displays the prompt, the statement saying “Press the enter key to exit”, and waits for the user to take action −

```
#!/usr/bin/python

raw_input("\n\nPress the enter key to exit.")
```

Here, "\n\n" is used to create two new lines before displaying the actual line. Once the user presses the key, the program ends. This is a nice trick to keep a console window open until the user is done with an application.

Multiple Statements on a Single Line

The semicolon (;) allows multiple statements on the single line given that neither statement starts a new code block. Here is a sample snip using the semicolon.

```
import sys; x = 'foo'; sys.stdout.write(x + '\n')
```

Multiple Statement Groups as Suites

A group of individual statements, which make a single code block are called suites in Python. Compound or complex statements, such as if, while, def, and class require a header line and a suite. Header lines begin the statement (with the keyword) and terminate with a colon (:) and are followed by one or more lines which make up the suite. For example −

```
if expression :
   suite
elif expression :
   suite
else :
   suite
```

Command Line Arguments

Many programs can be run to provide you with some basic information about how they should be run. Python enables you to do this with -h −

```
$ python -h
usage: python [option] ... [-c cmd | -m mod | file | -] [arg] ...
Options and arguments (and corresponding environment variables):
-c cmd : program passed in as string (terminates option list)
-d     : debug output from parser (also PYTHONDEBUG=x)
-E     : ignore environment variables (such as PYTHONPATH)
-h     : print this help message and exit
```

You can also program your script in such a way that it should accept various options. Command Line Arguments is an advanced topic and should be studied a bit later once you have gone through rest of the Python concepts.

Python Lists

The list is a most versatile datatype available in Python which can be written as a list of comma-separated values (items) between square brackets. Important thing about a list is that items in a list need not be of the same type.

Creating a list is as simple as putting different comma-separated values between square brackets. For example −

```
list1 = ['physics', 'chemistry', 1997, 2000];
list2 = [1, 2, 3, 4, 5 ];
list3 = ["a", "b", "c", "d"]
```

Similar to string indices, list indices start at 0, and lists can be sliced, concatenated and so on.

A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.

Creating a tuple is as simple as putting different comma-separated values. Optionally you can put these comma-separated values between parentheses also. For example −

```
tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5 );
tup3 = "a", "b", "c", "d";
```

The empty tuple is written as two parentheses containing nothing −

```
tup1 = ();
```

To write a tuple containing a single value you have to include a comma, even though there is only one value −

```
tup1 = (50,);
```

Like string indices, tuple indices start at 0, and they can be sliced, concatenated, and so on.

Accessing Values in Tuples

To access values in tuple, use the square brackets for slicing along with the index or indices to obtain value available at that index. For example −

Live Demo

```
#!/usr/bin/python

tup1 = ('physics', 'chemistry', 1997, 2000);
tup2 = (1, 2, 3, 4, 5, 6, 7 );
print "tup1[0]: ", tup1[0];
print "tup2[1:5]: ", tup2[1:5];
```

When the above code is executed, it produces the following result −

```
tup1[0]:  physics
tup2[1:5]:  [2, 3, 4, 5]
```

Updating Tuples

Accessing Values in Dictionary

To access dictionary elements, you can use the familiar square brackets along with the key to obtain its value. Following is a simple example −

Live Demo

```
#!/usr/bin/python
```

```
dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Name']: ", dict['Name']
print "dict['Age']: ", dict['Age']
```

When the above code is executed, it produces the following result −

```
dict['Name']:  Zara
dict['Age']:  7
```

If we attempt to access a data item with a key, which is not part of the dictionary, we get an error as follows −

Live Demo

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
print "dict['Alice']: ", dict['Alice']
```

When the above code is executed, it produces the following result −

```
dict['Alice']:
Traceback (most recent call last):
  File "test.py", line 4, in <module>
    print "dict['Alice']: ", dict['Alice'];
KeyError: 'Alice'
```

Updating Dictionary

You can update a dictionary by adding a new entry or a key-value pair, modifying an existing entry, or deleting an existing entry as shown below in the simple example −

Live Demo

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
dict['Age'] = 8; # update existing entry
dict['School'] = "DPS School"; # Add new entry
```

```
print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

When the above code is executed, it produces the following result −

```
dict['Age']:  8
dict['School']:  DPS School
```

Delete Dictionary Elements

You can either remove individual dictionary elements or clear the entire contents of a dictionary. You can also delete entire dictionary in a single operation.

To explicitly remove an entire dictionary, just use the del statement. Following is a simple example −

Live Demo

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Class': 'First'}
del dict['Name']; # remove entry with key 'Name'
dict.clear();     # remove all entries in dict
del dict ;        # delete entire dictionary

print "dict['Age']: ", dict['Age']
print "dict['School']: ", dict['School']
```

This produces the following result. Note that an exception is raised because after del dict dictionary does not exist any more −

```
dict['Age']:
Traceback (most recent call last):
  File "test.py", line 8, in <module>
    print "dict['Age']: ", dict['Age'];
TypeError: 'type' object is unsubscriptable
```

Note − del() method is discussed in subsequent section.

Properties of Dictionary Keys

Dictionary values have no restrictions. They can be any arbitrary Python object, either standard objects or user-defined objects. However, same is not true for the keys.

There are two important points to remember about dictionary keys −

(a) More than one entry per key not allowed. Which means no duplicate key is allowed. When duplicate keys encountered during assignment, the last assignment wins. For example −

Live Demo

```
#!/usr/bin/python

dict = {'Name': 'Zara', 'Age': 7, 'Name': 'Manni'}
print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result −

```
dict['Name']:  Manni
```

(b) Keys must be immutable. Which means you can use strings, numbers or tuples as dictionary keys but something like ['key'] is not allowed. Following is a simple example −

Live Demo

```
#!/usr/bin/python

dict = {['Name']: 'Zara', 'Age': 7}
print "dict['Name']: ", dict['Name']
```

When the above code is executed, it produces the following result −

```
Traceback (most recent call last):
  File "test.py", line 3, in <module>
    dict = {['Name']: 'Zara', 'Age': 7};
TypeError: unhashable type: 'list'
```

Tuples are immutable which means you cannot update or change the values of tuple elements. You are able to take portions of existing tuples to create new tuples as the following example demonstrates –

Live Demo

```
#!/usr/bin/python

tup1 = (12, 34.56);
tup2 = ('abc', 'xyz');

# Following action is not valid for tuples
# tup1[0] = 100;

# So let's create a new tuple as follows
tup3 = tup1 + tup2;
print tup3;
```

When the above code is executed, it produces the following result −

```
(12, 34.56, 'abc', 'xyz')
```

Delete Tuple Elements

Removing individual tuple elements is not possible. There is, of course, nothing wrong with putting together another tuple with the undesired elements discarded.

To explicitly remove an entire tuple, just use the del statement. For example −

Live Demo

```
#!/usr/bin/python

tup = ('physics', 'chemistry', 1997, 2000);
print tup;
del tup;
print "After deleting tup : ";
print tup;
```

This produces the following result. Note an exception raised, this is because after del tup tuple does not exist any more −

```
('physics', 'chemistry', 1997, 2000)
After deleting tup :
Traceback (most recent call last):
  File "test.py", line 9, in <module>
    print tup;
NameError: name 'tup' is not defined
```

DJANGO

Django is a high-level Python Web framework that encourages rapid development and clean, pragmatic design. Built by experienced developers, it takes care of much of the hassle of Web development, so you can focus on writing your app without needing to reinvent the wheel. It's free and open source.

Django's primary goal is to ease the creation of complex, database-driven websites. Django emphasizes reusabilityand "pluggability" of components, rapid development, and the principle of don't repeat yourself. Python is used throughout, even for settings files and data models.

Figure 2.3.1 Setting File and Data Models

Django also provides an optional administrative create, read, update and delete interface that is generated dynamically through introspection and configured via admin models

Figure 2.3.2 Admin Models

Create a Project

Whether you are on Windows or Linux, just get a terminal or a cmd prompt and navigate to the place you want your project to be created, then use this code −

```
$ django-admin startproject myproject
```

This will create a "myproject" folder with the following structure −

```
myproject/
  manage.py
  myproject/
    __init__.py
    settings.py
    urls.py
    wsgi.py
```

The Project Structure

The “myproject” folder is just your project container, it actually contains two elements −

manage.py − This file is kind of your project local django-admin for interacting with your project via command line (start the development server, sync db...). To get a full list of command accessible via manage.py you can use the code −

```
$ python manage.py help
```

The “myproject” subfolder − This folder is the actual python package of your project. It contains four files −

__init__.py − Just for python, treat this folder as package.

settings.py − As the name indicates, your project settings.

urls.py − All links of your project and the function to call. A kind of ToC of your project.

wsgi.py − If you need to deploy your project over WSGI.

Setting Up Your Project

Your project is set up in the subfolder myproject/settings.py. Following are some important options you might need to set −

DEBUG = True

This option lets you set if your project is in debug mode or not. Debug mode lets you get more information about your project's error. Never set it to ‘True’ for a live project. However, this has to be set to ‘True’ if you want the Django light server to serve static files. Do it only in the development mode.

```
DATABASES = {
  'default': {
    'ENGINE': 'django.db.backends.sqlite3',
    'NAME': 'database.sql',
    'USER': '',
```

```
    'PASSWORD': '',
    'HOST': '',
    'PORT': '',
  }
}
```

Database is set in the 'Database' dictionary. The example above is for SQLite engine. As stated earlier, Django also supports −

MySQL (django.db.backends.mysql)
PostGreSQL (django.db.backends.postgresql_psycopg2)
Oracle (django.db.backends.oracle) and NoSQL DB
MongoDB (django_mongodb_engine)
Before setting any new engine, make sure you have the correct db driver installed.

You can also set others options like: TIME_ZONE, LANGUAGE_CODE, TEMPLATE…

Now that your project is created and configured make sure it's working −

```
$ python manage.py runserver
```

You will get something like the following on running the above code −

```
Validating models...

0 errors found
September 03, 2015 - 11:41:50
Django version 1.6.11, using settings 'myproject.settings'
Starting development server at http://127.0.0.1:8000/
Quit the server with CONTROL-C.
```

A project is a sum of many applications. Every application has an objective and can be reused into another project, like the contact form on a website can be an application, and can be reused for others. See it as a module of your project.

Create an Application

We assume you are in your project folder. In our main “myproject” folder, the same folder then manage.py −

```
$ python manage.py startapp myapp
```

You just created myapp application and like project, Django create a “myapp” folder with the application structure −

```
myapp/
  __init__.py
  admin.py
  models.py
  tests.py
  views.py
```

__init__.py − Just to make sure python handles this folder as a package.

admin.py − This file helps you make the app modifiable in the admin interface.

models.py − This is where all the application models are stored.

tests.py − This is where your unit tests are.

views.py − This is where your application views are.

Get the Project to Know About Your Application

At this stage we have our "myapp" application, now we need to register it with our Django project "myproject". To do so, update INSTALLED_APPS tuple in the settings.py file of your project (add your app name) −

```
INSTALLED_APPS = (
  'django.contrib.admin',
  'django.contrib.auth',
  'django.contrib.contenttypes',
```

```
  'django.contrib.sessions',
  'django.contrib.messages',
  'django.contrib.staticfiles',
  'myapp',
)
```

Creating forms in Django, is really similar to creating a model. Here again, we just need to inherit from Django class and the class attributes will be the form fields. Let's add a forms.py file in myapp folder to contain our app forms. We will create a login form.

myapp/forms.py

```
#-*- coding: utf-8 -*-
from django import forms

class LoginForm(forms.Form):
  user = forms.CharField(max_length = 100)
  password = forms.CharField(widget = forms.PasswordInput())
```

As seen above, the field type can take "widget" argument for html rendering; in our case, we want the password to be hidden, not displayed. Many others widget are present in Django: DateInput for dates, CheckboxInput for checkboxes, etc.

Using Form in a View

There are two kinds of HTTP requests, GET and POST. In Django, the request object passed as parameter to your view has an attribute called "method" where the type of the request is set, and all data passed via POST can be accessed via the request.POST dictionary.

Let's create a login view in our myapp/views.py −

```
#-*- coding: utf-8 -*-
from myapp.forms import LoginForm

def login(request):
  username = "not logged in"
```

```
  if request.method == "POST":
    #Get the posted form
    MyLoginForm = LoginForm(request.POST)

    if MyLoginForm.is_valid():
      username = MyLoginForm.cleaned_data['username']
  else:
    MyLoginForm = Loginform()

  return render(request, 'loggedin.html', {"username" : username})
```

The view will display the result of the login form posted through the loggedin.html. To test it, we will first need the login form template. Let's call it login.html.

```
<html>
  <body>

    <form name = "form" action = "{% url "myapp.views.login" %}"
      method = "POST" >{% csrf_token %}

      <div style = "max-width:470px;">
        <center>
          <input type = "text" style = "margin-left:20%;"
            placeholder = "Identifiant" name = "username" />
        </center>
      </div>

      <br>

      <div style = "max-width:470px;">
        <center>
          <input type = "password" style = "margin-left:20%;"
            placeholder = "password" name = "password" />
```

```
        </center>
      </div>

      <br>

      <div style = "max-width:470px;">
        <center>

          <button style = "border:0px; background-color:#4285F4; margin-top:8%;
            height:35px; width:80%;margin-left:19%;" type = "submit"
            value = "Login" >
            <strong>Login</strong>
          </button>

        </center>
      </div>

    </form>

  </body>
</html>
```

The template will display a login form and post the result to our login view above. You have probably noticed the tag in the template, which is just to prevent Cross-site Request Forgery (CSRF) attack on your site.

```
{% csrf_token %}
```

Once we have the login template, we need the loggedin.html template that will be rendered after form treatment.

```
<html>

  <body>
    You are : <strong>{{username}}</strong>
```

```
  </body>

</html>
```

Now, we just need our pair of URLs to get started: myapp/urls.py

```
from django.conf.urls import patterns, url
from django.views.generic import TemplateView

urlpatterns = patterns('myapp.views',
  url(r'^connection/',TemplateView.as_view(template_name = 'login.html')),
  url(r'^login/', 'login', name = 'login'))
```

When accessing "/myapp/connection", we will get the following login.html template rendered −

Setting Up Sessions

In Django, enabling session is done in your project settings.py, by adding some lines to the MIDDLEWARE_CLASSES and the INSTALLED_APPS options. This should be done while creating the project, but it's always good to know, so MIDDLEWARE_CLASSES should have −

```
'django.contrib.sessions.middleware.SessionMiddleware'
```

And INSTALLED_APPS should have −

```
'django.contrib.sessions'
```

By default, Django saves session information in database (django_session table or collection), but you can configure the engine to store information using other ways like: in file or in cache.

When session is enabled, every request (first argument of any view in Django) has a session (dict) attribute.

Let's create a simple sample to see how to create and save sessions. We have built a simple login system before (see Django form processing chapter and Django Cookies Handling chapter). Let us save the username in a cookie so, if not signed out, when accessing our login page you won't see the login form. Basically, let's make our login system we used in Django Cookies handling more secure, by saving cookies server side.

For this, first lets change our login view to save our username cookie server side −

```
def login(request):
  username = 'not logged in'

  if request.method == 'POST':
    MyLoginForm = LoginForm(request.POST)

    if MyLoginForm.is_valid():
      username = MyLoginForm.cleaned_data['username']
      request.session['username'] = username
    else:
      MyLoginForm = LoginForm()

  return render(request, 'loggedin.html', {"username" : username}
```

Then let us create formView view for the login form, where we won't display the form if cookie is set –

```
def formView(request):
  if request.session.has_key('username'):
    username = request.session['username']
    return render(request, 'loggedin.html', {"username" : username})
  else:
    return render(request, 'login.html', {})
```

Now let us change the url.py file to change the url so it pairs with our new view −

```
from django.conf.urls import patterns, url
from django.views.generic import TemplateView

urlpatterns = patterns('myapp.views',
  url(r'^connection/','formView', name = 'loginform'),
  url(r'^login/', 'login', name = 'login'))
```

When accessing /myapp/connection, you will get to see the following page

Chapter 3 PROPOSED SYSTEMS

MODULES:

3.1User

3.**2 Admin**

3.3 **svm**

3.4 **randomforest.**

MODULES DESCRIPTION:

3.1 User:

The User can register the first. While registering he required a valid user email and mobile for further communications. Once the user registers, then admin can activate the customer. Once the admin activates the customer then the customer can login into our system. After login he can add the data to predict the traffic prediction . After adding the data we can find the prediction of the algorithm. First we can find the svm algorithm and then we can find the random forest algorithm.

3.2 Admin:

Admin can login with his credentials. Once he logs in he can activate the users. The activated user only login in our applications. The admin can set the predictions of algorithms.Admin can predict random forest algorithms and also predict the support vector machine algorithm. The admin can add new data to the dataset. .

3.3svm:

Support Vector Machine" (SVM) is a supervised machine learning algorithm which can be used for both classification or regression challenges. However, it is mostly used in classification problems. In the SVM algorithm, we plot each data item as a point in n-dimensional space (where n is number of features you have) with the value of each feature being the value of a particular coordinate. Then, we perform classification by finding the hyper-plane that differentiates the two classes very well .

3.4Random Forest:

Random forest is like a bootstrapping algorithm with Decision tree (CART) model. Say, we have 1000 observations in the complete population with 10 variables. Random forest tries to build multiple CART models with different samples and different initial variables. For instance, it will take a random sample of 100 observations and 5 randomly chosen initial variables to build a CART model. It will repeat the process (say) 10 times and then make a final prediction on each observation. Final prediction is a function of each prediction. This final prediction can simply be the mean of each prediction.

Chapter 4. SYSTEM DESIGN

SYSTEM ARCHITECTURE:

DATA FLOW DIAGRAM:

1. The DFD is also called as bubble chart. It is a simple graphical formalism that can be used to represent a system in terms of input data to the system, various processing carried out on this data, and the output data is generated by this system.
2. The data flow diagram (DFD) is one of the most important modeling tools. It is used to model the system components. These components are the system process, the data used by the process, an external entity that interacts with the system and the information flows in the system.
3. DFD shows how the information moves through the system and how it is modified by a series of transformations. It is a graphical technique that depicts information flow and the transformations that are applied as data moves from input to output.
4. DFD is also known as bubble chart. A DFD may be used to represent a system at any level of abstraction. DFD may be partitioned into levels that represent increasing information flow and functional detail.

Fig 4.1.1: System Architecture

UML DIAGRAMS

UML stands for Unified Modelling Language. UML is a standardized general-purpose modelling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group

The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with, UML

The Unified Modelling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems.

The UML represents a collection of best engineering practices that have proven successful in the modelling of large and complex systems.

The UML is a very important part of developing objects oriented software and the software development process. The UML uses mostly graphical notations to express the design of software projects.

GOALS:

The Primary goals in the design of the UML are as follows:

- Provide users a ready-to-use, expressive visual modelling Language so that they can develop and exchange meaningful models.
- Provide extendibility and specialization mechanisms to extend the core concepts.
- Be independent of particular programming languages and development process.
- Provide a formal basis for understanding the modelling language.
- Encourage the growth of OO tools market.
- Support higher level development concepts such as collaborations, frameworks, patterns and components.
- Integrate best practices.

USE CASE DIAGRAM:

A use case diagram in the Unified Modelling Language (UML) is a type of behavioural diagram defined by and created from a Use-case analysis. Its purpose is to present a graphical overview of the functionality provided by a system in terms of actors, their goals (represented as use cases), and any dependencies between those use cases. The main purpose of a use case diagram is to

show what system functions are performed for which actor. Roles of the actors in the system can be depicted.

Fig 4.1.2: **Use case Diagram**

CLASS DIAGRAM:

In software engineering, a class diagram in the Unified Modelling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the relationships among the classes. It explains which class contains information.

Fig 4.1.3: Class Diagram

SEQUENCE DIAGRAM:

A sequence diagram in Unified Modelling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message

Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagrams.

Fig 4.1.3 Sequence Diagram

ACTIVITY DIAGRAM:

Activity diagrams are graphical representations of workflows of stepwise activities and actions with support for choice, iteration and concurrency. In the Unified Modelling Language, activity diagrams can be used to describe the business and operational step-by-step workflows of components in a system. An activity diagram shows the overall flow of control.

Fig 4.1.4: Activity Diagram

Chapter 5. IMPLEMENTATION

REQUIREMENT ANALYSIS

The project involved analyzing the design of few applications so as to make the application more users friendly. To do so, it was really important to keep the navigations from one screen to the other well ordered and at the same time reducing the amount of typing the user needs to do. In order to make the application more accessible, the browser version had to be chosen so that it is compatible with most of the Browsers.

REQUIREMENT SPECIFICATION

Functional Requirements

- Graphical User interface with the User.

Software Requirements

For developing the application the following are the Software Requirements:

1. Python
2. Django

Operating Systems supported

1. Windows 7
2. Windows XP
3. Windows 8

Technologies and Languages used to Develop

1. Python

Debugger and Emulator

- Any Browser (Particularly Chrome)

Hardware Requirements

For developing the application the following are the Hardware Requirements:

- Processor: Pentium IV or higher
- RAM: 256 MB
- Space on Hard Disk: minimum 512

5.2 CODE:

url.py:

```
from django.contrib import admin
from django.urls import path
from users import views as users
from traffic import views as traffic


urlpatterns = [
    path('admin/', admin.site.urls),
    path('index/', users.index, name='index'),
    path('logout/',users.logout,name='logout'),
    path('userlogin/',users.userlogin,name='userlogin'),
    path('userregister/',users.userregister,name='userregister'),
    path('userlogincheck/',users.userlogincheck,name='userlogincheck'),
    path('randomforest/', users.randomforest, name='randomforest'),
    path('svm/', users.svm, name='svm'),
    path('adddata/', users.adddata, name='adddata'),


    path('adminlogin1/',traffic.adminlogin1,name='adminlogin1'),
    path('adminloginentered/',traffic.adminloginentered,name='adminloginentered'),
    path('storecsvdata/',traffic.storecsvdata,name='storecsvdata'),
    path('userdetails/',traffic.userdetails,name='userdetails'),
    path('activateuser/',traffic.activateuser,name='activateuser'),
```

views.py:

```
from collections import defaultdict
```

```
from io import TextIOWrapper
import io
from users.forms import *

from django.contrib import messages
from django.shortcuts import render, HttpResponse

from users.models import storetrafficdata
import csv
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd

from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import classification_report, confusion_matrix
import pandas as pd
from django_pandas.io import read_frame
```

def adminlogin1(request):

```
    return render(request, "admin/adminlogin.html")

def adminloginentered(request):
    if request.method == 'POST':
        uname=request.POST['uname']
        passwd=request.POST['upasswd']
        if uname =='admin' and passwd=='admin':
            return render(request,"admin/adminloginentered.html")
        else:
            return HttpResponse("invalied credentials")
```

```
    return render(request, "admin/adminloginentered.html")

def storecsvdata(request):
        if request.method == 'POST':
            #if request.method == "GET":
                #return render(request, template, prompt)
            csv_file = request.FILES['file']
            # let's check if it is a csv file
            if not csv_file.name.endswith('.csv'):
                messages.error(request, 'THIS IS NOT A CSV FILE')
            data_set = csv_file.read().decode('UTF-8')

            # setup a stream which is when we loop through each line we are able to handle a data in a stream
            io_string = io.StringIO(data_set)
            next(io_string)
            for column in csv.reader(io_string, delimiter=',', quotechar="|"):
                _, created = storetrafficdata.objects.update_or_create(
                    Date=column[1],
                    Day = column[2],
                    CodedDay = column[3],
                    Zone = column[4],
                    Weather = column[5],
                    Temperature = column[6],
                    Traffic = column[7]

                )
            context = {}

            '''
            name = request.POST.get('name')
            csvfile = TextIOWrapper(request.FILES['file'])
            # columns = defaultdict(list)
```

```
            storecsvdata=csv.DictReader(csvfile)

            for row1 in storecsvdata:
                Date = row1["Date"]
                Day = row1["Day"]
                CodedDay = row1["CodedDay"]
                Zone = row1["Zone"]
                Weather = row1["Weather"]
                Temperature = row1["Temperature"]
                Traffic = row1["Traffic"]



                storetrafficdata.objects.create(Date=Date, Day=Day, CodedDay=CodedDay,
                                                Zone=Zone, Weather=Weather, Temperature=Temperature,
                                                Traffic=Traffic)

            print("Name is ", csvfile)
            return HttpResponse('CSV file successful uploaded')
        else:
'''
        return render(request, 'admin/storecsvdata.html', {})
```

def userdetails(request):

```
    qs=userModel.objects.all()
    return render(request,"admin/userdetails.html",{"qs":qs})


def activateuser(request):
    if request.method =='GET':
        uname=request.GET.get('pid')
        print(uname)
        status='Activated'
        print("pid=",uname,"status=",status)
```

```
    userModel.objects.filter(id=uname).update(status=status)
    qs=userModel.objects.all()
    return render(request,"admin/userdetails.html",{"qs":qs})
```

models.py:

```
from django.db import models

# Create your models here.
class storetrafficdata(models.Model):
    Date=models.CharField(max_length=150)
    Day=models.CharField(max_length=100)
    CodedDay=models.CharField(max_length=100)
    Zone=models.CharField(max_length=100)
    Weather=models.CharField(max_length=100)
    Temperature=models.CharField(max_length=100)
    Traffic=models.CharField(max_length=100)

    def __str__(self):
        return self.Date, self.Day, self.CodedDay, self.Zone, self.Weather, self.Temperature, self.Traffic

    class Meta:
        db_table = 'storetrafficdata'

class userModel(models.Model):
    name = models.CharField(max_length=50)
    email = models.EmailField()
    passwd = models.CharField(max_length=40)
    cwpasswd = models.CharField(max_length=40)
    mobileno = models.CharField(max_length=50, default="", editable=True)
    status = models.CharField(max_length=40, default="", editable=True)
```

```
    def __str__(self):
        return self.email

    class Meta:
        db_table='userregiste
```

forms.py:

```
from users.models import *
from django import forms
from django.core import validators


class storetrafficdataForm(forms.ModelForm):

    class Meta():
        model = storetrafficdata
        fields = '__all__'
```

class userForm(forms.ModelForm):

```
    name = forms.CharField(widget=forms.TextInput(), required=True, max_length=100,)
    passwd = forms.CharField(widget=forms.PasswordInput(), required=True, max_length=100)
    cwpasswd = forms.CharField(widget=forms.PasswordInput(), required=True, max_length=100)
    email = forms.CharField(widget=forms.TextInput(),required=True)
    mobileno= forms.CharField(widget=forms.TextInput(), required=True,
max_length=10,validators=[validators.MaxLengthValidator(10),validators.MinLengthValidator(10)])
    status = forms.CharField(widget=forms.HiddenInput(), initial='waiting', max_length=100)

    def __str__(self):
        return self.email

    class Meta:
        model=userModel
```

```
    fields=['name','passwd','cwpasswd','email','mobileno','status']
```

storecsv.html:

```
{% extends 'adminbase.html' %}
{% load static %}

{% block contents %}
<div class="hero row">
        <div class="hero-right col-sm-6 col-sm-6">
          <h1 class="header-headline bold"> Traffic Prediction for Intelligent Transportatio System using Machine Learning <br></h1>
          <!--<h4 class="header-running-text light"> A top notch premium quality template for your next
            web project.</h4>-->
<!--          <a href="#email-form">-->
<!--            <button class="hero-btn"> Traffic</button>&ndash;&gt;-->
<!--          </a>-->
          <center><h3>welcome to admin page</h3>
            <form method="POST" enctype="multipart/form-data">
   {% csrf_token %}
      <center>
        <table>
          <!--<tr><td>Name<input type="text" name="name" value=""><br><br><br></td></tr>-->
           <tr><td> File<input type="file" name="file" value="file"><br><br><br></td></tr>
           <tr><td><button type="submit" >Save</button></tr></td>
        </table>
      </center>
</form>
          </center>
        </div><!--hero-left-->

<!-         <div class="col-sm-6 col-sm-6 ipad">-->
<!-          <img class="ipad-screen img-responsive" src="{% static 'images/traffic.png' %}"/>-->
```

```
<!--          </div>-->



    </div>
{% endblock %}
```

5.3 INPUT AND OUTPUT DESIGN

INPUT DESIGN

The input design is the link between the information system and the user. It comprises the developing specification and procedures for data preparation and those steps are necessary to put transaction data in to a usable form for processing can be achieved by inspecting the computer to read data from a written or printed document or it can occur by having people keying the data directly into the system.

The design of input focuses on controlling the amount of input required, controlling the errors, avoiding delay, avoiding extra steps and keeping the process simple. The input is designed in such a way so that it provides security and ease of use with retaining the privacy. Input Design considered the following things:

- What data should be given as input?
- How the data should be arranged or coded?
- The dialog to guide the operating personnel in providing input.
- Methods for preparing input validations and steps to follow when error occur.

OBJECTIVES

1.Input Design is the process of converting a user-oriented description of the input into a computer-based system. This design is important to avoid errors in the data input process and show the correct direction to the management for getting correct information from the computerized system.

2. It is achieved by creating user-friendly screens for the data entry to handle large volume of data. The goal of designing input is to make data entry easier and to be free from errors. The data entry screen is designed in such a way that all the data manipulates can be performed. It also provides record viewing facilities.

3. When the data is entered it will check for its validity. Data can be entered with the help of screens. Appropriate messages are provided as when needed so that the user will not be in maize of instant. Thus the objective of input design is to create an input layout that is easy to follow

OUTPUT DESIGN

A quality output is one, which meets the requirements of the end user and presents the information clearly. In any system results of processing are communicated to the users and to other system through outputs. In output design it is determined how the information is to be displaced for immediate need and also the hard copy output. It is the most important and direct source information to the user. Efficient and intelligent output design improves the system's relationship to help user decision-making.

1. Designing computer output should proceed in an organized, well thought out manner; the right output must be developed while ensuring that each output element is designed so that people will find the system can use easily and effectively. When analysis design computer output, they should Identify the specific output that is needed to meet the requirements.

2.Select methods for presenting information.

3.Create document, report, or other formats that contain information produced by the system.

The output form of an information system should accomplish one or more of the following objectives.

- Convey information about past activities, current status or projections of the
- Future.
- Signal important events, opportunities, problems, or warnings.
- Trigger an action.
- Confirm an action.

Screenshots

Home page:

Fig 5.3.1 Home page

User Register Form:

Fig 5.3.2 User Register Form

User Login Page :

Fig 5.3.3 **User Login Page**

User Home Page :

Fig 5.3.4 **User Home Page**

add data:

Fig 5.3.5 **add data**

Randomforest:

Fig 5.3.6 **Randomforest**

admin login:

Fig 5.3.7 **admin login**

admin page:

Fig 5.3.**8** **admin page**

user details:

Fig 5.3.9 **user details**

Chapter 6: TESTING

SYSTEM TEST

The purpose of this testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub assemblies, assemblies and/or a finished product it is the process of exercising software with the intent of ensuring that the Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement.

TYPES OF TESTS

Unit testing

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive.

Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

Integration testing

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

Functional test

Functional tests provide systematic demonstrations that functions tested are available as specified by the business and technical requirements, system documentation, and user manuals.

Functional testing is centered on the following items:

Valid Input	:	identified classes of valid input must be accepted.
Invalid Input	:	identified classes of invalid input must be rejected.
Functions	:	identified functions must be exercised.
Output	:	identified classes of application outputs must be exercised.
Systems/Procedures	:	interfacing systems or procedures must be invoked.

Organization and preparation of functional tests is focused on requirements, key functions, or special test cases. In addition, systematic coverage pertaining to identify Business process flows; data fields, predefined processes, and successive processes must be considered for testing.

Before functional testing is complete, additional tests are identified and the effective value of current tests is determined.

System Test

System testing ensures that the entire integrated software system meets requirements. It tests a configuration to ensure known and predictable results. An example of system testing is the configuration oriented system integration test. System testing is based on process descriptions and flows, emphasizing pre-driven process links and integration points.

White Box Testing

White Box Testing is a testing in which in which the software tester has knowledge of the inner workings, structure and language of the software, or at least its purpose. It is purpose. It is used to test areas that cannot be reached from a black box level.

Black Box Testing

Black Box Testing is testing the software without any knowledge of the inner workings, structure or language of the module being tested. Black box tests, as most other kinds of tests, must be written from a definitive source document, such as specification or requirements document, such as

specification or requirements document. It is a testing in which the software under test is treated, as a black box .you cannot "see" into it. The test provides inputs and responds to outputs without considering how the software works.

Unit Testing

Unit testing is usually conducted as part of a combined code and unit test phase of the software lifecycle, although it is not uncommon for coding and unit testing to be conducted as two distinct phases.

Test strategy and approach

Field testing will be performed manually and functional tests will be written in detail.

Test objectives

- All field entries must work properly.
- Pages must be activated from the identified link.
- The entry screen, messages and responses must not be delayed.

Features to be tested

- Verify that the entries are of the correct format
- No duplicate entries should be allowed
- All links should take the user to the correct page.

Integration Testing

Software integration testing is the incremental integration testing of two or more integrated software components on a single platform to produce failures caused by interface defects.

The task of the integration test is to check that components or software applications, e.g. components in a software system or – one step up – software applications at the company level – interact without error.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Acceptance Testing

User Acceptance Testing is a critical phase of any project and requires significant participation by the end user. It also ensures that the system meets the functional requirements.

Test Results: All the test cases mentioned above passed successfully. No defects encountered.

Sample Test Cases

S.no	Test Case	Excepted Result	Result	Remarks(IF Fails)
1.	User Register	If User registration successfully.	Pass	If already user email exists then it fails.
2.	User Login	If the Username and password is correct then it will be a valid page.	Pass	Un Register Users will not log in.
3.	Admin Add the Data	A new record will be added to our dataset.	Pass	According to India meteorological repository the data must be float fields otherwise it fails.
4.	user adddata	user will add the data to find out location prediction based on details	Pass	we couldn't get the required data to predict..
5.	randomforest	will get the random forest accuracy results.	Pass	won't get accuracy
6.	svm	will get the svm result	Pass	won't get prediction
7.				
8.	admin home	admin will get all	Pass	it won't display

	page	implemented algorithms and user details		any data
9.	Admin login	Admin can login with his login credential. If success he get his home page	Pass	Invalid login details will not allowed here
10.	Admin can activate the register users	Admin can activate the register user id	Pass	If the user did not find it then it won't login.

Table 6.1 Sample Test Cases

Chapter 7: CONCLUSION

7.1 Conclusion

Although deep learning and genetic algorithms are an important problem in data analysis, it has not been dealt with extensively by the ML community. The proposed algorithm gives higher accuracy than the existing algorithms also, It improves the complexity issues throughout the dataset. Also we have planned to integrate the web server and the application. Also the things algorithms will be further improved to much more higher accuracy

REFERENCES

[1] Fei-Yue Wang et al. Parallel control and management for intelligent transportation systems: Concepts, architectures, and applications. IEEE Transactions on Intelligent Transportation Systems, 2010.

[2] Yongchang Ma, Mashrur Chowdhury, Mansoureh Jeihani, and Ryan Fries. Accelerated incident detection across transportation networks using vehicle kinetics and support vector machines in cooperation with infrastructure agents. IET intelligent transport systems, 4(4):328–337, 2010.

[3] Rutger Claes, Tom Holvoet, and Danny Weyns. A decentralized approach for anticipatory vehicle routing using delegate multiagent systems. IEEE Transactions on Intelligent Transportation Systems, 12(2):364–373, 2011.

[4] Mehul Mahrishi and Sudha Morwal. Index point detection and semantic indexing of videos - a comparative review. Advances in Intelligent Systems and Computing, Springer, 2020.

[5] Joseph D Crabtree and Nikiforos Stamatiadis. Dedicated short-range communications technology for freeway incident detection: Performance assessment based on traffic simulation data. Transportation Research Record, 2000(1):59–69, 2007.

[6] H Qi, RL Cheu, and DH Lee. Freeway incident detection using kinematic data from probe vehicles. In 9th World Congress on Intelligent Transport SystemsITS America, ITS Japan, ERTICO (Intelligent Transport Systems and Services-Europe), 2002.

[7] Z. Zhao, W. Chen, X. Wu, P. C. Y. Chen, and J. Liu. Lstm network: a deep learning approach for short-term traffic forecast. IET Intelligent Transport Systems, 11(2):68–75, 2017.

[8] C. Zhang, P. Patras, and H. Haddadi. Deep learning in mobile and wireless networking: A survey. IEEE Communications Surveys Tutorials, 21(3):2224–2287, third quarter 2019.

[9] Chun-Hsin Wu, Jan-Ming Ho, and D. T. Lee. Travel-time prediction with support vector regression. IEEE Transactions on Intelligent Transportation Systems, 5(4):276–281, Dec 2004.

[10] Yan-Yan Song and LU Ying. Decision tree methods: applications for classification and prediction. Shanghai archives of psychiatry, 27(2):130, 2015.

[11] Yiming He, Mashrur Chowdhury, Yongchang Ma, and Pierluigi Pisu. Merging mobility and energy vision with hybrid electric vehicles and vehicle infrastructure integration. Energy Policy, 41:599–609,

2012. [12] Jason Brownlee. Bagging and random forest ensemble algorithms for machine learning. Machine Learning Algorithms, pages