INSTITUTE FOR ADVANCED COMPUTING AND SOFTWARE DEVELOPMENT

AKURDI, PUNE

Documentation On

"Vehicle Accident Prediction model based on Weather Factors"

PG-DBDA September 2020

Submitted By:

Group No:21

Snehal Padekar -1532

Pooja Maske -1534

Mr. Prashant Karhale
Centre Coordinator

Mr. Akshay Tilekar
Project Guide

Table of Contents

ACKNOWLEDGEMENT

It is our privilege to express our sincerest regards to our project guide **Mr. Akshay Tilekar** for their valuable inputs, able guidance, whole-hearted cooperation, constructive and criticism, encouragement throughout the project work.

We deeply express our sincere thanks to **Mr. Prashant Karhale**, Centre Co-ordinator IACSD CDAC, Akurdi for encouraging and supporting. We are also thankful to our respected Internal Project Guides Mr. Rahul, and Mr. Manish. We take this opportunity to thanks all our staff members who have directly or indirectly helped in our project work.

We pay our respect and love to our parents and all other family members for their love and Encouragement throughout the project work. Last but not least we express our thanks to our friends for their cooperation and support.

1532-Snehal Padekar

1534-Pooja Maske

ABSTRACT

In the fast pace of life, people usually tends to drive recklessly. Everyone is in hurry to reach their respective location, which can lead to towards fatal accidents. Sometimes accidents occur not only because of drivers fault or negligence but also due to bad weather or due to bad conditions of the roads and at times the accidents are fatal and loss of life and property is just too much.

The purpose of this study is to predict how various weather conditions lead to accidents which will benefit the insurance company and the transport authority as well as the citizens. In this project we are going to predict whether the accident will happen or not? , using various parameters like speed of the vehicle Precipitation intensity, wind direction, condition of the roads with respect to weather, lighting conditions based on time of the day.

INTRODUCTION

Traffic accidents have an adverse effect both on an individual and societal level resulting in costs due to person injuries and property damage, increased travel times and emissions due to congestion. The studies conclude that accident risks are significantly heightened during snowy and icy road conditions. The accident risk was more than four times higher for snowy or icy road surface compared to bare road surface. For slushy road surfaces, the corresponding risk for fatal accidents was five fold. In the study, accident risk for a specific condition was defined as the number of accidents per vehicle mileage occurring in the conditions.

In this proposed system, we are going to determine whether the accident will happen or not? based upon various weather conditions will affect. Dataset mainly contains the vehicle id, speed of the vehicle, weather condition with respect to weather, lightening conditions, perception type, Intensity, wind speed direction. Proposed system generate a whether the accident will happen or not depend on the weather conditions and speed of the vehicle. The system uses a dataset and generates the output depending on it. In previous Research only the driving style of the person is predicted and how it leads to bad outcomes. However in this model we are going to predict how weather conditions, speed of the vehicles may leads to accident and also how various atmosphere conditions may increase the chances of accidents.

PROBLEM STATEMENT

In the existing system, there is no optimum pattern analysis done for the vehicle accident will happen or not? based on the weather condition and speed factors. But, this approach does not give an accurate analysis, as all possible combinations of patterns are not considered. This will make the pattern analysis faulty, and the actions taken on the basis of this analysis will not be correct too, so we need to propose a solution which will improve the accuracy of the vehicle accident based on the input dataset.

MODULES USED IN THE PROJECT

- Collect the input dataset for the driving patterns.
- Remove any outlier from the dataset.
- Apply Genetic Algorithm on the dataset to find the patterns of driving.
- Result analysis and comparison with previous method.

- Collect The Input Dataset for the Driving Patterns

In this module we collect the information from the dataset of vehicle accident, it gives information about the speed. Vehicle's ID, weather conditions etc. this information are used to derives the pattern of weather it lead to accident or not?

- Remove Any Outlier from the Dataset

In this module, an outlier will get removed from dataset. An outlier may be due to variability in the measurement or it may indicate experimental error; the latter are sometimes excluded from the data set

- Apply Genetic Algorithm on the Dataset to Find the Patterns of accident will happen or not?

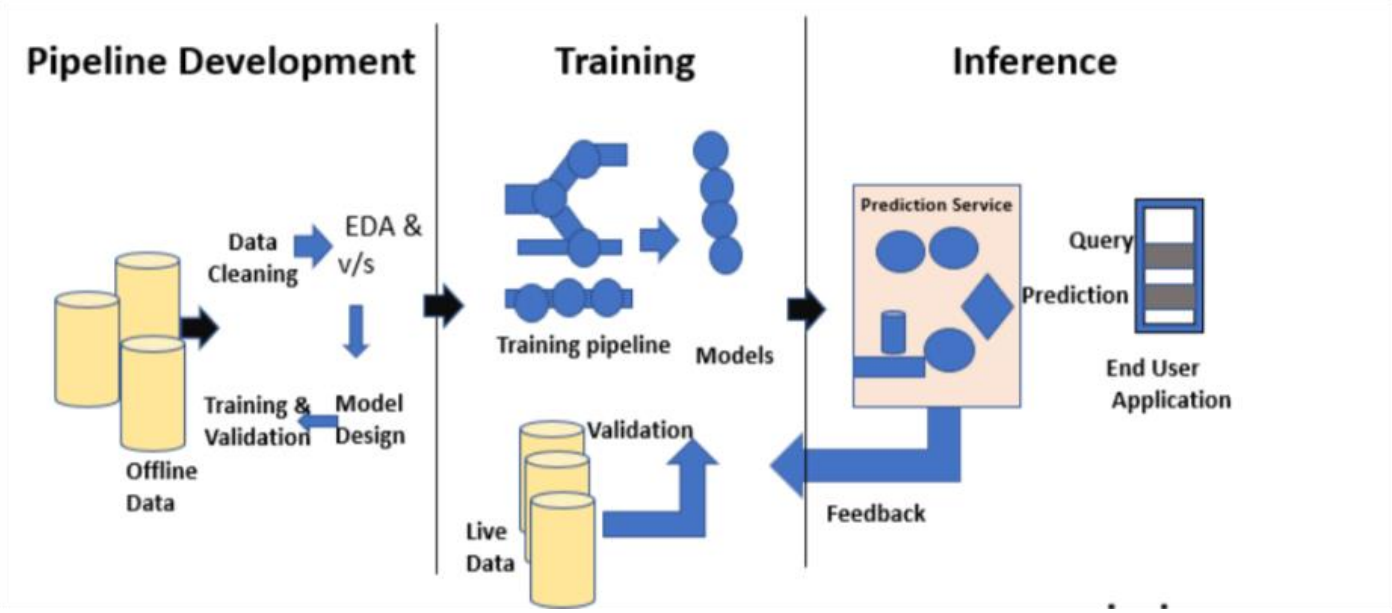
In this module, we apply the genetic algorithm on dataset to find the pattern of driving and generate the solution for the driving pattern. Also find the fitness of each solution

- Result Analysis and Comparison with other model

Here we analyse and compare the result with other models

Machine Learning Life cycle

Machine Learning Life Cycle is defined as a cyclical process which involves three-phase process (Pipeline development, Training phase, and Inference phase) acquired by the data scientist and the data engineers to develop, train and serve the models using the huge amount of data that are involved in various applications so that the organization can take advantage of artificial intelligence and machine learning algorithms to derive a practical business value.



The first step in the machine learning lifecycle consists of transforming raw data into a cleaned dataset, that dataset is often shared and reused. If an analyst or a data scientist who encounter issues in the received data, they need to access the original data and transformation scripts. There are a variety of reasons that we may want to return to earlier versions of our models and data. For example, finding the earlier best version may require searching through many alternative versions as models unavoidably degrade in their predictive power. There are many reasons for this degradation, like a shift in the distribution of data that can result in a rapid decline in predictive power as compensation for errors. Diagnosing this decline may require comparing training data with live data, retraining the model, revisiting earlier design decisions or even redesigning the model.

Steps Involved In Machine Learning Lifecycle

1. Building the machine learning model

This step decides the type of the model based on the application. It also finds that the application of the model in the model learning stage so that they can be designed properly according to the need of an intended application. A variety of machine learning models are available, such as the supervised model, unsupervised model, classification models, regression models, clustering models, and reinforcement learning models.

2. Data Preparation

A variety of data can be used as input for machine learning purposes. This data can come from a number of sources, such as a business, pharmaceutical companies, IoT devices, enterprises, banks, hospitals etc. Large volumes of data are provided at the learning stage of the machine since as the number of data increases it aligns towards yielding desired results. This output data can be used for analysis or fed as input into other machine learning applications or systems for which it will act as a seed.

3. Model Training

This stage is concerned with creating a model from the data given to it. At this stage, a part of the training data is used to find model parameters such as the coefficients of a polynomial or weights of in machine learning which helps to minimize the error for the given data set. The remaining data are then used to test the model. These two steps are generally repeated a number of times in order to improve the performance of the model.

4. Parameter Selection

It involves the selection of the parameters associated with the training which are also called the hyperparameters. These parameters control the effectiveness of the training process and hence, ultimately the performance of the model depends on this. They are very much crucial for the successful production of the machine learning model.

5. Model Verification

The input of this stage is the trained model produced by the model learning stage and the output is a verified model that provides sufficient information to allow users to determine whether the model is suitable for its intended application. Thus, this stage of the machine learning lifecycle is concerned with the fact that a model is working properly when treated with inputs that are unseen.

6. Deploy the machine learning model

In this stage of the Machine learning lifecycle, we apply to integrate machine learning models into processes and applications. The ultimate aim of this stage is the proper functionality of the model after deployment. The models should be deployed in such a way that they can be used for inference as well as they should be updated regularly.

7. Monitoring

It involves the inclusion of safety measures for the assurance of proper operation of the model during its life span. In order to make this happen proper management and updating are required.

OVERALL DESCRIPTION

- The aim of our project is to make use of pandas, matplotlib, & seaborn libraries from python to extract the meaningful insights from the data and classifier models like Random forest, Logistic Regression, Ridge Classifier, Decision Tree, Adaboost, & scikit-learn libraries for machine learning.
- To learn the hyper tune parameters.
- To predict whether the vehicle accident will happen or not with that machine learning algorithm which gives maximum accuracy.

Attributes in the dataset:

- **DateTime**
- **ID**
- **Intensity**
- **Wind Direction**
- **Wind speed**
- **Time gap**
- **Light Condition**
- **Road Accident will not happen (0) and accident will happen (1)**
- **Weight**
- **Condition of road**
- **Humidity**
- **Wind Intensity**
- **Percep type**

Data Processing and Data Visualisation

Data Processing:

Sample dataset:-

ID	weather_i	intensity	humidity	wind_direc	wind_spee	light_cond	percep_ty	ID	DateTime	speed	ID_prec	speed_pre	weight	leng_prec	time_gapr	condition_	lane_road
DR_24526	7	None	95	146	1	daylight	clear	DR_24526	21-03-2012 09:14	81	57227	87	16986	941	94	Dry	1
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:15	88	57229	81	1708	551	11	Dry	1
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:15	88	57230	88	22892	1698	4	Dry	1
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:15	84	57228	89	1945	544	127	Dry	0
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:15	89	57231	88	13787	1893	42	Dry	1
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:16	91	57232	84	2637	586	30	Dry	0
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:16	92	57234	91	10243	1265	4	Dry	0
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:16	90	57235	92	10915	1247	2	Dry	0
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:17	92	57236	90	1316	456	47	Dry	0
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:17	84	57237	92	2489	517	31	Dry	0
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:18	91	57238	84	1555	524	47	Dry	0
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:18	85	57239	91	1408	459	9	Dry	0
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:19	91	57240	85	2311	531	17	Dry	0
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:20	81	57241	91	1747	522	75	Dry	0
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:20	85	57242	81	17122	1953	24	Dry	0
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:21	87	57243	85	11898	1237	30	Dry	0
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:23	102	57244	87	12375	1228	160	Dry	0
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:24	79	57233	89	911	487	531	Dry	1
DR_24526	7	None	95	124	0	daylight	clear	DR_24526	21-03-2012 09:24	82	57246	79	26187	1705	4	Dry	1
DR_30052	9	None	95	118	6	daylight	clear	DR_30052	21-03-2012 09:30	79	57249	90	2018	538	36	Dry	1
DR_30052	9	None	95	118	6	daylight	clear	DR_30052	21-03-2012 09:31	81	57250	79	374	428	40	Dry	1
DR_30052	9	None	95	118	6	daylight	clear	DR_30052	21-03-2012 09:32	86	57245	102	1842	545	490	Dry	0
DR_30052	9	None	95	118	6	daylight	clear	DR_30052	21-03-2012 09:34	83	57251	81	1176	518	198	Dry	1
DR_30052	9	None	95	118	6	daylight	clear	DR_30052	21-03-2012 09:34	81	57253	83	30871	1680	22	Dry	1
DR_30052	1	None	96	191	6	daylight	clear	DR_30052	21-03-2012 09:37	88	57252	86	2135	1112	313	Dry	0
DR_30052	1	None	96	191	6	daylight	clear	DR_30052	21-03-2012 09:38	88	57254	81	1429	543	219	Dry	1
DR_30052	1	None	96	191	6	daylight	clear	DR_30052	21-03-2012 09:38	91	57255	88	14764	1689	88	Dry	0
DR_30052	1	None	96	191	6	daylight	clear	DR_30052	21-03-2012 09:38	71	57256	88	17361	1929	23	Dry	1
DR_30052	1	None	96	191	6	daylight	clear	DR_30052	21-03-2012 09:39	78	57257	91	2105	637	12	Dry	0
DR_30052	1	None	96	191	6	daylight	clear	DR_30052	21-03-2012 09:40	79	57259	78	1682	508	81	Dry	0

Here we can see that the date variable is in string format, so while loading the data we have to take care of the date variable, it should be in python date time object.

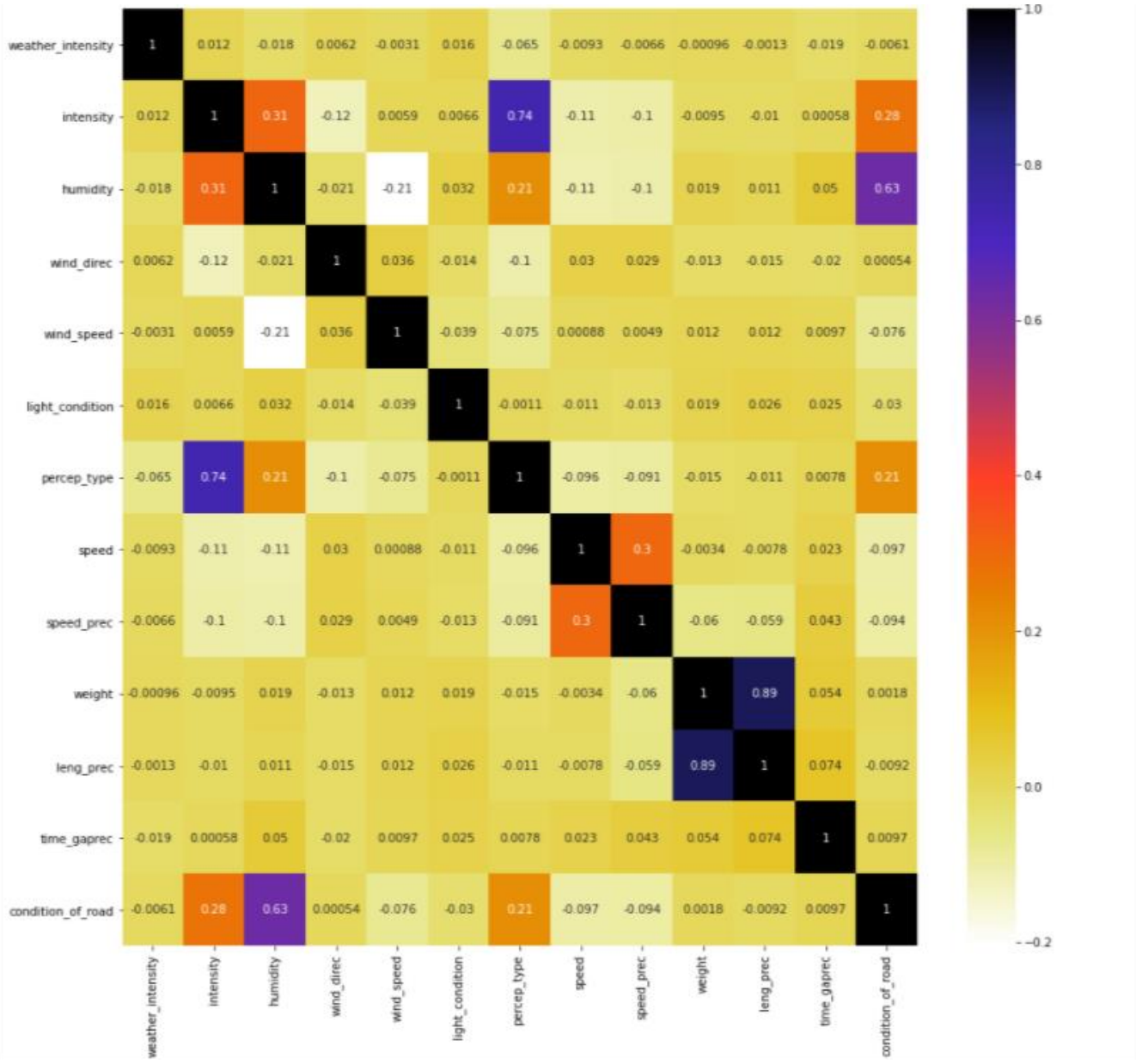
Descriptive Statistics:-

	weather_intensity	intensity	humidity	wind_direc	wind_speed	light_condition	percep_type	speed
count	162566.000000	162566.000000	162566.000000	162566.000000	162566.000000	162566.000000	162566.000000	162566.000000
mean	4.607999	1.057724	60.457845	182.429173	4.125161	0.819070	0.106117	83.455483
std	3.232133	0.252619	18.229576	88.348203	3.047835	0.614165	0.409859	9.375122
min	-13.000000	1.000000	16.000000	6.000000	0.000000	0.000000	0.000000	8.000000
25%	2.000000	1.000000	46.000000	152.000000	1.000000	0.000000	0.000000	78.000000
50%	5.000000	1.000000	57.000000	180.000000	4.000000	1.000000	0.000000	83.000000
75%	7.000000	1.000000	76.000000	208.000000	7.000000	1.000000	0.000000	88.000000
max	24.000000	4.000000	97.000000	360.000000	17.000000	2.000000	2.000000	161.000000

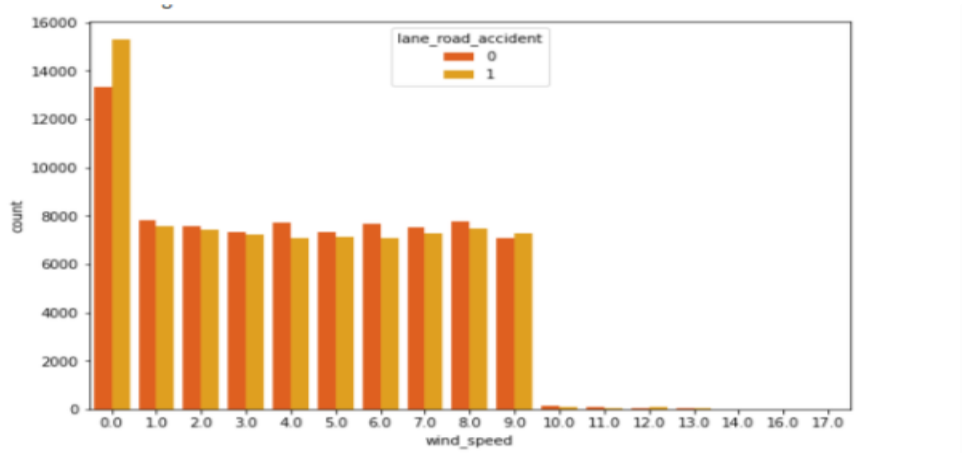
speed_prec	weight	leng_prec	time_gaprec	condition_of_road	lane_road_accident
162566.000000	162566.000000	162566.000000	162566.000000	162566.000000	162566.000000
83.458817	5017.562793	790.775285	103.764656	0.773169	0.499151
9.373120	7399.315434	481.944297	176.318944	1.277600	0.500001
0.000000	3.000000	102.000000	1.000000	0.000000	0.000000
78.000000	1502.000000	527.000000	6.000000	0.000000	0.000000
83.000000	1862.000000	560.000000	42.000000	0.000000	0.000000
88.000000	2669.000000	701.000000	123.000000	2.000000	1.000000
161.000000	69548.000000	2981.000000	1797.000000	3.000000	1.000000

Data Visualisation:-

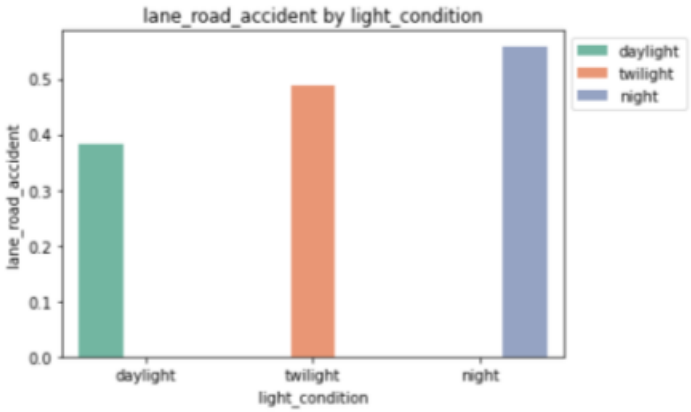
1) Correlation Heat map:-



2) The graph shows road accident due to wind speed where chances of getting accident is high with count of 16000

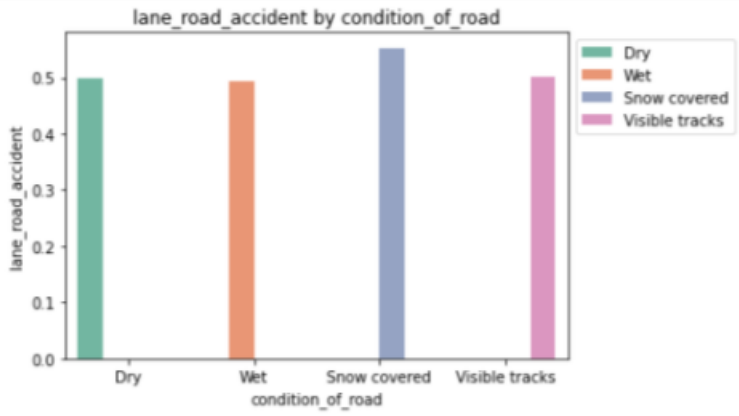


3)



The above graph is light_condition vs road_accident.By observing the chances of getting accident is high during night and in daylight there is low chances of getting accident

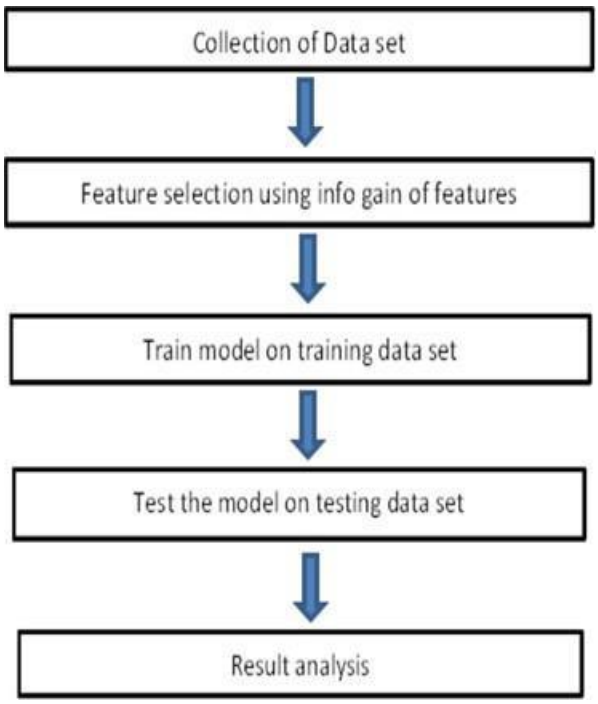
4)



The above graph is the 'condition_of_road' with repsect to 'road_accident'. By considering 'Snow_covered' condition the chances of accident is high as compared others.

System Design

Flowchart of the System:



The above flowchart describes the working flow of the project. We first built the UI of the our project. Then research was done to select the best algorithm from the list of algorithms. The algorithm was later implemented in python and deployed.

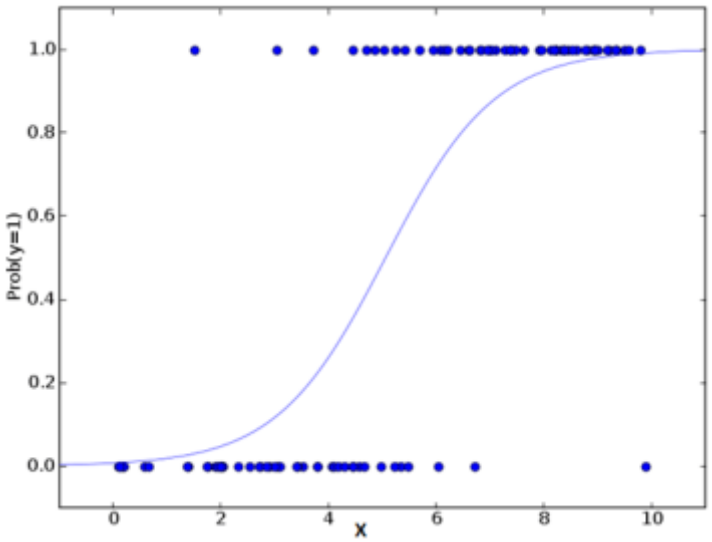
Model Building Algorithms

Logistic Regression

Logistic Regression is a classification algorithm. It is used to predict a binary outcome (1 / 0, Yes / No, True / False) given a set of independent variables. To represent binary/categorical outcome, we use dummy variables. You can also think of logistic regression as a special case of linear regression when the outcome variable is categorical, where we are using log of odds as dependent variable. In simple words, it predicts the probability of occurrence of an event by fitting data to a logit function.

The fundamental equation of generalized linear model is:

$$g(E(y)) = \alpha + \beta x1 + \gamma x2$$



Advantages:-

- Logistic regression is easier to implement, interpret, and very efficient to train.

Disadvantages:

- If the number of observations is lesser than the number of features, Logistic Regression should not be used, otherwise, it may lead to over fitting.
- It constructs linear boundaries.

Logistic Regression

```
[ ]  from sklearn.metrics import accuracy_score
     from sklearn.linear_model import LogisticRegression
     from sklearn.metrics import roc_auc_score
     from sklearn.metrics import roc_curve
     log_model = LogisticRegression()
     #fit the model
     log_model.fit(x_train, y_train)
     y_t1=log_model.predict(x_test)
     y_t2=log_model.predict(x_train)
     #evaluate model
     score=accuracy_score(y_test,y_t1)
     score1=accuracy_score(y_train,y_t2)
     print(score)
     print(score1)
```

```
0.6340485212341912
0.6348791050162397
```

ROC CURVE

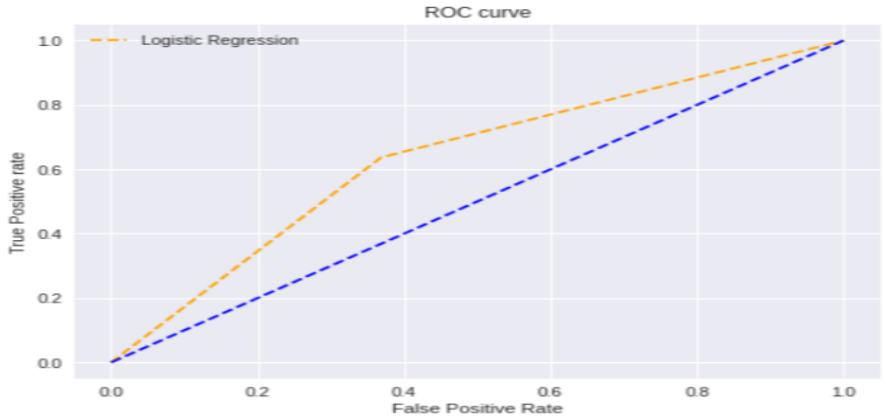
```
[148] from sklearn.metrics import roc_curve

      # roc curve for models
      fpr1, tpr1, thresh1 = roc_curve(y_test, y_t1, pos_label=1)
      #fpr2, tpr2, thresh2 = roc_curve(y_test, pred_prob2[:,1], pos_label=1)

      # roc curve for tpr = fpr
      random_probs = [0 for i in range(len(y_test))]
      p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)
```

```
[149] import matplotlib.pyplot as plt
      plt.style.use('seaborn')

      # plot roc curves
      plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='Logistic Regression')
      plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
      # title
      plt.title('ROC curve')
      # x label
      plt.xlabel('False Positive Rate')
      # y label
      plt.ylabel('True Positive rate')
      plt.legend(loc='best')
      plt.show();
```



After tuning

After Tunning

```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import RepeatedStratifiedKFold
from sklearn.model_selection import GridSearchCV

solvers=['newton-cg','lbfgs','liblinear']
penalty=['l2']
c_values=[100,10,1.0,0.1,0.01,1000]
grid=dict(solver=solvers,penalty=penalty,C=c_values)
model_params={
    'logistic':{
        'model':LogisticRegression(),
        'params':grid
    }

}

scores=[]

for model_name,mp in model_params.items():
  clf1=GridSearchCV(mp['model'],mp['params'],cv=10,return_train_score=False)
  clf1.fit(x_test,y_test)
  scores.append({
      'model':model_name,
      'best_score':clf1.best_score_,
      'best_params':clf1.best_params_
  })
logistic_df=pd.DataFrame(scores,columns=['model','best_score','best_params'])
logistic_df
```

	model	best_score	best_params
0	logistic	0.634737	{'C': 1000, 'penalty': 'l2', 'solver': 'newton...

After Tuning Accuracy is 63.47%

RIDGE CLASSIFIER

The Ridge Classifier, based on Ridge regression method, converts the label data into [-1, 1] and solves the problem with regression method. The highest value in prediction is accepted as a target class and for multiclass data multi-output regression is applied.

Advantages:-

- Avoids over fitting a model.
- They do not require unbiased estimators.
- They add just enough bias to make the estimates reasonably reliable approximations to true population values.
- They still perform well in cases of a large multivariate data with the number of predictors (p) larger than the number of observations (n).
- The ridge estimator is preferably good at improving the least-squares estimate when there is multicollinearity.

Disadvantages:-

- They include all the predictors in the final model.
- They are unable to perform feature selection.
- They shrink the coefficients towards zero.
- They trade the variance for bias.

Ridge Classifier

```
[ ] from sklearn.linear_model import RidgeClassifier
    rc = RidgeClassifier(alpha=1, normalize=True)
    rc.fit(x_train, y_train)
    rc_pred = rc.predict(x_test)
```

```
[ ] from sklearn.metrics import accuracy_score,confusion_matrix
    import sklearn.metrics as metrics
    print(confusion_matrix(y_test,rc_pred))
    print(metrics.accuracy_score(y_test,rc_pred))
    print(metrics.recall_score(y_test,rc_pred))
    print(metrics.precision_score(y_test,rc_pred))
    print(metrics.f1_score(y_test,rc_pred))
```

```
[[12953  7487]
 [ 7356 12846]]
0.6347866738841592
0.6358776358776359
0.6317808488663749
0.6338226224250648
```

After tuning

```
[ ] from sklearn.linear_model import RidgeClassifier
    rc = RidgeClassifier(alpha=1,normalize=True)
    rc.fit(x_train, y_train)
```

```
RidgeClassifier(alpha=1, class_weight=None, copy_X=True, fit_intercept=True,
                max_iter=None, normalize=True, random_state=None, solver='auto',
                tol=0.001)
```

```
[ ] param_rc={

        'max_iter':range(5,20),
        'alpha':[0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1.0,1.5],
        'normalize':[True,False],
        #'tol':[1e-2,1e-4,1e-6,1e-8,1e-10]
        'solver':['svd','sparse_cg']


        }
    rc_grid=GridSearchCV(estimator=rc,param_grid=param_rc,cv=5,verbose=1)
    rc_grid.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 660 candidates, totalling 3300 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 3300 out of 3300 | elapsed:  6.9min finished
GridSearchCV(cv=5, error_score=nan,
             estimator=RidgeClassifier(alpha=1, class_weight=None, copy_X=True,
                                       fit_intercept=True, max_iter=None,
                                       normalize=True, random_state=None,
                                       solver='auto', tol=0.001),
             iid='deprecated', n_jobs=None,
             param_grid={'alpha': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9,
                                   1.0, 1.5],
                         'max_iter': range(5, 20), 'normalize': [True, False],
                         'solver': ['svd', 'sparse_cg']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=1)
```

```
[ ] from sklearn.linear_model import RidgeClassifier
    rct = RidgeClassifier(alpha=1,max_iter=20,solver='sparse_cg',random_state=9)
    rct.fit(x_train, y_train)
    rct_pred = rct.predict(x_test)
    print(confusion_matrix(y_test,rct_pred))
    print(metrics.accuracy_score(y_test,rct_pred))
    print(metrics.recall_score(y_test,rct_pred))
    print(metrics.precision_score(y_test,rct_pred))
    print(metrics.f1_score(y_test,rct_pred))
```

```
[[12902  7523]
 [ 7365 12852]]
0.6336794449092072
0.6357026265024485
0.6307730061349693
0.6332282223098148
```

After Tuning Accuracy is 63.57%

RANDOM FOREST

The random forest algorithm is based on supervised learning. It can be used for both regression and classification problems. As the name suggests Random Forest can be viewed as a collection of multiple decision trees algorithm with random sampling. This algorithm is made to eradicate the shortcomings of the Decision tree algorithm.

Random forest is a combination of bagging idea and random selection of features. The idea is to make the prediction precise by taking average or mode of the output of multiple decision trees. The greater the number of decision trees is considered the more precise output will be.

Advantages:-

- Random forest is considered as a highly accurate and robust method because of the number of decision trees participating in the process.
- It does not suffer from the over fitting problem. The main reason is that it takes the average of all the predictions, which cancels out the biases.
- The algorithm can be used in both classification and regression problems.

Disadvantages:-

- Random forest is slow in generating predictions because it has multiple decision trees. Whenever it makes a prediction, all the trees in the forest
- Have to make a prediction for the same given input and then perform voting on it. This whole process is time-consuming.
- The model is difficult to interpret compared to a decision tree, where you can easily make a decision by following the path in the tree.

```
[86] from sklearn.ensemble import RandomForestClassifier
     rfc=RandomForestClassifier(n_estimators=1,criterion='entropy',random_state=7)
     rfc.fit(x_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=None, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=1,
                       n_jobs=None, oob_score=False, random_state=7, verbose=0,
                       warm_start=False)
```

```
[87] rfc_pred=rfc.predict(x_test)
```

```
[88] print(confusion_matrix(y_test,rfc_pred))
     print(metrics.accuracy_score(y_test,rfc_pred))
```

```
[[12418  8007]
 [ 7962 12255]]
0.6070813444220264
```

Accuracy is 60.70%

After tuning

Hyperparamter Tunning

```
[89] param_grid = {
        'n_estimators': range(25,100,25),
        'max_depth' : range(1,25,5),
        'criterion' :['gini','entropy']
     }
```

```
[90] CV_rfc = GridSearchCV(estimator=rfc,param_grid=param_grid,cv=5, verbose=1)
```

```
[91] CV_rfc.fit(x_train, y_train)
```

```
Fitting 5 folds for each of 30 candidates, totalling 150 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 150 out of 150 | elapsed: 15.7min finished
GridSearchCV(cv=5, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='entropy',
                                              max_depth=None,
                                              max_features='auto',
                                              max_leaf_nodes=None,
                                              max_samples=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              n_estimators=1, n_jobs=None,
                                              oob_score=False, random_state=7,
                                              verbose=0, warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(1, 25, 5),
                         'n_estimators': range(25, 100, 25)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=1)
```

```
[92] print(CV_rfc.best_estimator_)
     print(CV_rfc.best_score_)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=16, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=75,
                       n_jobs=None, oob_score=False, random_state=7, verbose=0,
                       warm_start=False)
0.7021751315968631
```

```
[93] rfc_t=RandomForestClassifier(criterion='entropy',
                                  random_state=7,
                                  max_depth=21,
                                 n_estimators=75)
     rfc_t.fit(x_train,y_train)
```

```
RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                       criterion='entropy', max_depth=21, max_features='auto',
                       max_leaf_nodes=None, max_samples=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=1, min_samples_split=2,
                       min_weight_fraction_leaf=0.0, n_estimators=75,
                       n_jobs=None, oob_score=False, random_state=7, verbose=0,
                       warm_start=False)
```

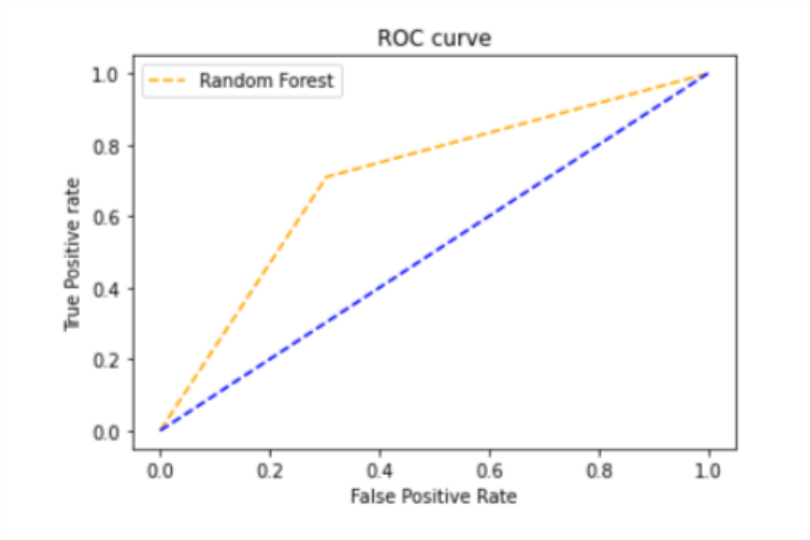
```
[94] y_pred_rfc=rfc_t.predict(x_test)
```

```
[95] print(confusion_matrix(y_test,y_pred_rfc))
     print(metrics.accuracy_score(y_test,y_pred_rfc))
```

```
[[14227  6198]
 [ 5863 14354]]
0.7032380296245263
```

After Tuning Accuracy is 70.32%

Roc Curve:


ROC curve
Random Forest
True Positive rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

DECISION TREE

Decision tree is a **graphical representation of all possible solutions to a decision**. These days, tree-based algorithms are the most commonly used algorithms in the case of supervised learning scenarios. They are easier to interpret and visualize with great adaptability. We can use tree-based algorithms for both regression and classification problems, However, most of the time they are used for classification problem.

Advantages:-

- Less data cleaning required-It is fairly immune to outliers and missing data, hence less data cleaning is needed.
- The data type is not a constraint-It can handle both categorical and numerical data.

Disadvantages:-

- Over fitting: single decision tree tends to over fit the data which is solved by setting constraints on model parameters.
- Not exact fit for continuous data: It losses some of the information associated with numerical variables when it classifies them into different categories.

```
[76] from sklearn.tree import DecisionTreeClassifier

     dt_default=DecisionTreeClassifier(criterion='entropy',random_state=3)
     dt_default.fit(x_train,y_train)

     DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                            max_depth=None, max_features=None, max_leaf_nodes=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, presort='deprecated',
                            random_state=3, splitter='best')

[77] dt_default_pred=dt_default.predict(x_test)

[78] confusion_matrix(y_test,dt_default_pred)

     array([[12542,  7883],
            [ 7856, 12361]])

[79] metrics.accuracy_score(y_test,dt_default_pred)

     0.6127405147384479
```

Accuracy is 61.27%

After tuning

```
from sklearn.tree import DecisionTreeClassifier


# Setup the parameters and distributions to sample from: param_dist
param_dist = {"max_depth": range(10,25,5),
              "min_samples_split": range(50,200,25),
              "min_samples_leaf": range(50,200,25),
              "criterion": ["gini", "entropy"]}

# Instantiate a Decision Tree classifier: tree
dtree = DecisionTreeClassifier(random_state=7)


dtree_cv = GridSearchCV(dtree, param_dist, cv=5,verbose=1
                        ,return_train_score=True)

# Fit it to the data
dtree_cv.fit(x_train,y_train)
```

```
Fitting 5 folds for each of 216 candidates, totalling 1080 fits
[Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
[Parallel(n_jobs=1)]: Done 1080 out of 1080 | elapsed: 11.4min finished
GridSearchCV(cv=5, error_score=nan,
             estimator=DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=None,
                                              max_leaf_nodes=None,
                                              min_impurity_decrease=0.0,
                                              min_impurity_split=None,
                                              min_samples_leaf=1,
                                              min_samples_split=2,
                                              min_weight_fraction_leaf=0.0,
                                              presort='deprecated',
                                              random_state=7, splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': range(10, 25, 5),
                         'min_samples_leaf': range(50, 200, 25),
                         'min_samples_split': range(50, 200, 25)},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=True,
             scoring=None, verbose=1)
```

```
[82] print("best_accuracy",dtree_cv.best_score_)
     print(dtree_cv.best_estimator_)
     print(dtree_cv.best_params_)
```

```
best_accuracy 0.675691427085301
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=20, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=125, min_samples_split=50,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=7, splitter='best')
{'criterion': 'entropy', 'max_depth': 20, 'min_samples_leaf': 125, 'min_samples_split': 50}
```

```
dtree_tn=DecisionTreeClassifier(criterion='entropy',
                                random_state=7,
                                max_depth=25,#15
                                min_samples_leaf=125,#95
                                min_samples_split=60)#50


dtree_tn.fit(x_train,y_train)
```

```
DecisionTreeClassifier(ccp_alpha=0.0, class_weight=None, criterion='entropy',
                       max_depth=25, max_features=None, max_leaf_nodes=None,
                       min_impurity_decrease=0.0, min_impurity_split=None,
                       min_samples_leaf=125, min_samples_split=60,
                       min_weight_fraction_leaf=0.0, presort='deprecated',
                       random_state=7, splitter='best')
```

```
[112] dtree_tn.score(x_test,y_test)
```

```
0.6723586437675311
```

accuracy is 67.23%

After Tuning Accuracy is 67.23%

ADABOOST

This is a type of ensemble technique, where a number of weak learners are combined together to form a strong learner. Here, usually, each weak learner is developed as **decision stumps** (A stump is a tree with just a single split and two terminal nodes) that are used to classify the observations.

Adaboost increases the predictive accuracy by assigning weights to both observations at end of every tree and weights (scores) to every classifier. Hence, in Adaboost, every classifier has a different weightage on final prediction contrary to the random forest where all trees are assigned equal weights.

```
[ ] from sklearn.ensemble import AdaBoostClassifier
    ada = AdaBoostClassifier(random_state=7)
    ada.fit(x_train, y_train)
    ada_pred = ada.predict(x_test)
```

```
[ ] print(confusion_matrix(y_test,ada_pred))
    print(accuracy_score(y_test,ada_pred))
```

```
[[13710  6715]
 [ 6787 13430]]
0.6677820973377294
```

```
[ ] from sklearn.metrics import f1_score
    from sklearn.model_selection import cross_val_score
    scores = cross_val_score(ada, x_train, y_train, cv=5)
    print("Average cross validation score: {:.3f}".format(scores.mean()))
    print("Test accuracy: {:.3f}".format(ada.score(x_test, y_test)))
    print("F1 score: {:.3f}".format(f1_score(y_test, ada_pred)))
```

```
Average cross validation score: 0.672
Test accuracy: 0.668
F1 score: 0.665
```

Accuracy is 66.80%

XGBOOST

XGBoost is an implementation of gradient boosted **decision trees** designed for speed and performance.

```
[ ] from xgboost import XGBClassifier
    from sklearn.metrics import accuracy_score
    model = XGBClassifier(n_estimators=100,random_state=7)
    model.fit(x_train,y_train)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=0,
              learning_rate=0.1, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=7,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
```

```
[ ] model_pred=model.predict(x_test)
```

```
[ ] print(confusion_matrix(y_test,model_pred))
    print(accuracy_score(y_test,model_pred))
```

```
[[13907  6518]
 [ 6042 14175]]
0.690960090546725
```

Before tuning Accuracy 69.09%

After Tuning

After tuning

```
[ ] param_grid = {
        'base_score':[0.8,0.9,1.0,2.0,0.5],
        'learning_rate':[0.1,0.2,0.3,0.4],
        'gamma' :[1]
    }
```

```
[ ] xgboost_df = GridSearchCV(estimator=model,param_grid=param_grid,cv=8, verbose=1)
```

```
[ ] xgboost_df.fit(x_train, y_train)
```

```
  FitFailedWarning)
[Parallel(n_jobs=1)]: Done 160 out of 160 | elapsed: 10.2min finished
GridSearchCV(cv=8, error_score=nan,
             estimator=XGBClassifier(base_score=0.5, booster='gbtree',
                                     colsample_bylevel=1, colsample_bynode=1,
                                     colsample_bytree=1, gamma=0,
                                     learning_rate=0.1, max_delta_step=0,
                                     max_depth=3, min_child_weight=1,
                                     missing=None, n_estimators=100, n_jobs=1,
                                     nthread=None, objective='binary:logistic',
                                     random_state=7, reg_alpha=0, reg_lambda=1,
                                     scale_pos_weight=1, seed=None, silent=None,
                                     subsample=1, verbosity=1),
             iid='deprecated', n_jobs=None,
             param_grid={'base_score': [0.8, 0.9, 1.0, 2.0, 0.5], 'gamma': [1],
                         'learning_rate': [0.1, 0.2, 0.3, 0.4]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
```

```
[ ] print(xgboost_df.best_estimator_)
    print(xgboost_df.best_score_)
```

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
              colsample_bynode=1, colsample_bytree=1, gamma=1,
              learning_rate=0.4, max_delta_step=0, max_depth=3,
              min_child_weight=1, missing=None, n_estimators=100, n_jobs=1,
              nthread=None, objective='binary:logistic', random_state=7,
              reg_alpha=0, reg_lambda=1, scale_pos_weight=1, seed=None,
              silent=None, subsample=1, verbosity=1)
0.7032413588476379
```

```
[ ] model_pred2=xgboost_df.predict(x_test)
```

```
[ ] print(confusion_matrix(y_test,model_pred2))
    print(metrics.accuracy_score(y_test,model_pred2))
```

```
[[14138  6287]
 [ 5807 14410]]
0.7024260617095616
```

After Tuning Accuracy is 70.24%

Plotting roc curve of random forest and xgboost

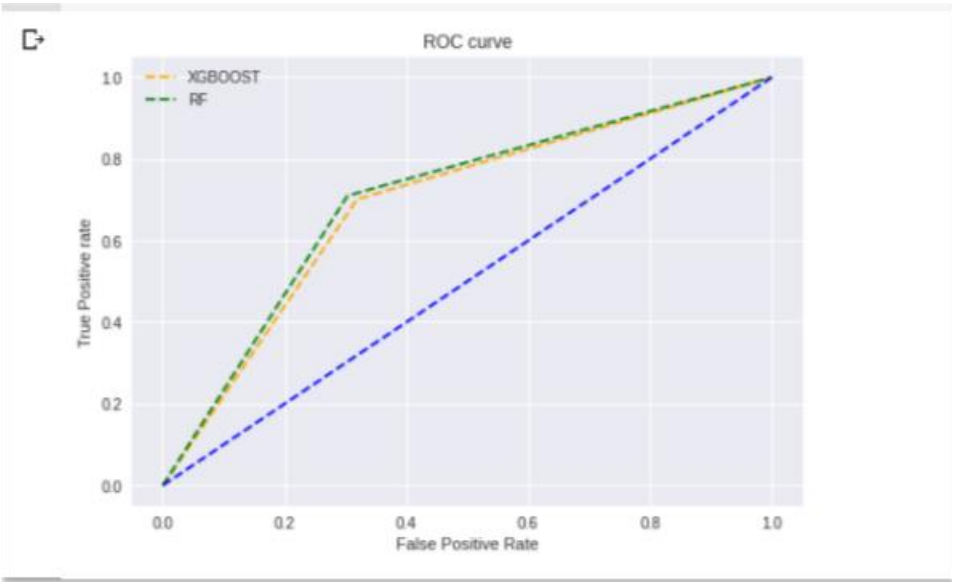
```
[ ] from sklearn.metrics import roc_curve

    # roc curve for models
    fpr1, tpr1, thresh1 = roc_curve(y_test, model_pred, pos_label=1)
    fpr2, tpr2, thresh2 = roc_curve(y_test, y_pred_rfc, pos_label=1)

    # roc curve for tpr = fpr
    random_probs = [0 for i in range(len(y_test))]
    p_fpr, p_tpr, _ = roc_curve(y_test, random_probs, pos_label=1)
```

```
[ ] import matplotlib.pyplot as plt
    plt.style.use('seaborn')

    # plot roc curves
    plt.plot(fpr1, tpr1, linestyle='--',color='orange', label='XGBOOST')
    plt.plot(fpr2, tpr2, linestyle='--',color='green', label='RF')
    plt.plot(p_fpr, p_tpr, linestyle='--', color='blue')
    # title
    plt.title('ROC curve')
    # x label
    plt.xlabel('False Positive Rate')
    # y label
    plt.ylabel('True Positive rate')
    plt.legend(loc='best')
    plt.show();
```



Results

1. From All the models which we have fitted Random forest is gives good accuracy around 70.32%
2. The accuracy of each and every algorithm is not up to good.
3. The reason behind accuracy may be, data is more complex and capturing so many variations.

Requirement Specifications

Operating system: Linux/Windows 8.1 above

Software Requirement:

- **Google collab/jupyter notebook**

Minimum Requirement:

- **Processor:-Intel(R) core(TM) i3-6006U CPU @2.00GHz 1.99 GHz**
- **RAM:-12GB**
- **Hard Disk:-1TB**

Technology Used:

- **Machine Learning**

Programming Language:

- **Python**

Future Scope

The scope of Machine Learning is not limited to only predictions. We pose the vehicle accident risk prediction as a classification problem with two labels (accident and no accident).

There are several parameters we use to predict the model such as type of road i.e. one way, two ways, highway or bypass etc.

Among many other parameters we also take into consideration weather conditions like the chances of road being wet due to rain, dew or are the road dry.

Conclusion

The study analysed the vehicle accident risk for different weather conditions. According to the results the vehicle accident risks were higher if we consider condition of road with respect to weather i.e. in snow covered. For the precipitation the relative accident risk was higher for precipitation type i.e. during clear compared to the other precipitation type. If we consider light condition then during night the chances of getting accident is higher than other light conditions. Applying various model we got best accuracy by using random Forest.

REFERENCES

- Accident risk of road and weather conditions on different road types
  VTT Technical Research Centre of Finland Ltd., Vuorimiehentie 3, 02150 Espoo, Finland
  Received 10 February 2018, Revised 10 August 2018, Accepted 20 October 2018, Available online 29 October 2018.

- "Driving Pattern Analysis System using Dataset
  IJIRST –International Journal for Innovative Research in Science & Technology| Volume 2 | Issue 12 | May 2016 ISSN (online): 2349-6010

- Aggressive Driving Detection Using Deep Learning-based Time Series Classification
  Aggressive Driving Detection Using Deep Learning-based Time Series Classification - IEEE Conference Publication