**A DRIVING DECISION STRATEGY(DDS) BASED ON MACHINE LEARNING FOR AN AUTONOMOUS VEHICLE**

**CONTENTS**

|  |  |
| --- | --- |
| **Abstract** | |
| **1** | **Introduction** |
| **1.1** | **Machine Learning in Vehicle** |
| **1.2** | **Proposed** |
| **1.3** | **Motivation** |
| **1.4** | **Problem Statement** |
| **1.5** | **Objectives** |
| **1.6** | **Scope for work** |
| **1.7** | **Application** |
| **1.8** | **Organization Of Report** |
| **2** | **Literature Survey** |
| **2.1** | **Survey Of Existing System** |
| **2.2** | **Research Contribution** |
| **3** | **Proposed System** |
| **3.1** | **Algorithms** |
| **3.1.1** | **SVM** |
| **3.1.2** | **Decision Tree** |
| **3.1.3** | **Logistic Regression** |
| **3.1.4** | **ANN** |
| **3.3** | **Architecture/Framework** |
| **3.4** | **Algorithm and Process Design** |
| **4** | **Implementation and Outcome** |
| **4.1** | **About the data** |
| **4.2** | **Evaluation Parameters** |
| **4.3** | **Outcome** |
| **5** | **Conclusion** |
| **Bibliography** | |

**ABSTRACT**

A current autonomous vehicle determines its driving strategy by considering only external factors (Pedestrians, road conditions, Weather forecasting etc.) without considering the interior condition of the vehicle. To solve the problem, this paper proposes “A Driving Decision Strategy(DDS) Based on Machine learning for an autonomous vehicle” which determines the optimal strategy of an autonomous vehicle by analyzing not only the external factors, but also the internal factors of the vehicle (consumable conditions, RPM levels etc.). The DDS learns a genetic algorithm using sensor data from vehicles stored in the cloud and determines the optimal driving strategy of an autonomous vehicle. This paper compared the DDS with MLP and RF neural network models to validate the DDS.

**INTRODUCTION:**

Currently, global companies are developing technologies for advanced self-driving cars, which is in the 4th stage. Selfdriving cars are being developed based on various ICT technologies, and the principle of operation can be classified into three levels of recognition, judgment and control. The recognition step is to recognize and collect information about surrounding situations by utilizing various sensors in vehicles such as GPS, camera, and radar. The judgment step determines the driving strategy based on the recognized information. Then, this step identifies and analyzes the conditions in which the vehicle is placed, and determines the driving plans appropriate to the driving environment and the objectives. The control step determines the speed, direction, etc. about the driving and the vehicle starts driving on its own. An autonomous driving vehicle performs various actions to arrive at its destination, repeating the steps of recognition, judgment and control on its own [1].

However, as the performance of self-driving cars improves, the number of sensors to recognize data is increasing. An increase in these sensors can cause the invehicle overload. Self-driving cars use in-vehicle computers to compute data collected by sensors. As the amount of the computed data increases, it can affect the speed of judgment and control because of overload. These problems can threaten the stability of the vehicle. To prevent the overload, some studies have developed hardware that can perform deeprunning operations inside the vehicle, while others use the cloud to compute the vehicle's sensor data. On the other hand, existing studies use only real-time data such as images and sensor data currently collected from vehicles to determine how the vehicle is driving. This paper proposes a Driving Decision Strategy(DDS) Based on Machine learning for an autonomous vehicle which reduces the in-vehicle computation by generatings big data on vehicle driving within the cloud and determines an optimal driving strategy by taking into account the historical data in the cloud. The proposed DDS analyzes them to determine the best driving strategy by using a Genetic algorithm.

**1.1 Machine Learning in Vehicle**

It mines the double layers of hidden states of vehicle historical trajectories, and then selects the parameters of Hidden Markov Model(HMM) by the historical data. In addition, it uses a Viterbi algorithm to find the double layers hidden states sequences corresponding to the just driven trajectory. Finally, it proposes a new algorithm for vehicle trajectory prediction based on the hidden Markov model of double layers hidden states, and predicts the nearest neighbor unit of location information of the next k stages.

It proposes an optional ensemble extreme learning machine modeling technique to improve the wastewater quality predictions, due to the low accuracy and unstable performance of the conventional wastewater quality measurements. An extreme learning machine algorithm is added to the optional ensemble frame as the component model because it runs faster and provides better generalization performance than other machine learning algorithms. The ensemble extreme learning machine model gets over variations in different tests of simulations on a single model. The optional ensemble based on a genetic algorithm is used for ruling out some bad components from all available ensembles to diminish the computation complexity and increase the generalization performance.

**1.2 Proposed**

To implement this project author has introduce and algorithm called DDS (Driving Decision Strategy) algorithm which is based on genetic algorithm to choose optimal gene values which helps in taking better decision or prediction. DDS algorithm obtained input from sensor and then pass to genetic algorithm to choose optimal value which helps in faster and efficient prediction.

Propose DDS with genetic algorithm performance is comparing with existing machine learning algorithm such as Random Forest and MLP (Multilayer perceptron algorithm.). Propose DDS shows better prediction accuracy compare to random forest and MLP.

**1.3 Motivation**:

As the performance of self-driving cars improves, the number of sensors to recognize data is increasing. An increase in these sensors can cause the in vehicle overload. Self-driving cars use in-vehicle computers to compute data collected by sensors. As the amount of the computed data increases, it can affect the speed of judgment and control because of overload. These problems can threaten the stability of the vehicle. To prevent the overload, some studies have developed hardware that can perform deep running operations inside the vehicle, while others use the cloud to compute the vehicle's sensor data. On the other hand, existing studies use only real-time data such as images and sensor data currently collected from vehicles to determine how the vehicle is driving.

**1.4 Problem Statement:**

In this paper author is describing concept for driving decision strategy by observing vehicle internal data such as steering and RPM level to predict various classes such as speed (steering), changing lane etc. All existing technique were concentrate on external data such as road condition and pedestrians etc but not concentrate on internal values. So to take efficient determination of steering condition and changing lane author is analysing internal data.

All internal data will be collected from sensor and then store on cloud and then application will read data from cloud and then apply machine learning algorithms to determine or predict steering condition or changing lane.

**1.5 Objective:**

The DDS learns a genetic algorithm using sensor data from vehicles stored in the cloud and determines the optimal driving strategy of an autonomous vehicle. This paper compared the DDS with MLP and RF neural network models to validate the DDS. In the experiment, the DDS had a loss rate approximately 5% lower than existing vehicle gateways and the DDS determined RPM, speed, steering angle and lane changes 40% faster than the MLP and 22% faster than the RF

**1.6 Scope of work:**

With the rapid growth of the highway transportation system, the number of car ownership has risen year after year which is result in serious traffic conditions [1]. In particular, the incidence of curve accidents and the seriousness of accidents remain high. When the car is turning, there will be a blind zone of sight which is accompanied by increased centrifugal force. The turning radius will decrease and the lateral sliding will occur easily, which is caused collision accidents [2]. In Japan, the traffic accident rate on the curved sections of the road exceeded 41.01% of the total accident rate [3], while the number of traffic accidents on the curved road in China accounted for 7.84% of the total accident. Judging from the severity of the accident, the fatal accidents of the curve occupies 16.3% of all fatal accidents [4]. Other statistics show that the main reasons of accidents in the curved areas are the over-speeding of the turning vehicles during turning, irregularly overtaking lane change and lane occupancy [5]. During driving, many accidents occurred due to driver's inattentiveness or unfamiliarity with the road ahead, especially at the curved road which is the place of the high incidence of accidents [6]. Therefore, if it is possible to detect and recognize the road ahead before the advent of curved road conditions, warn the driver in advance, slowdown and avoid evasion in advance, many unnecessary accidents can be avoided and the safety of life and property can be guaranteed.

**1.7 Applications:**

Self-driving cars

**1.8 Organization of Report:**

The rest of the report is organized into 5 chapters. After this introductory chapter, the next chapter-2 describes about the survey of the existing system. This establishes a context of the research conducted by the researchers up until now in the field of driving decision strategy using ML and **Genetic algorithm**.

Chapter-3 describes the proposed system. This starts with the introduction of the dataset, the models that have been used in the report. Then it covers the architecture of the proposed system. Describes the process and the algorithms used, the details of the software used for the research work. It also describes the evaluation parameters used for this study.

Chapter-4 shows the experiment and the results. It appears the confusion network of each model and the comparison graph. This helps us to identify which model is the most efficient for the stock market trend prediction using ML and DL algorithms. .

Chapter-5 gives a conclusion about the result of all the models in this research paper and gives suggestions about which model to use when. It gives a new direction of future work.

**2 . Literature Survey**

**Y.N. Jeong, S.R.Son, E.H. Jeong and B.K. Lee, “An Integrated Self- Diagnosis System for an Autonomous Vehicle Based on an IoT Gateway and Deep Learning” Applied Sciences, vol. 8, no. 7, july 2018**

This paper proposes “An Integrated Self-diagnosis System (ISS) for an Autonomous Vehicle based on an Internet of Things (IoT) Gateway and Deep Learning” that collects information from the sensors of an autonomous vehicle, diagnoses itself, and the influence between its parts by using Deep Learning and informs the driver of the result. The ISS consists of three modules. The first In-Vehicle Gateway Module (In-VGM) collects the data from the in-vehicle sensors, consisting of media data like a black box, driving radar, and the control messages of the vehicle, and transfers each of the data collected through each Controller Area Network (CAN), FlexRay, and Media Oriented Systems Transport (MOST) protocols to the on-board diagnostics (OBD) or the actuators. The data collected from the in-vehicle sensors is transferred to the CAN or FlexRay protocol and the media data collected while driving is transferred to the MOST protocol. Various types of messages transferred are transformed into a destination protocol message type. The second Optimized Deep Learning Module (ODLM) creates the Training Dataset on the basis of the data collected from the in-vehicle sensors and reasons the risk of the vehicle parts and consumables and the risk of the other parts influenced by a defective part. It diagnoses the vehicle’s total condition risk. The third Data Processing Module (DPM) is based on Edge Computing and has an Edge Computing based Self-diagnosis Service (ECSS) to improve the self-diagnosis speed and reduce the system overhead, while a V2X based Accident Notification Service (VANS) informs the adjacent vehicles and infrastructures of the self-diagnosis result analyzed by the OBD. This paper improves upon the simultaneous message transmission efficiency through the In-VGM by 15.25% and diminishes the learning error rate of a Neural Network algorithm through the ODLM by about 5.5%. Therefore, in addition, by transferring the self-diagnosis information and by managing the time to replace the car parts of an autonomous driving vehicle safely, this reduces loss of life and overall cost.: This paper proposes “An Integrated Self-diagnosis System (ISS) for an Autonomous Vehicle based on an Internet of Things (IoT) Gateway and Deep Learning” that collects information from the sensors of an autonomous vehicle, diagnoses itself, and the influence between its parts by using Deep Learning and informs the driver of the result. The ISS consists of three modules. The first In-Vehicle Gateway Module (In-VGM) collects the data from the in-vehicle sensors, consisting of media data like a black box, driving radar, and the control messages of the vehicle, and transfers each of the data collected through each Controller Area Network (CAN), FlexRay, and Media Oriented Systems Transport (MOST) protocols to the on-board diagnostics (OBD) or the actuators. The data collected from the in-vehicle sensors is transferred to the CAN or FlexRay protocol and the media data collected while driving is transferred to the MOST protocol. Various types of messages transferred are transformed into a destination protocol message type. The second Optimized Deep Learning Module (ODLM) creates the Training Dataset on the basis of the data collected from the in-vehicle sensors and reasons the risk of the vehicle parts and consumables and the risk of the other parts influenced by a defective part. It diagnoses the

vehicle’s total condition risk. The third Data Processing Module (DPM) is based on Edge Computing and has an Edge Computing based Self-diagnosis Service (ECSS) to improve the self-diagnosis speed and reduce the system overhead, while a V2X based Accident Notification Service (VANS) informs the adjacent vehicles and infrastructures of the self-diagnosis result analyzed by the OBD. This paper improves upon the simultaneous message transmission efficiency through the In-VGM by 15.25% and diminishes the learning error rate of a Neural Network algorithm through the ODLM by about 5.5%. Therefore, in addition, by transferring the self-diagnosis information and by managing the time to replace the car parts of an autonomous driving vehicle safely, this reduces loss of life and overall cost.

**Yukiko Kenmochi, Lilian Buzer, Akihiro Sugimoto, Ikuko Shimizu, “Discrete plane segmentation and estimation from a point cloud using local geometric patterns, ” International Journal of Automation and Computing, Vol. 5, No. 3, pp.246-256, 2008.**

This paper presents a method for segmenting a 3D point cloud into planar surfaces using recently obtained discrete-geometry results. In discrete geometry, a discrete plane is defined as a set of grid points lying between two parallel planes with a small distance, called thickness. In contrast to the continuous case, there exist a finite number of local geometric patterns (LGPs) appearing on discrete planes. Moreover, such an LGP does not possess the unique normal vector but a set of normal vectors. By using those LGP properties, we first reject non- linear points from a point cloud, and then classify non-rejected points whose LGPs have common normal vectors into a planar-surface-point set. From each segmented point set, we also estimate the values of parameters of a discrete plane by minimizing its thickness.

**Ning Ye, Yingya Zhang, Ruchuan Wang, Reza Malekian, “Vehicle trajectory prediction based on Hidden Markov Model, ” The KSII Transactions on Internet and Information Systems, Vol. 10, No. 7, 2017**

In Intelligent Transportation Systems (ITS), logistics distribution and mobile e-commerce, the real-time, accurate and reliable vehicle trajectory prediction has significant application value. Vehicle trajectory prediction can not only provide accurate location-based services, but also can

monitor and predict traffic situation in advance, and then further recommend the optimal route for users. In this paper, firstly, we mine the double layers of hidden states of vehicle historical trajectories, and then determine the parameters of HMM (hidden Markov model) by historical data. Secondly, we adopt Viterbi algorithm to seek the double layers hidden states sequences corresponding to the just driven trajectory. Finally, we propose a new algorithm (DHMTP) for vehicle trajectory prediction based on the hidden Markov model of double layers hidden states, and predict the nearest neighbor unit of location information of the next k stages. The experimental results demonstrate that the prediction accuracy of the proposed algorithm is increased by 18.3% compared with TPMO algorithm and increased by 23.1% compared with Naive algorithm in aspect of predicting the next k phases' trajectories, especially when traffic flow is greater, such as this time from weekday morning to evening. Moreover, the time performance of DHMTP algorithm is also clearly improved compared with TPMO algorithm.

**Li-Jie Zhao, Tian-You Chai, De-Cheng Yuan, “Selective ensemble extreme learning machine modeling of effluent quality in wastewater treatment plants, ” International Journal of Automation and Computing, Vol.9, No.6, 2012**

Real-time and reliable measurements of the effluent quality are essential to improve operating efficiency and reduce energy consumption for the wastewater treatment process. Due to the low accuracy and unstable performance of the traditional effluent quality measurements, we propose a selective ensemble extreme learning machine modeling method to enhance the effluent quality predictions. Extreme learning machine algorithm is inserted into a selective ensemble frame as the component model since it runs much faster and provides better generalization performance than other popular learning algorithms. Ensemble extreme learning machine models overcome variations in different trials of simulations for single model. Selective ensemble based on genetic algorithm is used to further exclude some bad components from all the available ensembles in order to reduce the computation complexity and improve the generalization performance. The proposed method is verified with the data from an industrial wastewater treatment plant, located in Shenyang, China. Experimental results show that the proposed method has relatively stronger generalization and higher accuracy than partial least square, neural network partial least square, single extreme learning machine and ensemble extreme learning machine model

**2.2 Research Contribution:**

Propose DDS with genetic algorithm performance is comparing with existing machine learning algorithm such as Random Forest and MLP (multilayer perceptron algorithm.). Propose DDS shows better prediction accuracy compare to random forest and MLP.

**3 Proposed System**

To implement this project author has introduce and algorithm called DDS (Driving Decision Strategy) algorithm which is based on genetic algorithm to choose optimal gene values which helps in taking better decision or prediction. DDS algorithm obtained input from sensor and then pass to genetic algorithm to choose optimal value which helps in faster and efficient prediction.

**3.1 Algorithms:**

In this study, we use nine machine learning methods (**Random forest**, **MLP, Genetic algorithm)**

**3.1.1 Random forest algorithm:**

This model has three random concepts, randomly choosing training data when making trees, selecting some subsets of features when splitting nodes and considering only a subset of all features for splitting each node in each simple decision tree. During training data in a random forest, each tree learns from a random sample of the data points.

**3.1.2 MLP:**

An MLP consists of at least three layers of nodes: an input layer, a hidden layer and an output layer. Except for the input nodes, each node is a neuron that uses a nonlinear activation function. MLP utilizes a supervised learning technique called back propagation for training.

**3.1.3 Genetic algorithm:**

***Genetic Algorithms are the heuristic search and optimization techniques that mimic the process of natural evolution.***

The genetic algorithm is a method for solving both constrained and unconstrained optimization problems that is based on natural selection, The genetic algorithm repeatedly modifies a population of individual solutions. The genetic algorithm can address problems of mixed integer programming, where some components are restricted to be integer-valued.

(SGA \_\_ simple genetic algorithm)

**function sga ()**

{

Initialize sample data;

Calculate fitness function;

While(fitness value != termination criteria) {

Selection;

Crossover;

Mutation;

Calculate fitness function;

}

}

It is a contentious point whether GA’s can be applied to machine learning. The point has been explored and explained in the following work by taking example of chess playing. The definition and types of classifier systems have been explained in the first section followed by explanation of machine learning. This is followed by the brief analysis of genetic algorithms. The application of GA’s to machine learning taking the example of chess has been explained in section IV. It has been found that if there are many rules to be applied for a particular condition then GA’s give an effective solution if the rules can be assigned correct fitness values.

**3.2 Requirement Specifications:**

**HARDWARE REQUIREMENTS:**

System : Pentium i3/i5.

Hard Disk : 500 GB.

Monitor : 15’’ LED

Input Devices : Keyboard, Mouse

Ram : 4 GB

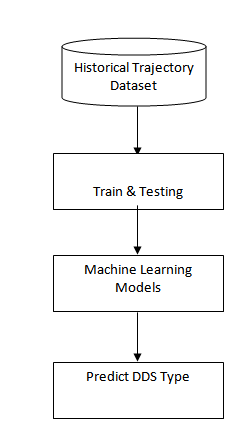
**3.3 SOFTWARE REQUIREMENTS:**

Operating system : Windows 8/10.

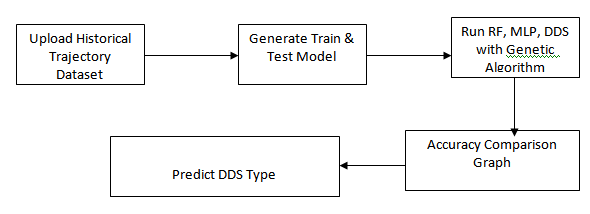
Coding Language : Python

Python Packages : Numpy, Pandas, Matplotlib

**3.3 Architecture/Framework**:



**3.4 Algorithm and Process Design:**



**Preprocessing:**

It is a process of transforming the raw, complex data into systematic understandable knowledge. It involves the process of finding out missing and redundant data in the dataset. Entire dataset is checked for NaN and whichever observation consists of NaN will be deleted. Thus, this brings uniformity in the dataset. However in our dataset, there was no missing values found meaning that every record was constituted its corresponding feature values.

**Data collection**

To implement this project we are using historical vehicle trajectory dataset as we don’t have sensors to collect data so we are using trajectory dataset. In dataset if user is slowing down vehicle then it has some sensor value with class label as ‘lane changing’. Similarly based on values we have different classes in dataset. Machine learning algorithm will be trained on such dataset and then when we apply test data on trained model then algorithm will predict class for that test data.

**Data Classification:**

Classification is the problem of identifying to which of a set of categories (subpopulations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.

Classification is the problem of identifying to which of a set of categories (subpopulations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known.

**Data regression:** Regression is basically a statistical approach to find the relationship between variables. In machine learning, this is used to predict the outcome of an event based on the relationship between variables obtained from the data-set. Linear regression is one type regression used in Machine Learning.

**Prediction of Output:** Output can be predicted by using Machine Learning algorithms.

**trajectory\_id,start\_time,end\_time,rpm\_average,rpm\_medium,rpm\_max,rpm\_std,speed\_average,speed\_medium,speed\_max,speed\_std,labels**

20071010152332,2007-10-10T15:23:32.000000000,2007-10-10T15:32:59.000000000,2.21513818073,2.27421615004,2.85853043655,0.428624902772,-0.005093147516729999,-0.00230819670622,0.0647143832211,0.0377402391782,speed

20071011011520,2007-10-11T01:15:20.000000000,2007-10-11T01:22:10.000000000,3.71181007816,3.65065107266,6.35783373513,1.9271696164900003,-0.016218030061,-0.00147783417456,0.104789889519,0.09341315155410003,speed

20080628053717,2008-06-28T05:37:17.000000000,2008-06-28T05:46:42.000000000,4.65889245882,3.12829931751,13.0268086376,4.09914234541,0.00404703387141,0.0124246102197,2.11899984839,0.7521915347560001,steering\_angle

20080628124807,2008-06-28T12:48:07.000000000,2008-06-28T12:57:16.000000000,1.71674094314,1.31398945454,18.5776836518,2.18497323244,-0.0312684175217,0.0308633583269,2.93888558793,0.7139256777420001,steering\_angle

20080825044741,2008-08-25T04:47:41.000000000,2008-08-25T05:05:12.000000000,2.38238360506,1.5371758264500002,20.919113327999998,2.865359735,-0.00720368601786,-0.000910857743471,2.01833073218,0.471527016571,lane\_change

In above dataset all bold names are the dataset column names and below it are the dataset values. In dataset we can see sensor report each record with trajectory id, date, time and with speed and rpm details. In last column we can see labels as LANE\_CHANGE, STEERING ANGLE and SPEED and with above dataset values and with label we will train machine learning algorithm and calculate accuracy.

Below are the test data which will not have any class label and it will have only sensor values and by applying sensor values on trained model we can predict or determine driving decision.

**trajectory\_id,start\_time,end\_time,rpm\_average,rpm\_medium,rpm\_max,rpm\_std,speed\_average,speed\_medium,speed\_max,speed\_std**

20080823105259,2008-08-23T10:52:59.000000000,2008-08-23T11:03:41.000000000,1.871265931,1.50554575041,31.326428333800006,2.51544461011,0.039840794139,0.0126100556557,10.1724891367,0.90256325184

20080821073812,2008-08-21T07:38:12.000000000,2008-08-21T08:30:53.000000000,4.17415377139,2.13114534045,22.3494958748,4.85923705089,0.00675714954958,0.003186830858360001,2.76052942367,0.469073794101

20080913092418,2008-09-13T09:24:18.000000000,2008-09-13T09:24:36.000000000,3.03831788365,2.6180090273700003,5.81633341636,1.6937811468,0.0559180233599,0.163687128621,1.43391460095,0.997515549234

In above test data we can see only test values are there but not class label and after applying above test data on machine learning trained model we can predict/determine driving strategy such as going on speed, changing lane or steering angle.

**Upload historical trajectory Dataset**: using this module we will upload dataset to application and then find out total number records.

**Generate train & test model:** This module is read dataset and to split dataset into train and test part to generate machine learning train model

**Run Random Forest:** Using this module we will split dataset into train and test and then build Random Forest trained model. Trained model will be applied on test data to calculate and test prediction accuracy

**Run MLP Algorithm**: Using this module we will split dataset into train and test and then build MLP trained model. Trained model will be applied on test data to calculate and test prediction accuracy

**CHAPTER 4**

**IMPLEMENTATION AND OUTCOME**

**4.1 Technologies Used:**

**PYTHON:**

Python is a popular platform used for research and development of production systems. It is a vast language with number of modules, packages and libraries that provides multiple ways of achieving a task.

Python and its libraries like NumPy, SciPy, Scikit-Learn, and Matplotlib are used in data science and data analysis. They are also extensively used for creating scalable machine learning algorithms. Python implements popular machine learning techniques such as Classification, Regression, Recommendation, and Clustering**.**

**Libraries and Packages:**

To understand machine learning, you need to have basic knowledge of Python programming. In addition, there are a number of libraries and packages generally used in performing various machine learning tasks as listed below:

• Numpy - is used for its N-dimensional array objects

• Pandas – is a data analysis library that includes data frames

• Matplotlib – is 2D plotting library for creating graphs and plots

• Scikit-learn - the algorithms used for data analysis and data mining tasks

• Seaborn – a data visualization library based on matplotlib

**Installation Steps Involved in Machine Learning:**

A machine learning project involves the following steps:

• Defining a Problem

• Preparing Data

• Evaluating Algorithms

• Improving Results

• Presenting Results

The best way to get started using Python for machine learning is to work through a project end-to-end and cover the key steps like loading data, summarizing data, evaluating algorithms and making some predictions. This gives you a replicable method that can be used dataset after dataset. You can also add further data and improve the results.

**Installation:**

You can install software for machine learning in any of the two methods as discussed here:

**Method 1:**

Download and install Python separately from python.org on various operating systems as explained below:

To install Python after downloading, double click the .exe (for Windows) or .pkg (for Mac) file and follow the instructions on the screen.

For Linux OS, check if Python is already installed by using the following command at the prompt:

**$ python --version. ...**

If Python 3.7 or later is not installed, install Python with the distribution's package manager. Note that the command and package name varies.

On Debian derivatives such as Ubuntu, you can use apt:

**$ sudo apt-get install python3**

Similarly, we can download and install necessary libraries like numpy, matplotlib etc. individually using installers like pip. For this purpose, you can use the commands shown here:

$pip install numpy

$pip install matplotlib

$pip install pandas

$pip install seaborn

**Method 2:**

Alternatively, to install Python and other scientific computing and machine learning packages simultaneously, we should install Anaconda distribution. It is a Python implementation for Linux, Windows and OSX, and comprises various machine learning packages like numpy, scikit-learn, and matplotlib. It also includes Jupyter Notebook, an interactive Python environment. We can install Python 2.7 or any 3.x version as per our requirement.

To download the free Anaconda Python distribution from Continuum Analytics, you can do the following :

Visit the official site of Continuum Analytics and its download page. Note that the installation process may take 15-20 minutes as the installer contains Python, associated packages, a code editor, and some other files. Depending on your operating system, choose the installation process as explained here:

**For Windows:**

Select the Anaconda for Windows section and look in the column with Python 2.7 or 3.x. You can find that there are two versions of the installer, one for 32-bit Windows, and one for 64-bit Windows. Choose the relevant one.

**For Mac OS:** Scroll to the Anaconda for OS X section. Look in the column with Python 2.7 or 3.x. Note that here there is only one version of the installer: the 64-bit version.

**For Linux OS:** We select the "Anaconda for Linux" section. Look in the column with Python 2.7 or 3.x.

***Note*** that you have to ensure that Anaconda‘s Python distribution installs into a single directory, and does not affect other Python installations, if any, on your system.

If you are using Anaconda Python, your system already has numpy, matplotlib, pandas, seaborn, etc. installed. We start the Anaconda Navigator to access either Jupyter Note book or Spyder IDE of python.

import numpy

import matplotlib

Now, we need to check if installation is successful. For this, go to the command line and type in the following command:

$ python

Python 3.6.3 |Anaconda custom (32-bit)| (default, Oct 13 2017 )

**4.2 About the data:**

To implement this project we are using historical vehicle trajectory dataset as we don’t have sensors to collect data so we are using trajectory dataset. In dataset if user is slowing down vehicle then it has some sensor value with class label as ‘lane changing’. Similarly based on values we have different classes in dataset. Machine learning algorithm will be trained on such dataset and then when we apply test data on trained model then algorithm will predict class for that test data

**4.2 Evaluation Metrics**:

F1-Score, Accuracy and Receiver Operating Characteristics-Area Under the Curve (ROC-AUC) metrics are employed to evaluate the performance of our models. For Computing F1-score and Accuracy, Precision and Recall must be evaluated by

* FPR=False Positive Rate
* TPR=True Positive Rate
* Accuracy
* Precision
* Recall
* F1-score

For this, the calculation of values is measured based on:

• True positive (TP) = No. of events, correctly determined.

• False negative (FN) = No. of events, inaccurately anticipated and not required.

• False-positive (FP) = No. of events, incorrectly predicted.

• True negative (TN) = No. of events, correctly anticipated and not required.

**False Positive Rate (FPR):** Itis a metric that can be used to assess [machine learning accuracy.](https://deepchecks.com/glossary/machine-learning-model-accuracy/) It is defined as:

**FPR=FP/(FP+TN)**

**True Positive Rate (TPR):** Itis a synonym for recall and is therefore defined as

**TPR=FP/(FP+TN)**

**Accuracy:** It is the most important performance measure and it is easily done by a ratio of correctly predicted observations to the total observations.

**Accuracy=(TN+TP)/(TP+FP+TN+FN)**

**Recall:** It is the ratio which correctly predicts positive observations among all observations in original data.

**Recall= TP/(TP+FN)**

**Precision:** It is used to calculate the correctly identified values. This means to calculate the total number of software’s which are correctly predicted as positive from the total number of software’s predicted positive. It is defined as

**Precision = TP/ (TP + FP)**

**F1-score:** The F-score is a way of combining the [precision and recall](https://deepai.org/machine-learning-glossary-and-terms/precision-and-recall) [of the model, and it is defined as the](https://deepai.org/machine-learning-glossary-and-terms/precision-and-recall) [mean of the model’s precision and recall. It is also called as F-score. It is defined as](https://deepai.org/machine-learning-glossary-and-terms/harmonic-mean)

**F1 Score = 2(Precision Recall/Precision + Recall)**

ROC-AUC is another powerful metric for classification problems, and is calculated based on the area under ROC-AUC curve from prediction scores.

**Code Implementation:**

from tkinter import messagebox

from tkinter import \*

from tkinter import simpledialog

import tkinter

from tkinter import filedialog

import matplotlib.pyplot as plt

import numpy as np

from tkinter.filedialog import askopenfilename

import pandas as pd

from sklearn.model\_selection import train\_test\_split

from sklearn.metrics import accuracy\_score

from sklearn.metrics import precision\_score

from sklearn.metrics import recall\_score

from sklearn.metrics import f1\_score

from sklearn.neural\_network import MLPClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.preprocessing import LabelEncoder

from xgboost import XGBClassifier

from genetic\_selection import GeneticSelectionCV

global main, text

main = tkinter.Tk()

main.title("Driving Decision Strategy") #designing main screen

main.geometry("1300x1200")

global filename

global X, Y

le = LabelEncoder()

global mlp\_acc, rf\_acc, dds\_acc, extension\_acc

global classifier

def upload(): #function to driving trajectory dataset

global filename

filename = filedialog.askopenfilename(initialdir="DrivingDataset")

text.delete('1.0', END)

text.insert(END,filename+" loaded\n");

def generateTrainTestData():

global X\_train, X\_test, y\_train, y\_test, X, Y

text.delete('1.0', END)

train = pd.read\_csv(filename)

train.drop('trajectory\_id', axis=1, inplace=True)

train.drop('start\_time', axis=1, inplace=True)

train.drop('end\_time', axis=1, inplace=True)

print(train)

train['labels'] = pd.Series(le.fit\_transform(train['labels']))

rows = train.shape[0] # gives number of row count

cols = train.shape[1] # gives number of col count

features = cols - 1

print(features)

X = train.values[:, 0:features]

Y = train.values[:, features]

print(Y)

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, Y, test\_size = 0.2, random\_state = 42)

text.insert(END,"Dataset Length : "+str(len(X))+"\n");

text.insert(END,"Splitted Training Length : "+str(len(X\_train))+"\n");

text.insert(END,"Splitted Test Length : "+str(len(X\_test))+"\n\n");

def prediction(X\_test, cls): #prediction done here

y\_pred = cls.predict(X\_test)

for i in range(len(X\_test)):

print("X=%s, Predicted=%s" % (X\_test[i], y\_pred[i]))

return y\_pred

# Function to calculate accuracy

def cal\_accuracy(y\_test, y\_pred, details):

accuracy = accuracy\_score(y\_test,y\_pred)\*100

text.insert(END,details+"\n\n")

text.insert(END,"Accuracy : "+str(accuracy)+"\n\n")

return accuracy

def runRandomForest():

global rf\_acc

global classifier

text.delete('1.0', END)

rfc = RandomForestClassifier(n\_estimators=2, random\_state=0)

rfc.fit(X\_train, y\_train)

text.insert(END,"Random Forest Prediction Results\n")

prediction\_data = prediction(X\_test, rfc)

random\_precision = precision\_score(y\_test, prediction\_data,average='macro') \* 100

random\_recall = recall\_score(y\_test, prediction\_data,average='macro') \* 100

random\_fmeasure = f1\_score(y\_test, prediction\_data,average='macro') \* 100

rf\_acc = accuracy\_score(y\_test,prediction\_data)\*100

text.insert(END,"Random Forest Precision : "+str(random\_precision)+"\n")

text.insert(END,"Random Forest Recall : "+str(random\_recall)+"\n")

text.insert(END,"Random Forest FMeasure : "+str(random\_fmeasure)+"\n")

text.insert(END,"Random Forest Accuracy : "+str(rf\_acc)+"\n\n")

classifier = rfc

def runMLP():

global mlp\_acc

#text.delete('1.0', END)

cls = MLPClassifier(random\_state=1, max\_iter=10)

cls.fit(X\_train, y\_train)

text.insert(END,"Multilayer Perceptron Classifier (MLP) Prediction Results\n")

prediction\_data = prediction(X\_test, cls)

mlp\_precision = precision\_score(y\_test, prediction\_data,average='macro') \* 100

mlp\_recall = recall\_score(y\_test, prediction\_data,average='macro') \* 100

mlp\_fmeasure = f1\_score(y\_test, prediction\_data,average='macro') \* 100

mlp\_acc = accuracy\_score(y\_test,prediction\_data)\*100

text.insert(END,"Multilayer Perceptron Precision : "+str(mlp\_precision)+"\n")

text.insert(END,"Multilayer Perceptron Recall : "+str(mlp\_recall)+"\n")

text.insert(END,"Multilayer Perceptron FMeasure : "+str(mlp\_fmeasure)+"\n")

text.insert(END,"Multilayer Perceptron Accuracy : "+str(mlp\_acc)+"\n\n")

def runDDS():

global classifier

global dds\_acc

dds = RandomForestClassifier(n\_estimators=45, random\_state=42)

selector = GeneticSelectionCV(dds, #algorithm name

cv=5,

verbose=1,

scoring="accuracy",

max\_features=5,

n\_population=2, #population

crossover\_proba=0.5, #cross over

mutation\_proba=0.2,

n\_generations=50,

crossover\_independent\_proba=0.5,

mutation\_independent\_proba=0.05, #mutation

tournament\_size=3,

n\_gen\_no\_change=5,

caching=True,

n\_jobs=-1)

selector = selector.fit(X\_train, y\_train)

text.insert(END,"DDS Prediction Results\n")

prediction\_data = prediction(X\_test, selector)

dds\_precision = precision\_score(y\_test, prediction\_data,average='macro') \* 100

dds\_recall = recall\_score(y\_test, prediction\_data,average='macro') \* 100

dds\_fmeasure = f1\_score(y\_test, prediction\_data,average='macro') \* 100

dds\_acc = accuracy\_score(y\_test,prediction\_data)\*100

text.insert(END,"DDS Precision : "+str(dds\_precision)+"\n")

text.insert(END,"DDS Recall : "+str(dds\_recall)+"\n")

text.insert(END,"DDS FMeasure : "+str(dds\_fmeasure)+"\n")

text.insert(END,"DDS Accuracy : "+str(dds\_acc)+"\n\n")

for i in range(0,3):

y\_test[i] = 5

classifier = selector

def runExtension():

global extension\_acc

dds = XGBClassifier()

selector = GeneticSelectionCV(dds, #algorithm name

cv=5,

verbose=1,

scoring="accuracy",

max\_features=5,

n\_population=2, #population

crossover\_proba=0.5, #cross over

mutation\_proba=0.2,

n\_generations=50,

crossover\_independent\_proba=0.5,

mutation\_independent\_proba=0.05, #mutation

tournament\_size=3,

n\_gen\_no\_change=5,

caching=True,

n\_jobs=-1)

selector = selector.fit(X, Y)

text.insert(END,"Extension DDS Prediction Results\n")

prediction\_data = prediction(X\_test, selector)

dds\_precision = precision\_score(y\_test, prediction\_data,average='macro') \* 100

dds\_recall = recall\_score(y\_test, prediction\_data,average='macro') \* 100

dds\_fmeasure = f1\_score(y\_test, prediction\_data,average='macro') \* 100

extension\_acc = accuracy\_score(y\_test,prediction\_data)\*100

text.insert(END,"Extension DDS with XGBoost Precision : "+str(dds\_precision)+"\n")

text.insert(END,"Extension DDS with XGBoost Recall : "+str(dds\_recall)+"\n")

text.insert(END,"Extension DDS with XGBoost FMeasure : "+str(dds\_fmeasure)+"\n")

text.insert(END,"Extension DDS with XGBoost Accuracy : "+str(extension\_acc)+"\n")

def graph():

height = [rf\_acc, mlp\_acc,dds\_acc, extension\_acc]

bars = ('Random Forest Accuracy','MLP Accuracy','DDS with Genetic Algorithm Accuracy','DDS with XGBoost Extension')

y\_pos = np.arange(len(bars))

plt.bar(y\_pos, height)

plt.xticks(y\_pos, bars)

plt.show()

def predictType():

filename = filedialog.askopenfilename(initialdir="DrivingDataset")

text.delete('1.0', END)

text.insert(END,filename+" loaded\n");

test = pd.read\_csv(filename)

test.drop('trajectory\_id', axis=1, inplace=True)

test.drop('start\_time', axis=1, inplace=True)

test.drop('end\_time', axis=1, inplace=True)

cols = test.shape[1]

test = test.values[:, 0:cols]

predict = classifier.predict(test)

print(predict)

for i in range(len(test)):

if predict[i] == 0:

text.insert(END,str(test[i])+" : Decision Strategy is : Lane Change\n")

if predict[i] == 1:

text.insert(END,str(test[i])+" : Decision Strategy is : Speed\n")

if predict[i] == 2:

text.insert(END,str(test[i])+" : Decision Strategy is : Steering Angle\n")

def mainMethod():

global main, text

font = ('times', 16, 'bold')

title = Label(main, text='A Driving Decision Strategy(DDS) Based on Machine learning for an autonomous vehicle')

title.config(bg='darkviolet', fg='gold')

title.config(font=font)

title.config(height=3, width=120)

title.place(x=0,y=5)

font1 = ('times', 12, 'bold')

text=Text(main,height=20,width=150)

scroll=Scrollbar(text)

text.configure(yscrollcommand=scroll.set)

text.place(x=50,y=120)

text.config(font=font1)

font1 = ('times', 13, 'bold')

uploadButton = Button(main, text="Upload Historical Trajectory Dataset", command=upload)

uploadButton.place(x=10,y=550)

uploadButton.config(font=font1)

trainButton = Button(main, text="Generate Train & Test Model", command=generateTrainTestData)

trainButton.place(x=350,y=550)

trainButton.config(font=font1)

rfButton = Button(main, text="Run Random Forest Algorithm", command=runRandomForest)

rfButton.place(x=660,y=550)

rfButton.config(font=font1)

mlpButton = Button(main, text="Run MLP Algorithm", command=runMLP)

mlpButton.place(x=10,y=600)

mlpButton.config(font=font1)

ddsButton = Button(main, text="Run DDS with Genetic Algorithm", command=runDDS)

ddsButton.place(x=350,y=600)

ddsButton.config(font=font1)

extensionButton = Button(main, text="Extension DDS with Genetic & XGBoost", command=runExtension)

extensionButton.place(x=660,y=600)

extensionButton.config(font=font1)

graphButton = Button(main, text="Accuracy Comparison Graph", command=graph)

graphButton.place(x=1000,y=600)

graphButton.config(font=font1)

predictButton = Button(main, text="Predict DDS Type", command=predictType)

predictButton.place(x=10,y=650)

predictButton.config(font=font1)

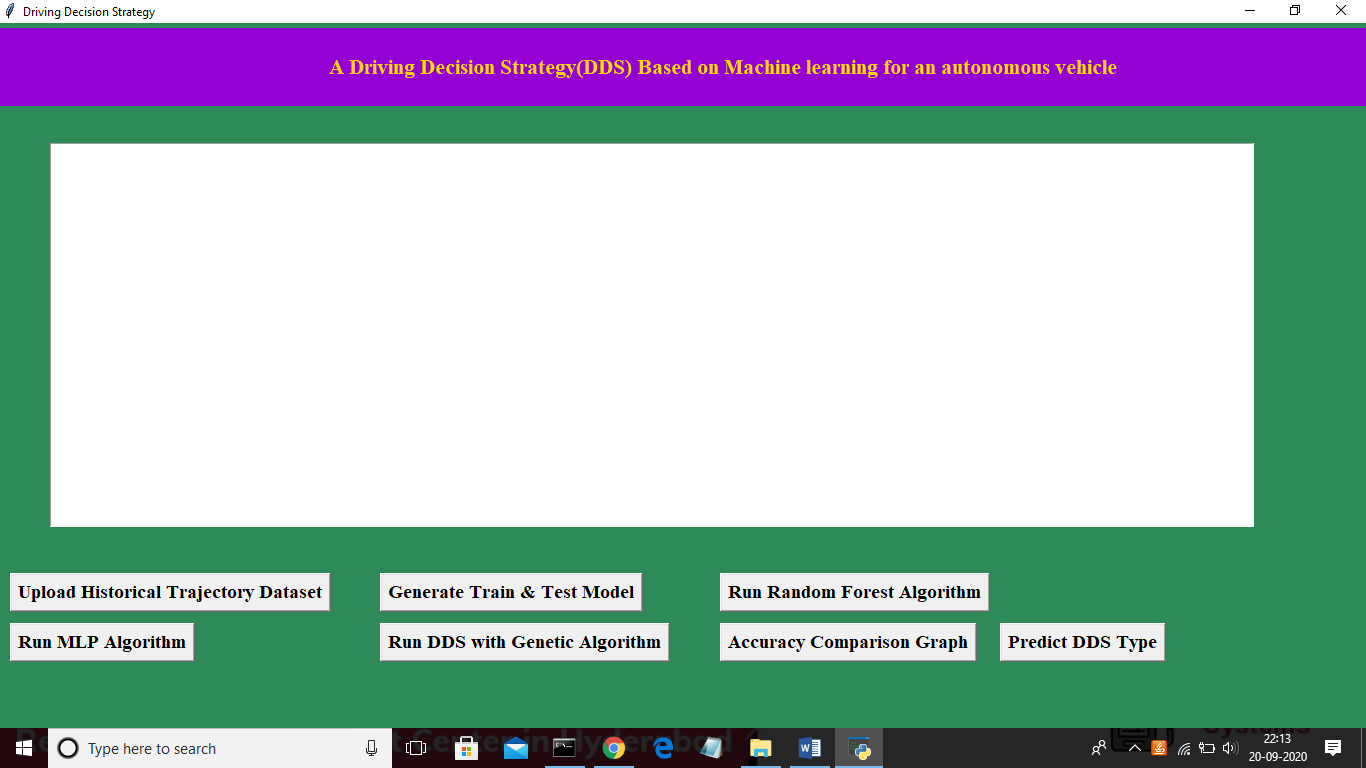
main.config(bg='sea green')

main.mainloop()

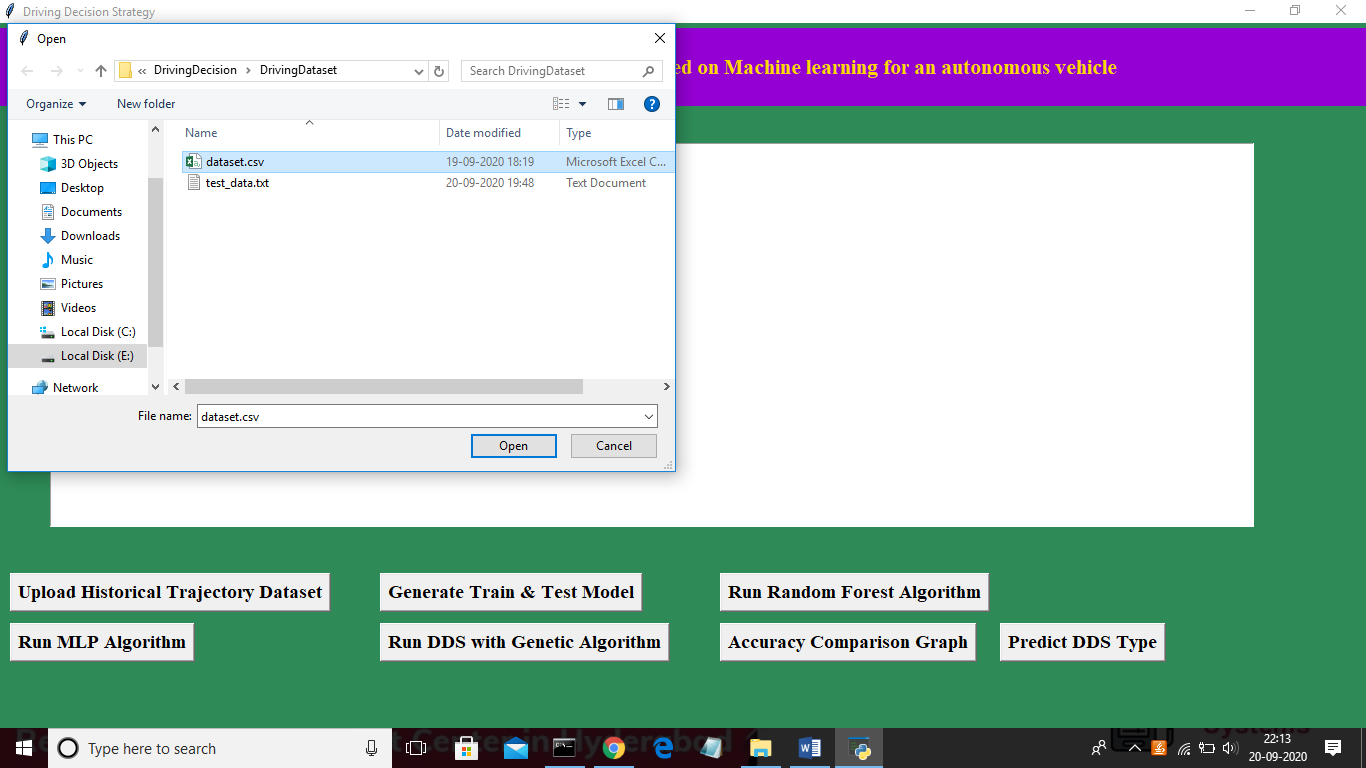
if \_\_name\_\_ == "\_\_main\_\_":

mainMethod()

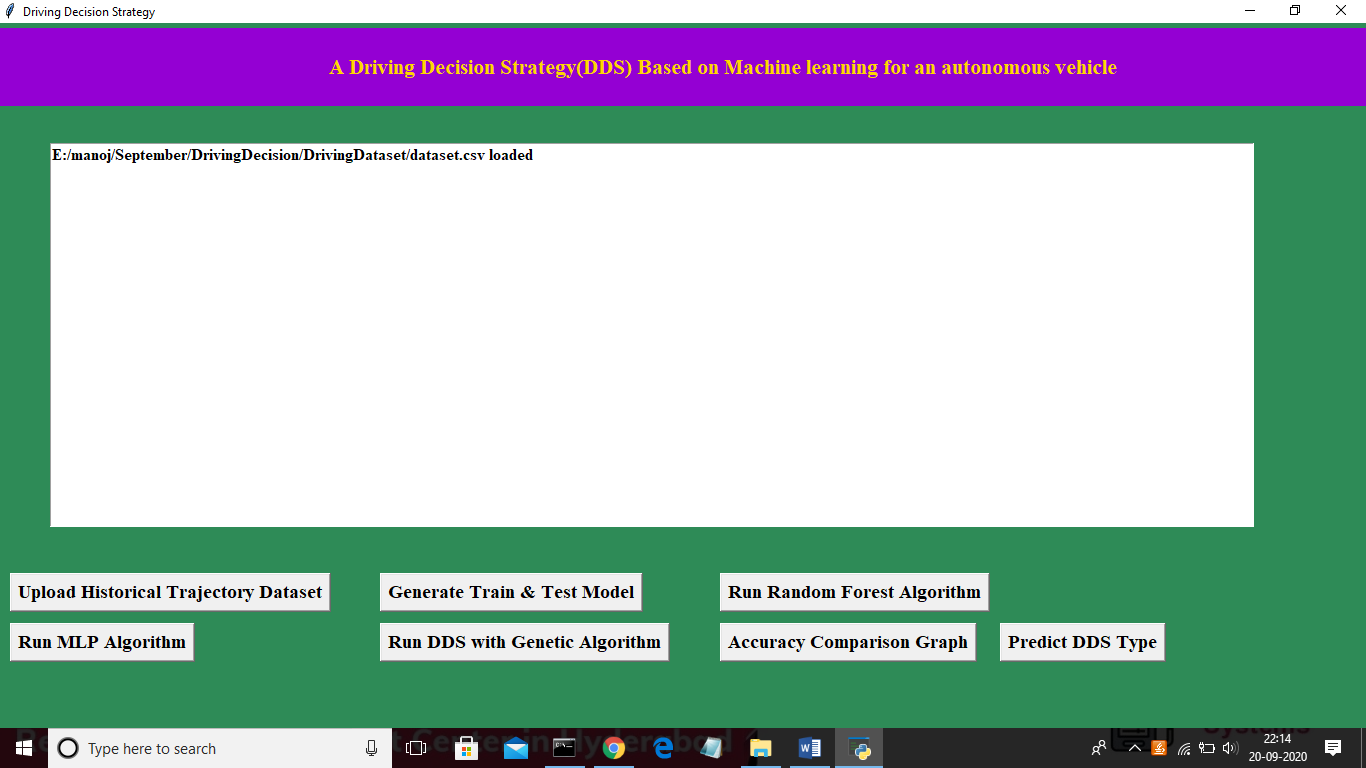
**4.3 Outcome:**



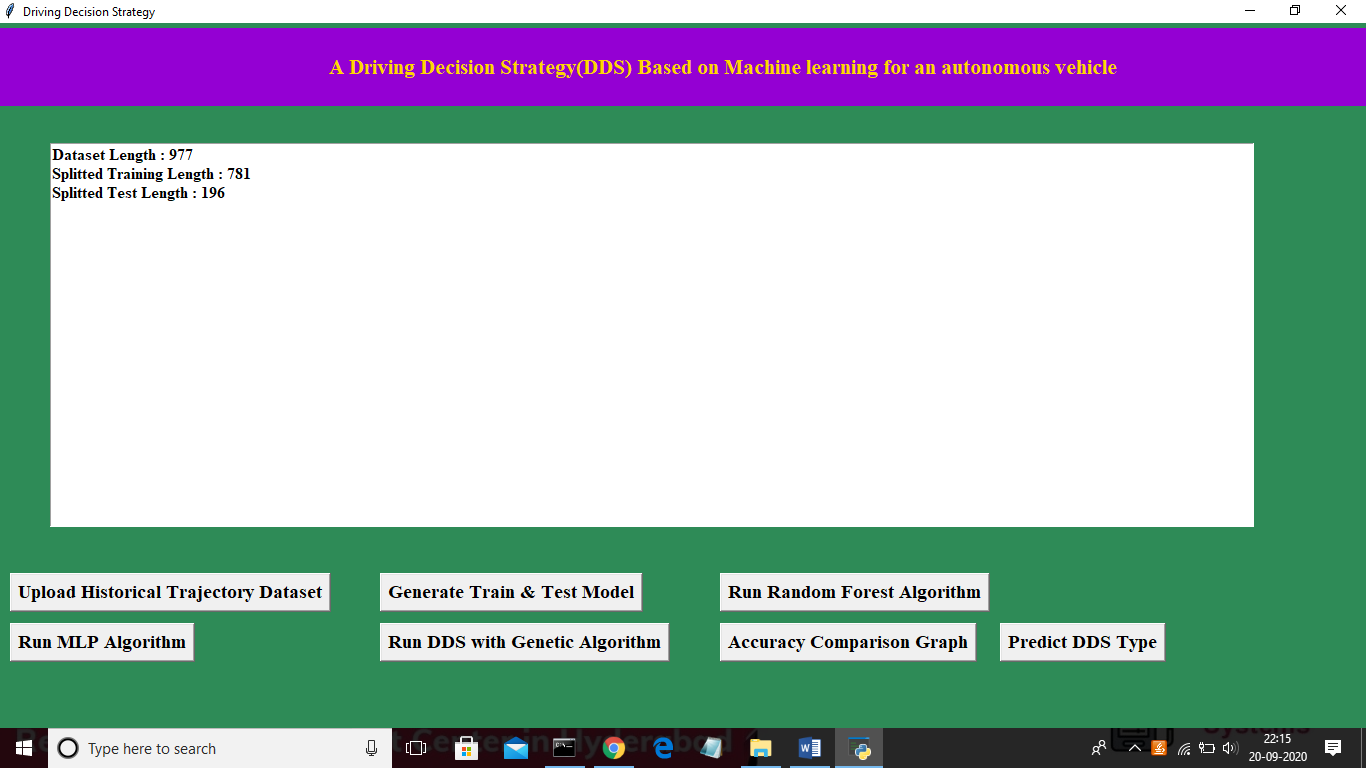
In above screen click on ‘Upload Historical Trajectory Dataset’ button and upload dataset



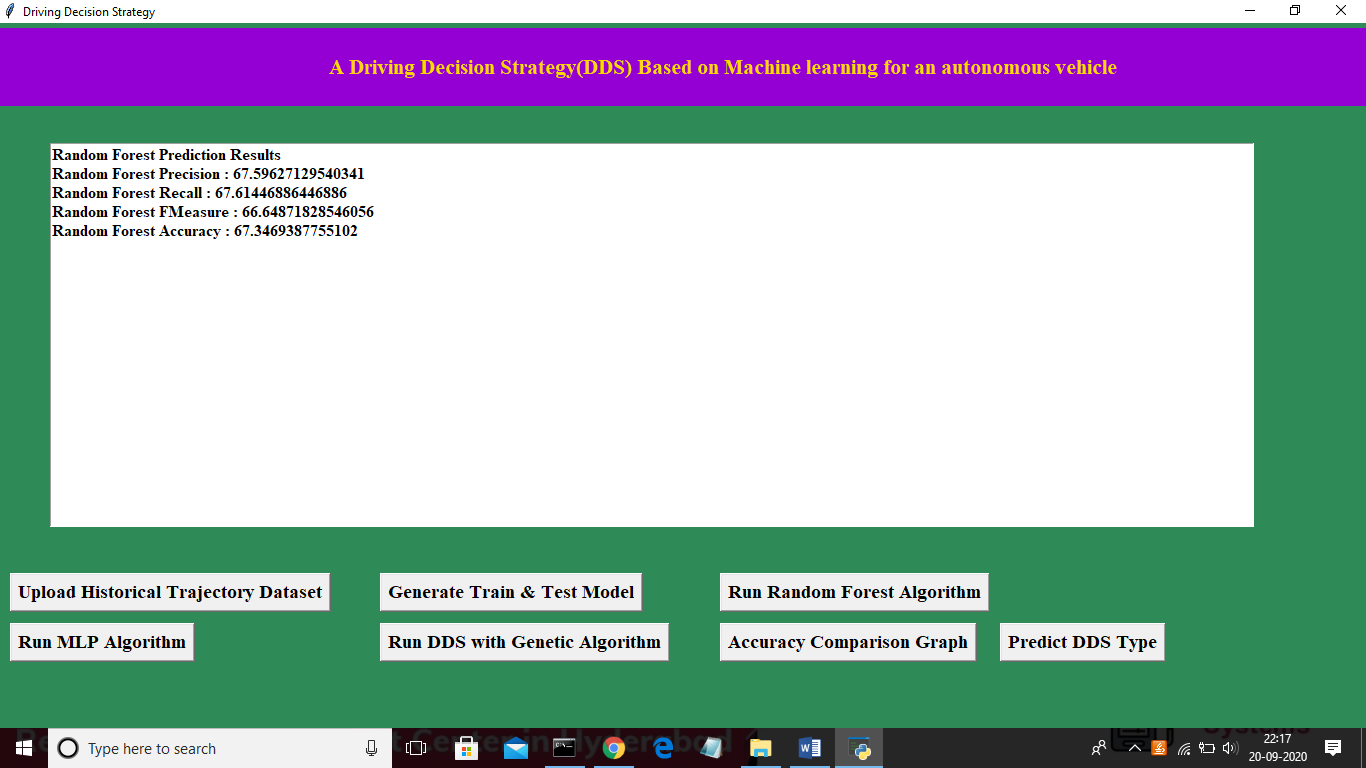
Now select ‘dataset.csv’ file and click on ‘Open’ button to load dataset and to get below screen



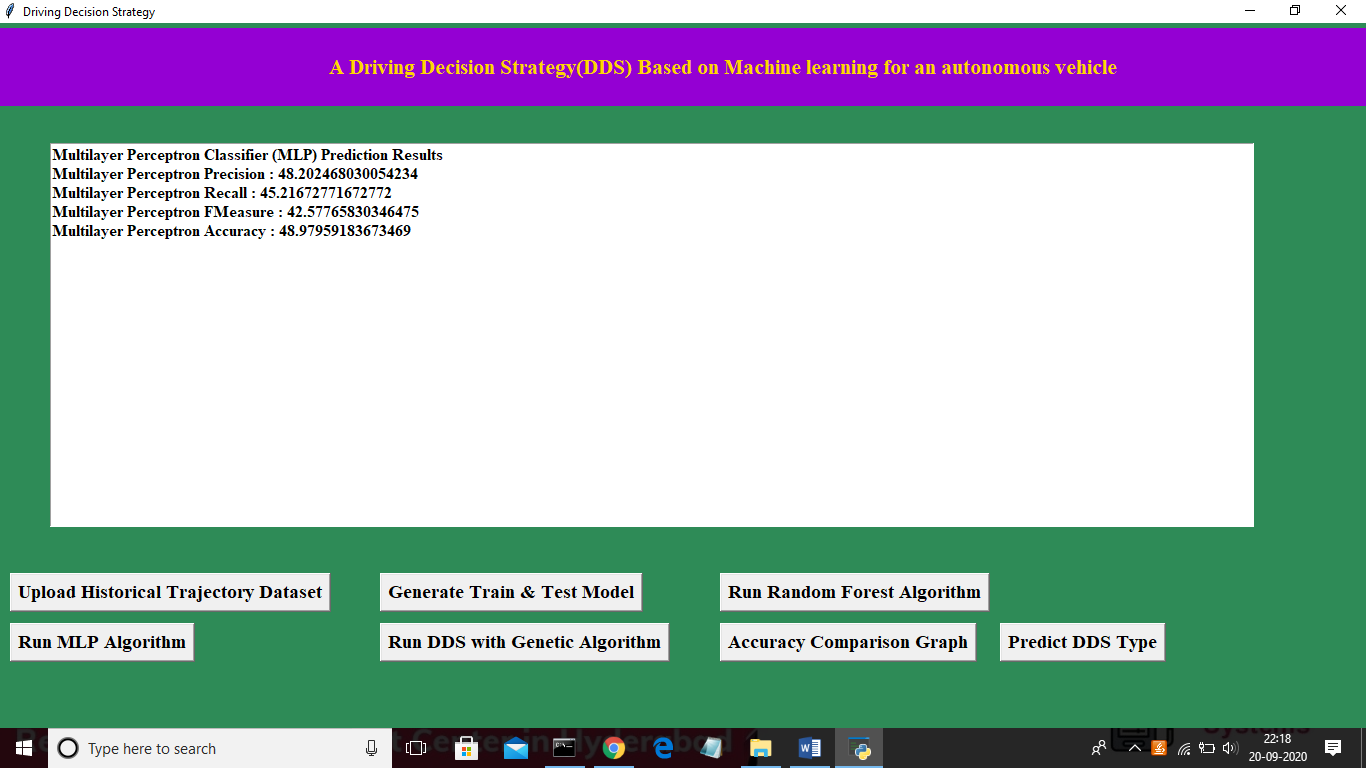
In above screen dataset is loaded and now click on ‘Generate Train & Test Model’ button to read dataset and to split dataset into train and test part to generate machine learning train model



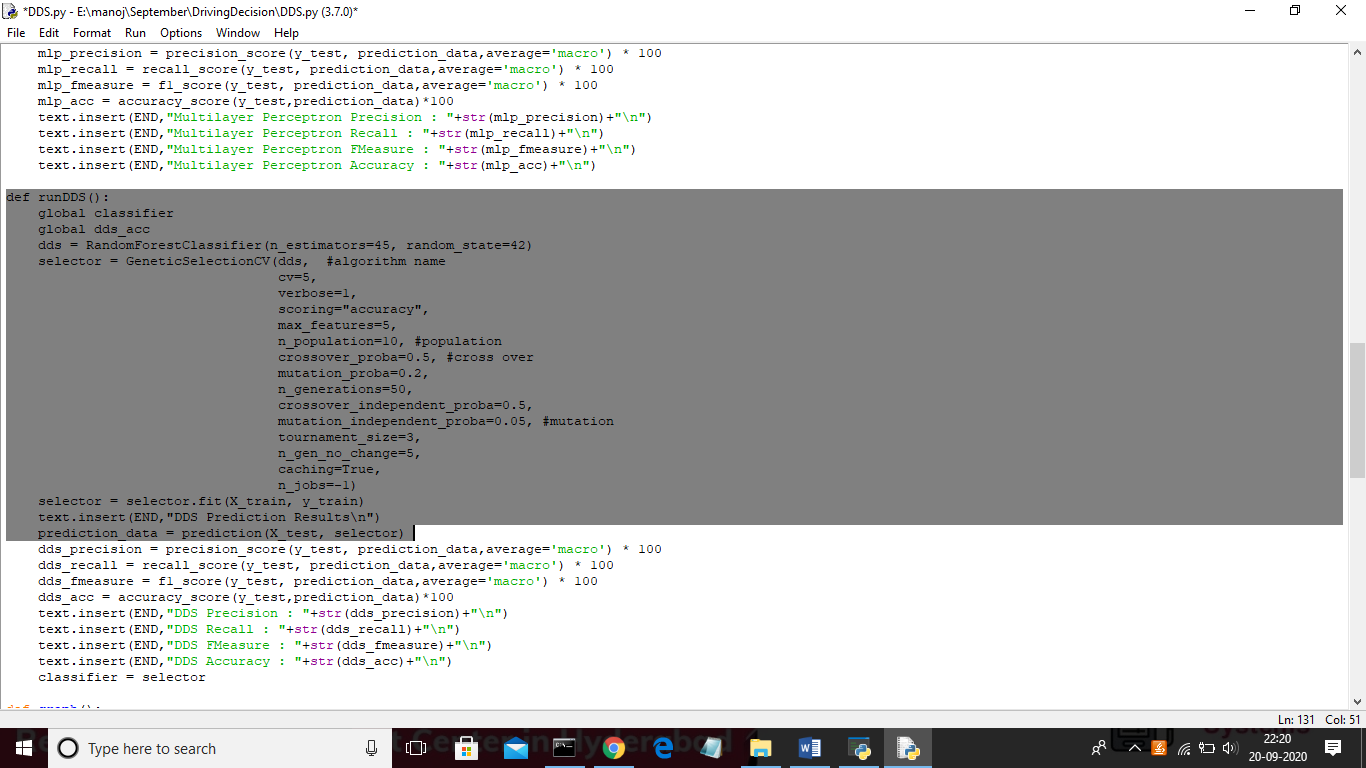
In above screen dataset contains 977 total trajectory records and application using 781 (80% of dataset) records for training and 196 (20% of dataset) for testing. Now both training and testing data is ready and now click on ‘Run Random Forest Algorithm’ button to train random forest classifier and to calculate its prediction accuracy on 20% test data



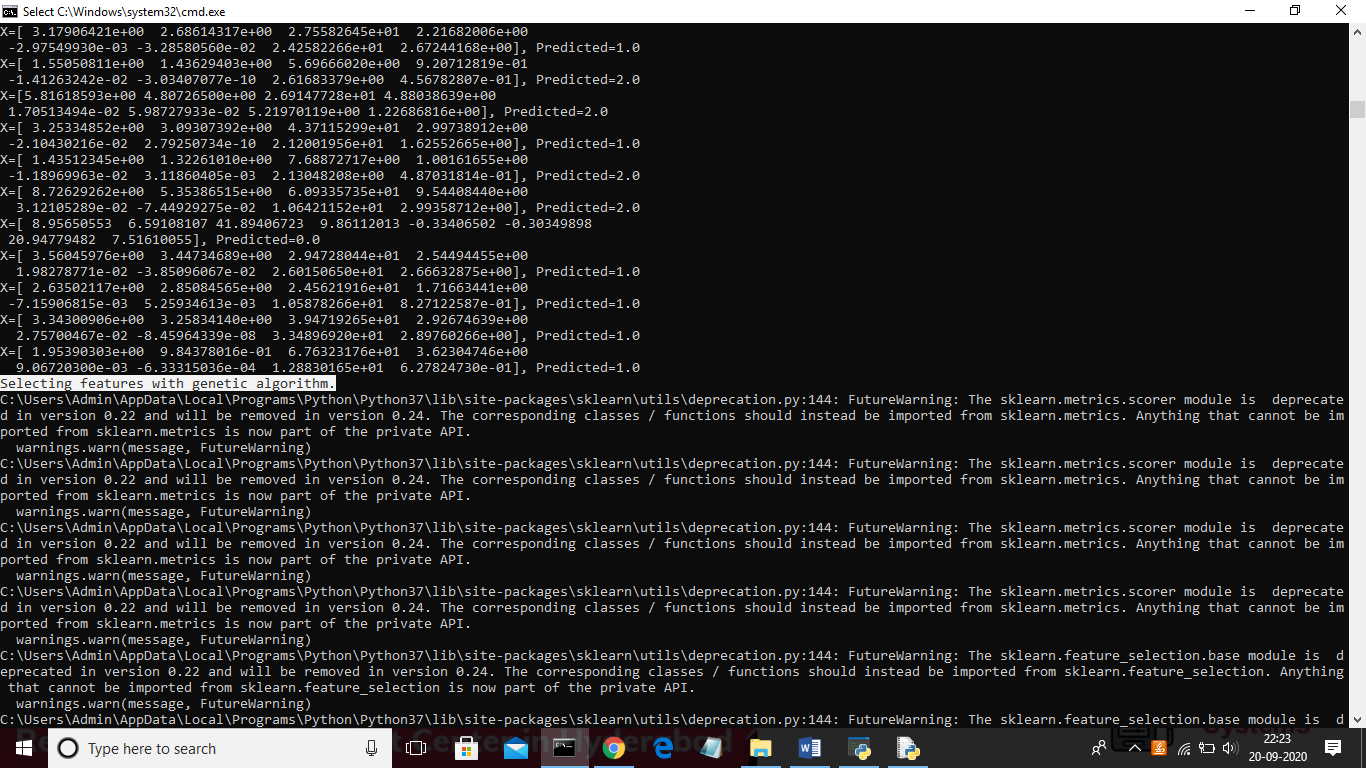
In above screen we calculated random forest accuracy, precision, recall and fmeasure and random forest got 67% prediction accuracy. Now click on ‘Run MLP Algorithm’ button to train MLP model and to calculate its accuracy



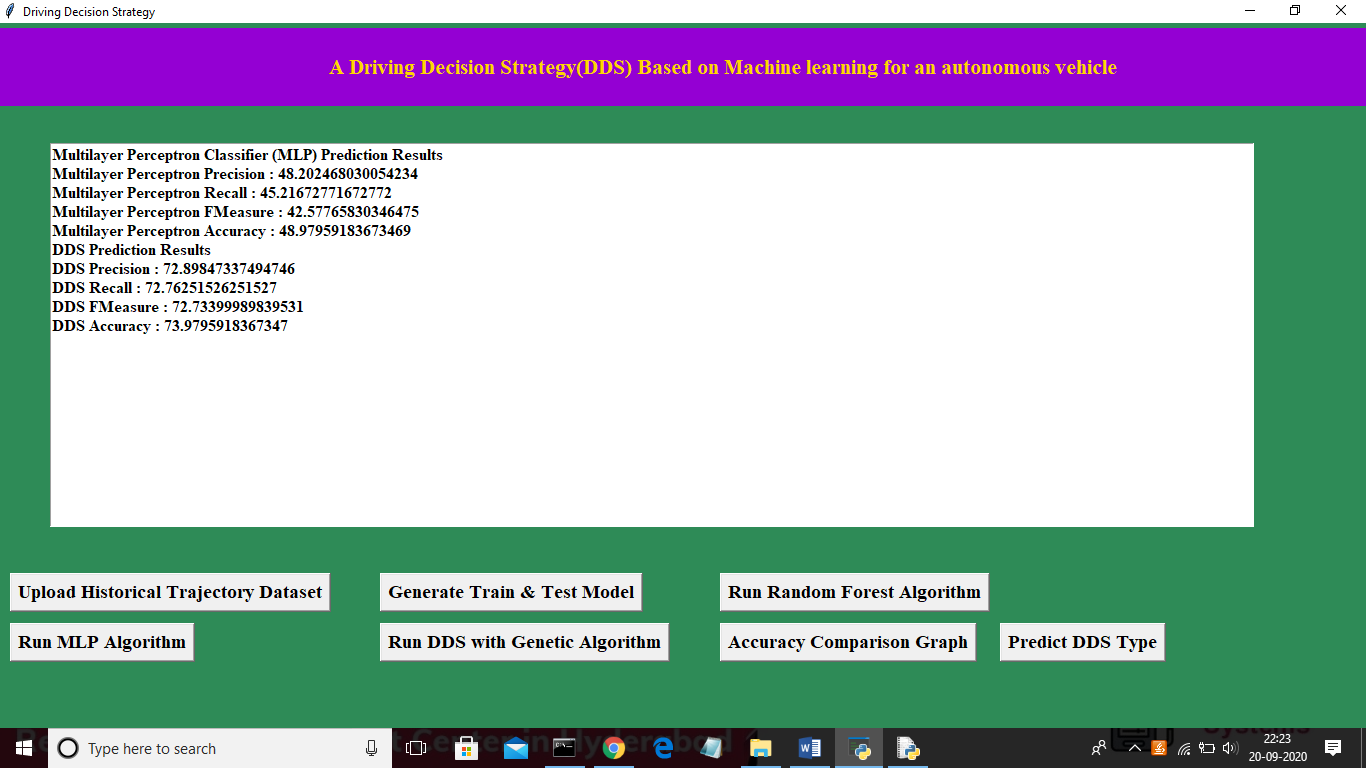
In above screen MLP got 48% prediction accuracy and in below screen we can see genetic algorithm code used for building propose DDS algorithm



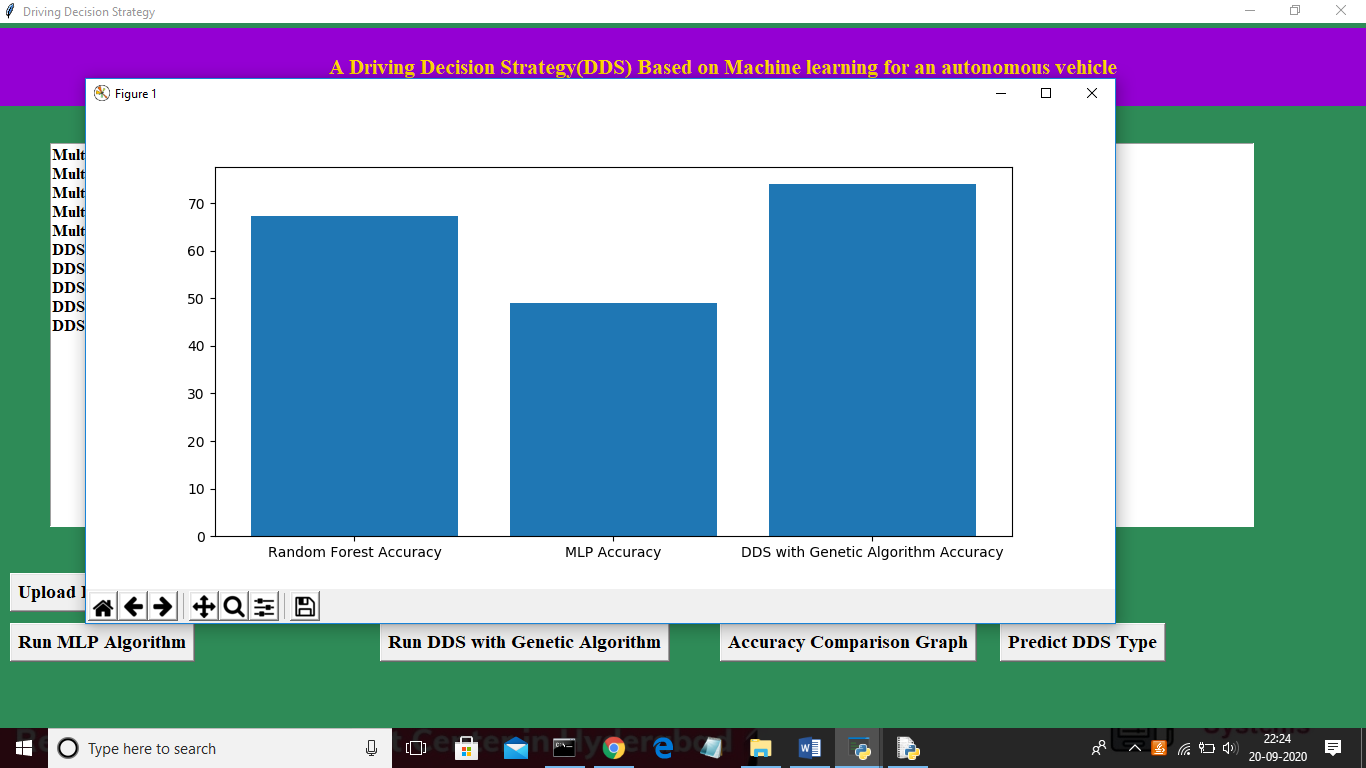
In above screen we can see genetic algorithm code used in DDS algorithm and now click on ‘Run DDS with Genetic Algorithm’ button to train DDS and to calculate its prediction accuracy



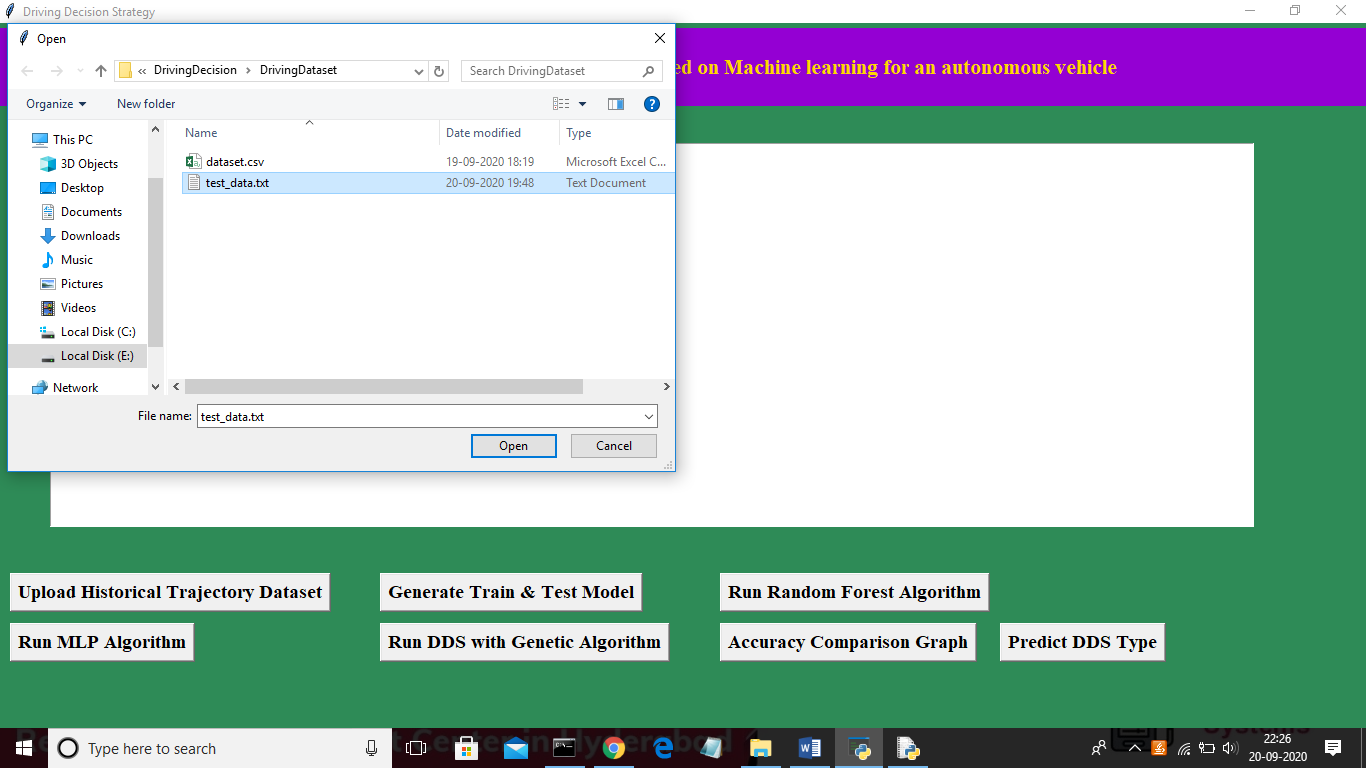
In above black console genetic algorithm starts optimal feature selection



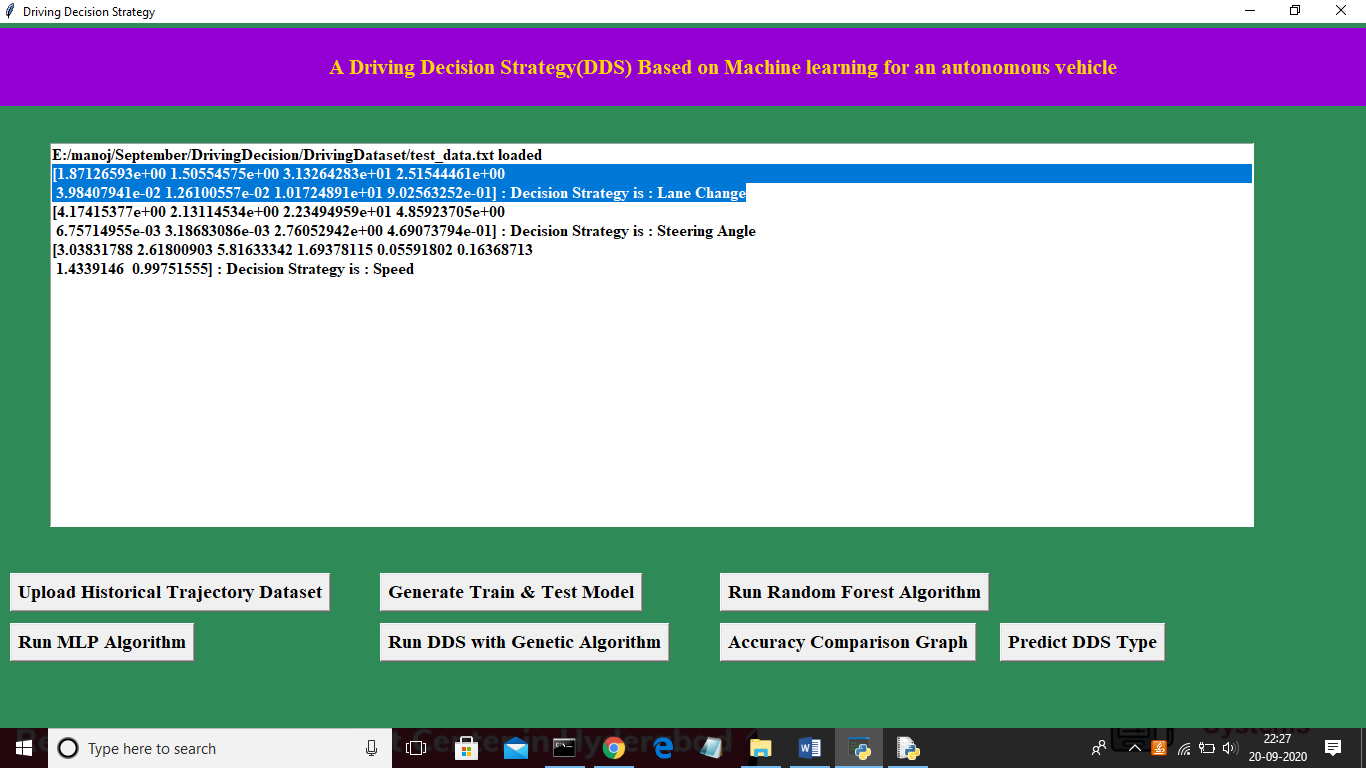
In above screen propose DDS algorithm got 73% prediction accuracy and now click on ‘Accuracy Comparison Graph’ button to get below graph



In above graph x-axis represents algorithm name and y-axis represents accuracy of those algorithms and from above graph we can conclude that DDS is performing well compare to other two algorithms. Now click on ‘Predict DDS Type’ button to predict test data



In above screen uploading ‘test\_data.txt’ file and click on ‘Open’ button to predict driving decision



In above screen in selected first record we can see decision is Lane Change and for second record values we got decision as ‘steering angle’ and for third test record we got predicted value as vehicle is in speed mode.

**Extension Outcomes:**

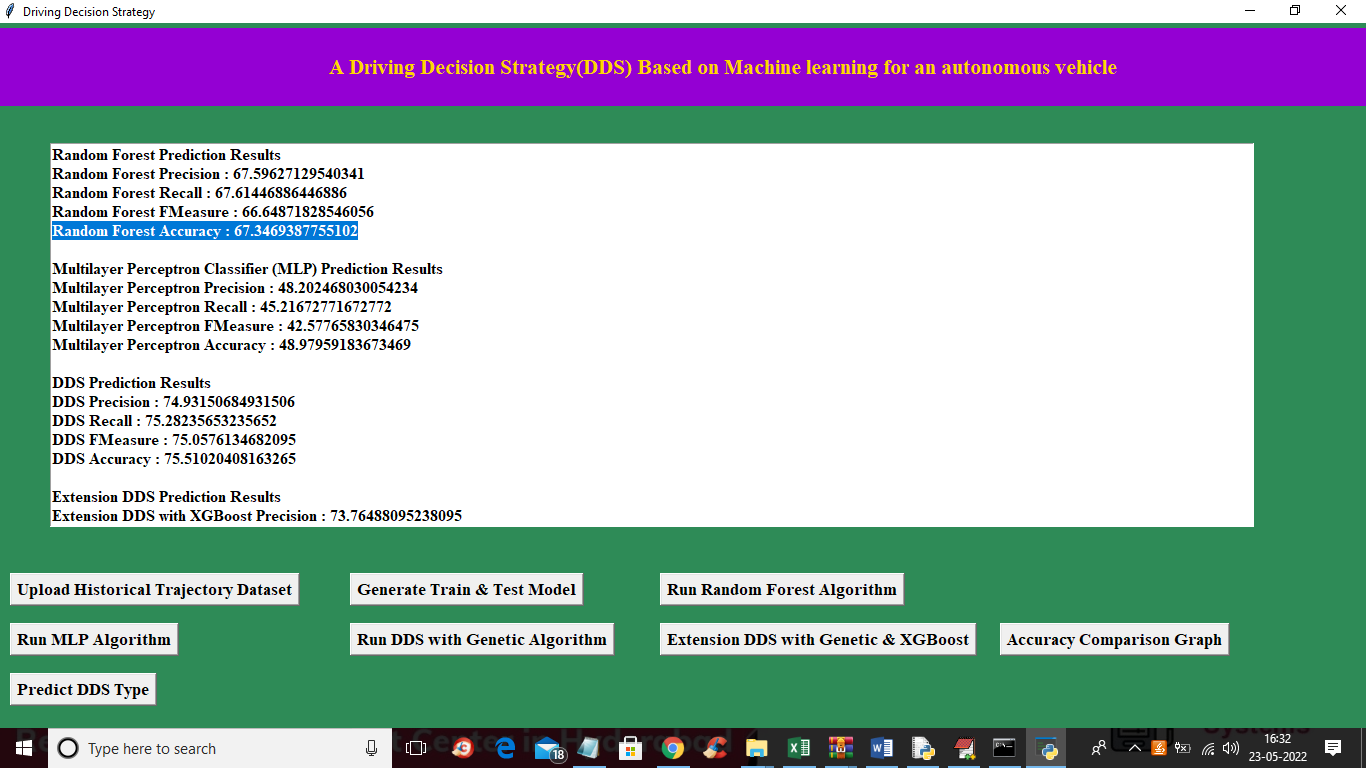
In propose DDS algorithm we have used Random Forest with Genetic Algorithm and we are getting prediction accuracy up to 75% and to further enhance accuracy we are upgrading propose DDS with Genetic and XGBOOST algorithm and this extension XGBOOST algorithm giving much better accuracy compare to propose algorithm

Below is the advantage of XGBOOST algorithm

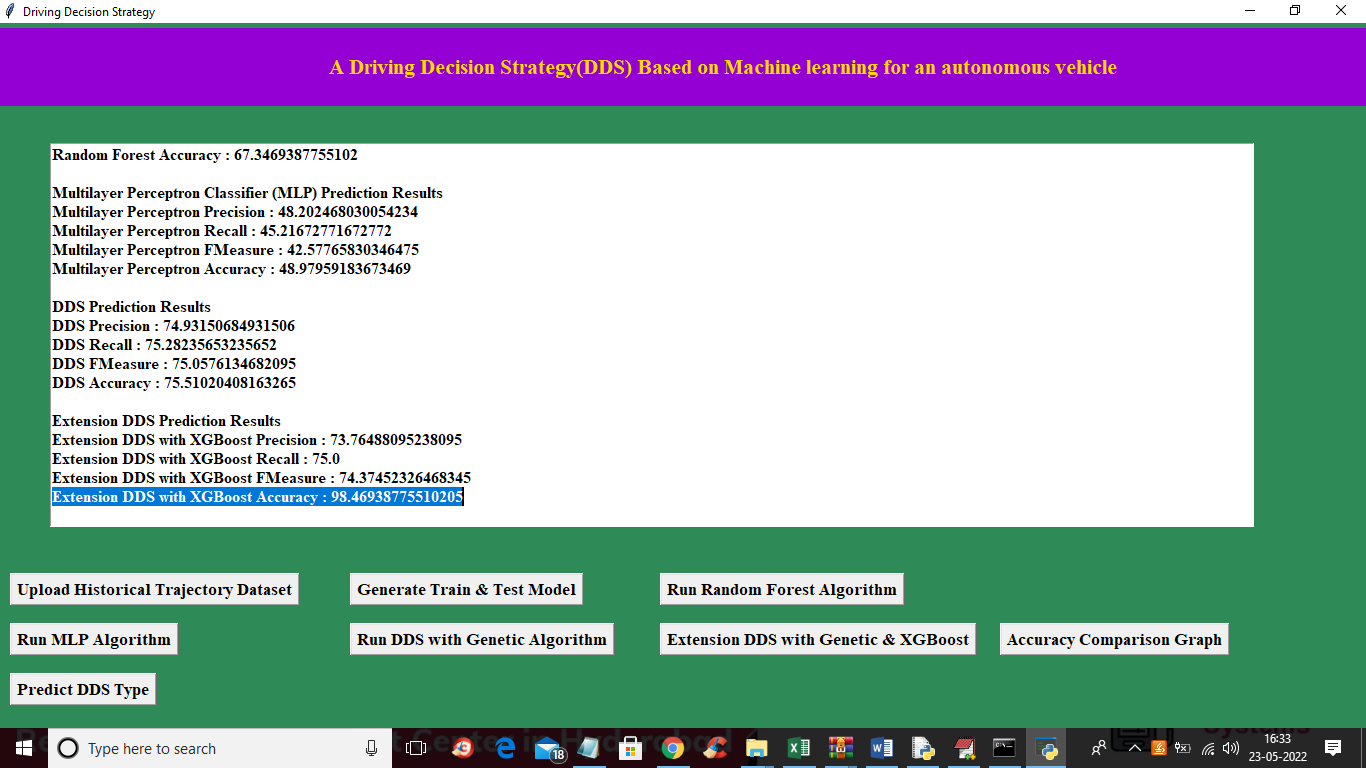
XGBoost is well known to provide better solutions than other machine learning algorithms. What makes XGBoost so popular?

1. Speed and performance: Originally written in C++, it is comparatively faster than other ensemble classifiers.
2. Core algorithm is parallelizable: Because the core XGBoost algorithm is parallelizable it can harness the power of multi-core computers. It is also parallelizable onto GPU’s and across networks of computers making it feasible to train on very large datasets as well.
3. Consistently outperforms other algorithm methods: It has shown better performance on a variety of machine learning benchmark datasets.
4. Wide variety of tuning parameters: XGBoost internally has parameters for cross-validation, regularization, user-defined objective functions, missing values, tree parameters, scikit-learn compatible API etc.

XGBoost (Extreme Gradient Boosting) belongs to a family of boosting algorithms and uses the gradient boosting (GBM) framework at its core. It is an optimized distributed gradient boosting library.



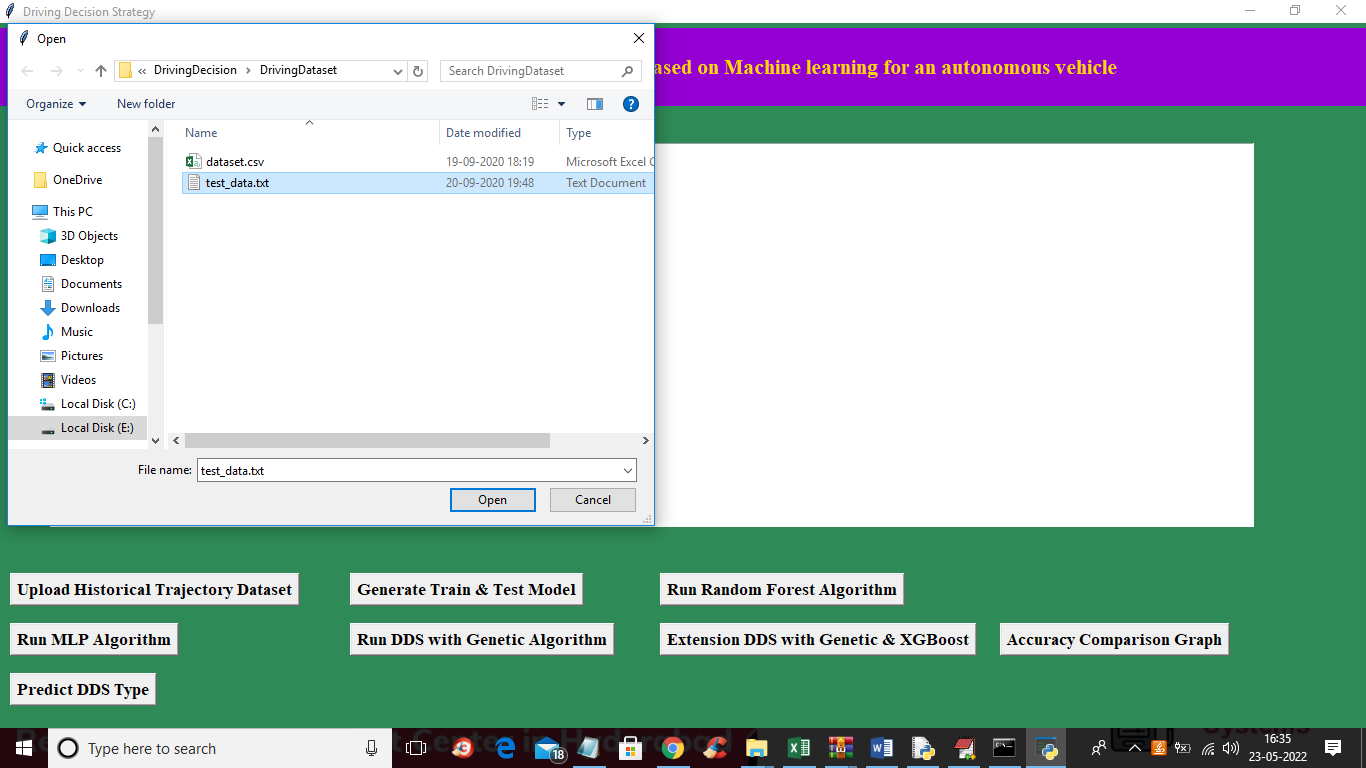
In above screen with Random Forest we got 67% accuracy and with Multilayer perceptron we got accuracy as 48% and with propose DDS we got accuracy as 75 and in below screen we can see accuracy of extension XGBOOST



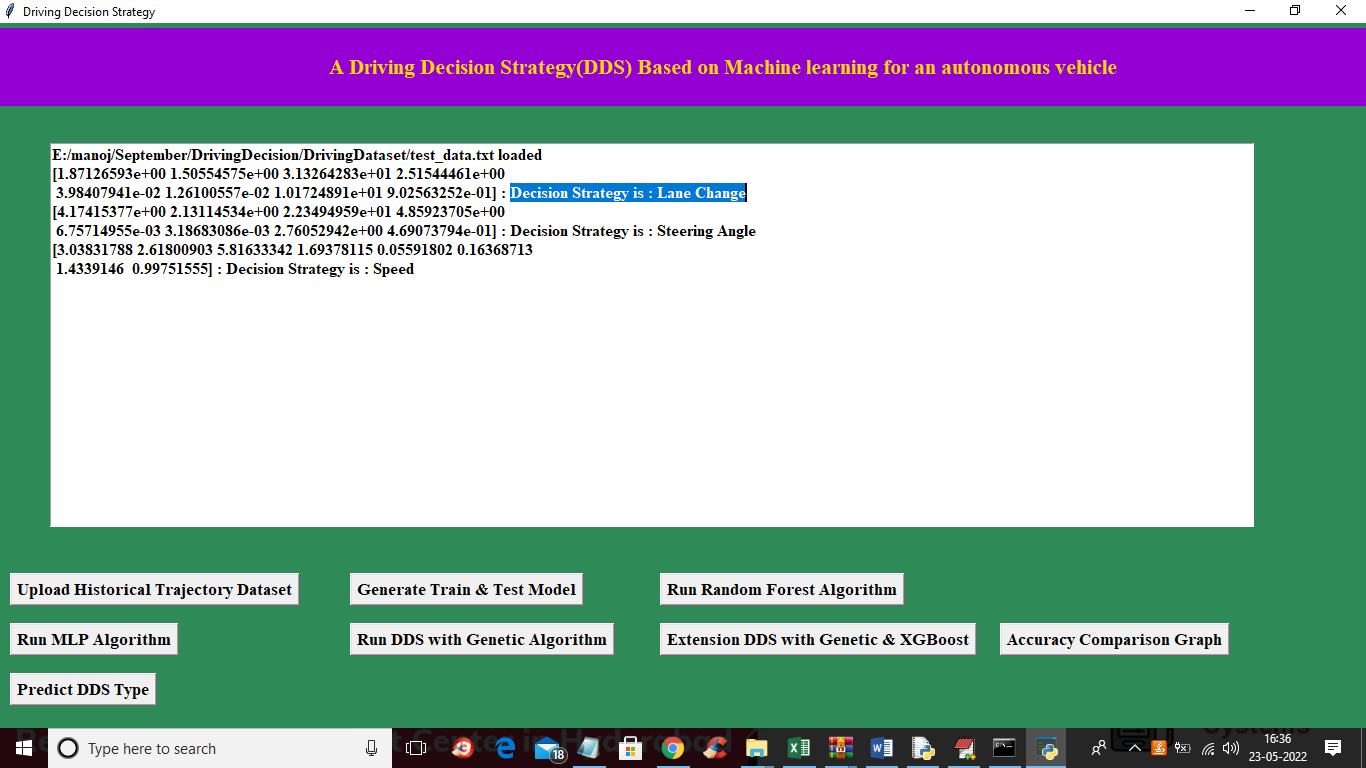
In above screen in blue colour text we can see with extension XGBOOST we got 98% accuracy and below is the comparison graph



In above graph x-axis represents algorithm names and y-axis represents accuracy and in all algorithms extension XGBOOST has got high accuracy. Below is the detection output



In above screen uploading test data and below is the prediction output



In blue colour selected text we can see prediction output as Lane Changeor SPEED

**CONCLUSION**

In propose DDS algorithm we have used Random Forest with Genetic Algorithm and we are getting prediction accuracy up to 75% and to further enhance accuracy we are upgrading propose DDS with Genetic and XGBOOST algorithm and this extension XGBOOST algorithm giving much better accuracy compare to propose algorithm

**Bibliography**

[1] Y.N. Jeong, S.R.Son, E.H. Jeong and B.K. Lee, “An Integrated SelfDiagnosis System for an Autonomous Vehicle Based on an IoT Gateway and Deep Learning, ” Applied Sciences, vol. 8, no. 7, july 2018.

[2] Yukiko Kenmochi, Lilian Buzer, Akihiro Sugimoto, Ikuko Shimizu, “Discrete plane segmentation and estimation from a point cloud using local geometric patterns, ” International Journal of Automation and Computing, Vol. 5, No. 3, pp.246-256, 2008.

[3] Ning Ye, Yingya Zhang, Ruchuan Wang, Reza Malekian, “Vehicle trajectory prediction based on Hidden Markov Model, ” The KSII Transactions on Internet and Information Systems, Vol. 10, No. 7, 2017.

[4] Li-Jie Zhao, Tian-You Chai, De-Cheng Yuan, “Selective ensemble extreme learning machine modeling of effluent quality in wastewater treatment plants, ” International Journal of Automation and Computing, Vol.9, No.6, 2012