Effect of Vehicle type, age and Engine capacity on Accident Severity using AI

2112300

GOUTHAM PAL (gr22096)

INTRODUCTION

predicting the possibility of a vehicle getting in an accident is the fundamental step which insurance companies make while fixing vehicle insurance cost. They take many attributes to consideration like age of driver, price of vehicle, age of vehicle, etc,. Like wise the government, while planning to build a road. analyse many attributes and come up with a best possible plan which can minimise the causality. Here we are predicting Accident severity with respect to vehicle type, vehicle age and Engine capacity. To what extent it contributes on deciding the level of severity. If this has a major impact on the model, this could be a useful guide for customers to make decision while buying a vehicle. we have a waste amount of data on the accident which happened on The United Kingdom(UK), can apply machine learning algorithm to make prediction.

This project aims to find the impact of variables on an algorithm, By using algorithm like K-Nearest Neighbours, Random forest, Decision tree, Gaussian Naïve Bayes, Linear Regression and Support Vector Machine to find the accuracy of each model.

Data Discription

The data have been open sourced by the UK Government, can find here [http://data.gov.uk/ (http://data.gov.uk/)](http://data.gov.uk/). This consist of the details of Road Accidents which occurred between 1979 to 2015 consisting of 70 features or columns and around 250,000 incidents/observations/rows. The zip file also consist second excel file named 'Road-Accident-Safety-Data-Guide' with multiple tabs for better understanding of data and values associated with data.

As this dataset contains many missing values, Rather than applying any regression or taking mean to find the missing values am just dropping the entire row/observation as the dataset massive with around 250k rows. Furthermore, there are some rows with -1 value, am converting thaem as missing values and droping those rows as well. Even after droping all those observation there will be enough amount of data to caryout our test

METHODS

The best way to know the effectivness of a feature on a specific outcome is:

- Creating two datasets with possible dependent features. one which has the required feature and the outher dataset which dosn't.
- Run both the datasets in some machine learning models.
- while going through this modle, the datasets have to be sampled and should be split into test and training data. Here I have taken 1:3 split (where 33% of the datas are test datas and remainig 66% are train datas).
- After applyinh the alogrithms on to the dataset we get the accuricy of the boht model.

- by comparing accruicy of both the dataset, we can determing that the impact of the attribute on the dererminent variable.
- If there is no much deviation in both the output, it means the vehicle infos dose not have impact on accident severity.

** Algorithms Used:**

- **K-Nearest Neighbors (KNN)**
- **Linear Regression (LR)**
- **Gaussian Naive Bayes (NB)**
- **Decision Tree (DT)**
- **Support Vector Machines**
- **Random Fprest(RF)**

K-Nearest Neighbors -- We just label the unknown variable to the group of values that are close to and maximam in number using the classification technique. Using this method, we may discover that the accuracy of the model has somewhat increased following the addition of the vehicle component. This indicates a minimal dependence if the classification approach is used.

Linear Regression -- Regression probably one of the best algorithm which gives considerabally good results in both the cases with accuricy of around 91%. in linear reegression first it takes the root mean square distance of all the points. Which discloses the liklywood the severity of the accident.

Gaussian Naïve Bayes -- Since Nave Bayes is a probabilistic classification technique based on applying Bayes' theorem with strong independence assumptions, applying it to this situation may not be a good idea. As a result of the fact that all of the data in this instance are interdependent, we can see that the first dataset's test run accuracy is very low—less than 10%—possibly because only a small number of features were chosen, each of which has a negligible impact on accident severity.

Decision Tree -- This classifier builds a decision tree to decide which class value should be assigned to each data item. We can choose the maximum number of features the model will take into account at this stage. Following the addition of the extra columns, reduced the accuracy of the model as the complixity of the tree increased making the modle more scattred.

Random Forest -- A large number of decision trees are built during the training phase of the random forests or random decision forests ensemble learning approach, which is used for classification, regression, and other tasks.The mean or average forecast of each individual tree is returned for regression tasks. This modle peformed exceptionally well on both the data set giving maximum accuracy with the improvement in the second dataset, which denotes the dependancy of severity on vehicle informaiton. The tendency of decision trees to overfit their training set is corrected by random decision forests. Although they frequently outperform decision trees, gradient enhanced trees are more accurate than random forests. But data features can impact how well they work.

RESULT

importing libraries

In [2]:

```
# Importing Libraries
import pandas as pd
import numpy as np
import seaborn as sns
# Importing matplotlib to show graphical representation of the attributes
import matplotlib.pyplot as plt
# to ignore/remove warnings
import warnings
warnings.filterwarnings('ignore')
```

DATASET

In [3]:

```
#importing dataset(read the data from csv)
ds = pd.read_csv("C:\\Users\\Dell Vostro\\Desktop\\FinalProject\\UK_AccidentData.csv",decim
#change file path accordingly
```

In [4]:

```
#droping the observation which contains NA(missing values)
#As the data on the observation is massive droping some dosen't effect the study
ds = ds.dropna()
ds.head()
```

Out[4]:

	accident_index	vehicle_reference	vehicle_type	towing_and_articulation	vehicle_manoeuvre
2	201506E098766	2	9	0	18
3	201506E098777	1	20	0	4
5	201506E098780	2	1	0	9
6	201506E098792	1	3	0	4
8	201506E098804	1	9	0	14

5 rows × 70 columns

In [5]:

```
ds.shape
```

Out[5]:

```
(174538, 70)
```

Initally there was around 250k observation, after droping the ones that contain NA values we get around 170k observaiton. This is more than sufficient for the analysis.

Creating two dataset with and without vehicle specification

In [6]:

```
#selection of the varaibles which has major impact on determining the accident severity
ds1 = ds[['special_conditions_at_site','pedestrian_movement','road_surface_conditions','lig
#same varaibles with additionalfeateres of vehicle like vehicle_type, engine_capacity_(cc)
ds2 = ds[['special_conditions_at_site','pedestrian_movement','vehicle_type','engine_capacit
```

I observed that some of the values are -1 which indicate missing values, which need to be converted to NA and dropped. Event after this filtering we have enough data for carrying out the observation.

In [7]:

```
#imputing -1 values to NaN to be recognized as missing data
ds1.replace(-1, np.nan, inplace=True)
#droping the rows which contains missing values
ds1 = ds1.dropna()
ds1.shape
```

Out[7]:

```
(99146, 13)
```

In [8]:

```
# Repeating the pervious step for second dataset
ds2.replace(-1,np.nan, inplace=True)
ds2 = ds2.dropna()
ds2.shape
```

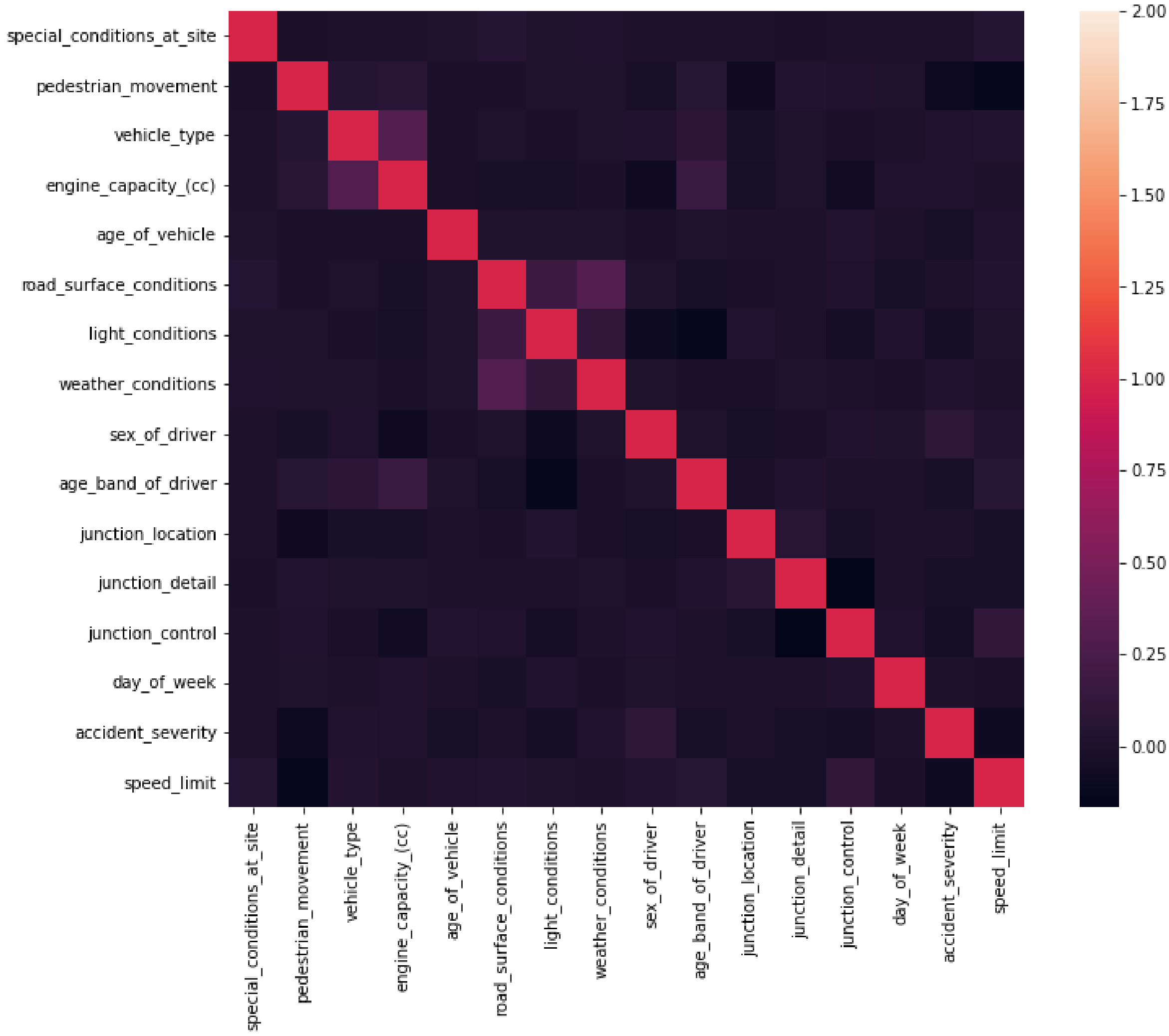
Out[8]:

```
(68008, 16)
```

PRELIMINARY DATA ANALYSIS

In [9]:

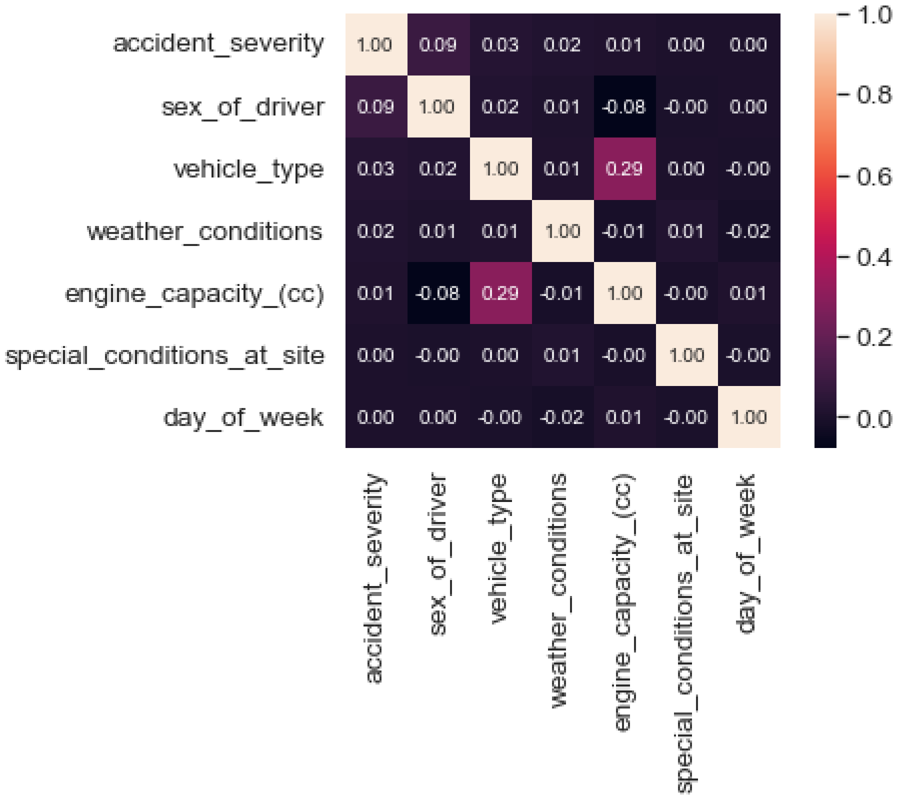
```
# Graphical repersentation to show the co-relation between each elements on the second data
import matplotlib.pyplot as plt
corrmat = ds2.corr()
f, ax = plt.subplots(figsize=(12,9))
sns.heatmap(corrmat, vmax=2, square=True)

plt.show()
```



From above, the coorelation between each specified feature is identified the interdependancy between variables.The matrix is color-coded, with a value of one denoted by beige and displaying a wholly positive linear connection. Dark purple is a zero that denotes the absence of a linear association. For clear view we take a closer look on the values of the coorelation in the next part which shows that there is slight dependancy between vehicle_type and engine_capacity(cc) with accident_severity.

In [10]:

```
k = 7 #number of variables for heatmap
cols = corrmat.nlargest(k, 'accident_severity')['accident_severity'].index
cm = np.corrcoef(ds2[cols].values.T)
sns.set(font_scale=1.25)
hm = sns.heatmap(cm, cbar=True, annot=True, square=True, fmt='.2f', annot_kws={'size': 10},
plt.show()
```



In [11]:

```
ds2.head()
```

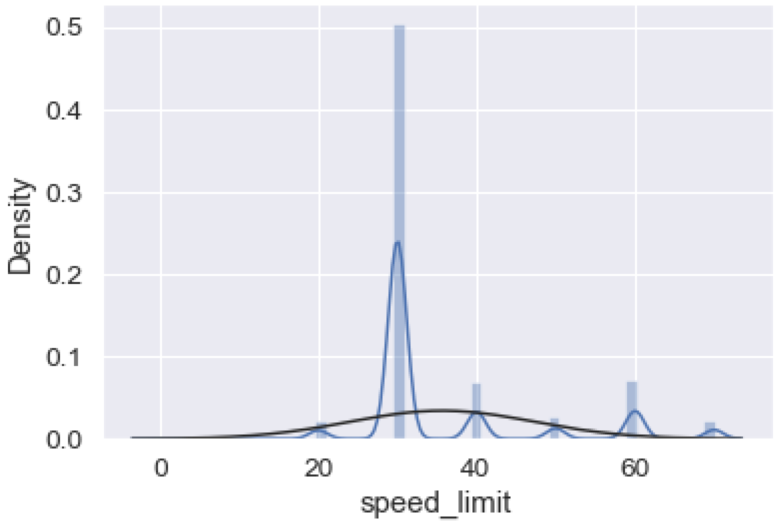
Out[11]:

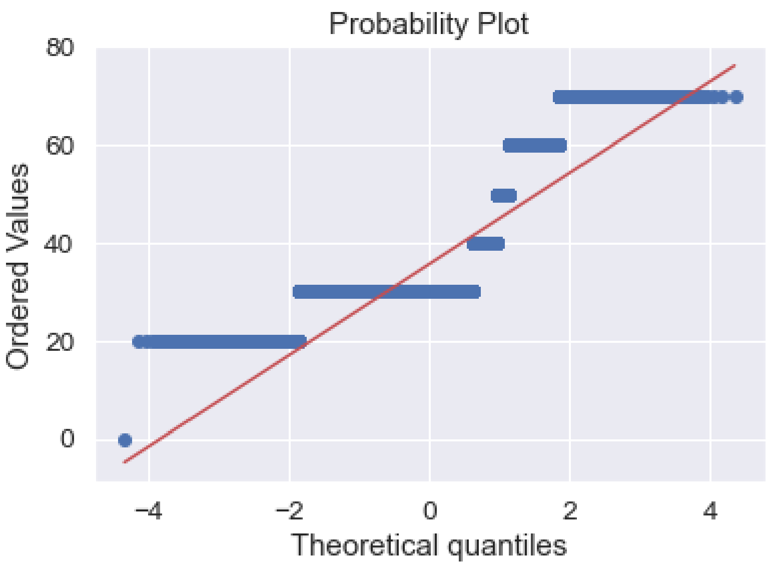
	special_conditions_at_site	pedestrian_movement	vehicle_type	engine_capacity_(cc)	age_of_
6	0.0	0.0	3.0	50.0	
8	0.0	2.0	9.0	1398.0	
14	0.0	0.0	9.0	999.0	
25	0.0	0.0	9.0	1242.0	
26	0.0	3.0	9.0	1896.0	

The next step was to normalize the only features that were not categorical: speed_limit and age_of_vehicle. Normalization involves taking the logarithm of the given features. This is done to because high values for certain variables computationally skew results more in favour of that variable, than their actual contribution. The only characteristics that were not categorical were speed limit and age of vehicle, which were then normalised. By taking the logarithm of the provided characteristics, normalisation is accomplished. High numbers for some variables computationally distort outcomes more in favour of that variable than their real contribution, therefore this is done.

In [12]:

```
from scipy.stats import norm
from scipy import stats
#histogram and normal probability plot
sns.distplot(ds1['speed_limit'], fit=norm);
fig = plt.figure()
res = stats.probplot(ds1['speed_limit'], plot=plt)
plt.show()
```





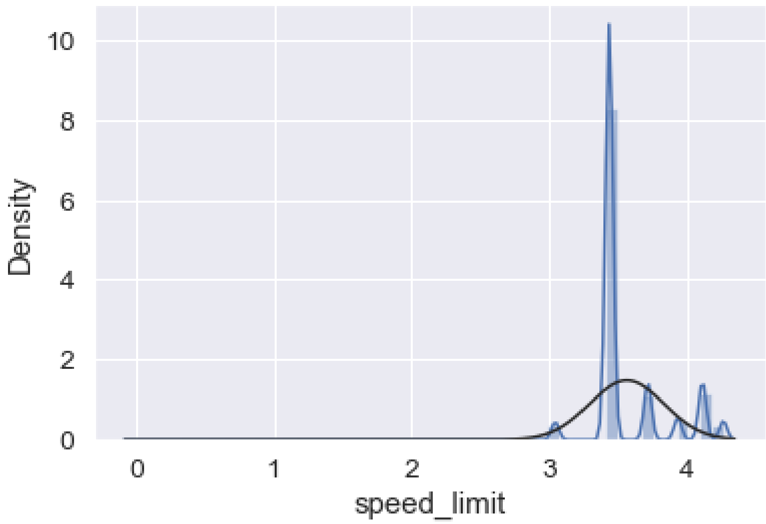
Here the only variables with a substantial numerical variation were the speed limit and the age of the vehicle, therefore logarithms of both variables were calculated. Taking the logarithm of both the speed limit and the vehicle's age further improved the fit by changing the scale and making the variables more "normally" distributed.

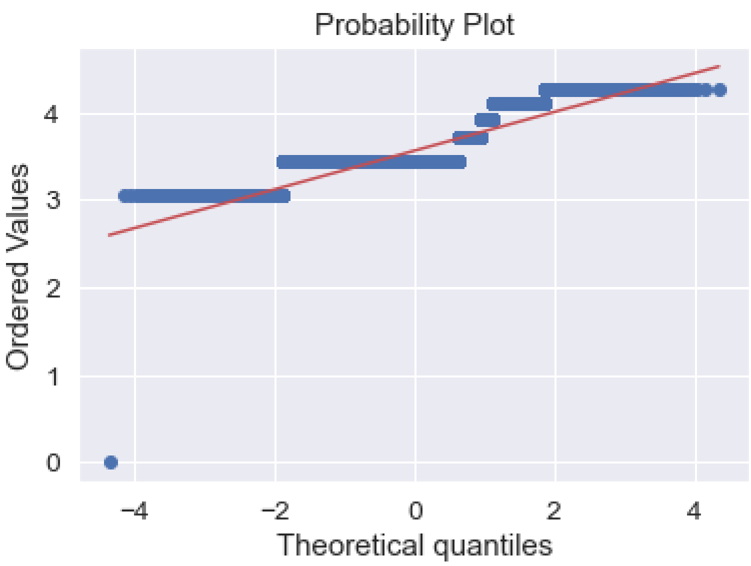
In [13]:

```
ds2['speed_limit'] = np.log1p(ds2['speed_limit'])
ds2['age_of_vehicle'] = np.log1p(ds2['age_of_vehicle'])
ds2['engine_capacity_(cc)'] = np.log1p(ds2['engine_capacity_(cc)'])

ds1['speed_limit'] = np.log1p(ds1['speed_limit'])
```

In [14]:

```
sns.distplot(ds1['speed_limit'], fit=norm);
fig = plt.figure()
res = stats.probplot(ds1['speed_limit'], plot=plt)
plt.show()
```





In [15]:

```
ds2
```

Out[15]:

	special_conditions_at_site	pedestrian_movement	vehicle_type	engine_capacity_(cc)	ag
6	0.0	0.0	3.0	3.931826	
8	0.0	2.0	9.0	7.243513	
14	0.0	0.0	9.0	6.907755	
25	0.0	0.0	9.0	7.125283	
26	0.0	3.0	9.0	7.548029	
...	...	...	...	...	
257629	0.0	0.0	11.0	9.392745	
257639	0.0	0.0	8.0	7.469654	
257643	0.0	0.0	5.0	7.158514	
257649	0.0	0.0	9.0	7.493874	
257650	0.0	0.0	9.0	7.372118	

68008 rows × 16 columns

In [16]:

```
# As the data is huge, we only take portion of data to reduse the computational time
ds1= ds1[:15000]
ds2= ds2[:15000]

Y = ds2.accident_severity.values
Y1 = ds1.accident_severity.values
Y
```

Out[16]:

```
array([3, 3, 3, ..., 3, 3, 3], dtype=int64)
```

In [17]:

```
cols = ds2.shape[1]
X = ds2.loc[:, ds2.columns != 'accident_severity']
X1 = ds1.loc[:, ds1.columns != 'accident_severity']
X.columns;
```

In [18]:

```
X1.shape
```

Out[18]:

```
(15000, 12)
```

In [19]:

```
X.shape
```

Out[19]:

```
(15000, 15)
```

Importing modules to deploy algorithms

In [20]:

```
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
```

Sampling

Sampling and trainig machine learning algorithms with out considering vehicle information

In [21]:

```
# splitting dataset into test and train with 1:3 ratio.
X_train1, X_test1,Y_train1,Y_test1 = train_test_split(X1, Y1, test_size=0.33, random_state=
```

Deploying Algorithms

K-Nearest Neighbors (KNN)

In [22]:

```
#KNN
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train1, Y_train1)
Y_pred = knn.predict(X_test1)
acc_knn1 = round(knn.score(X_test1, Y_test1) * 100, 2)
acc_knn1
```

Out[22]:

```
90.16
```

Linear Regression (LR)

In [23]:

```
#LR
logreg = LogisticRegression()
logreg.fit(X_train1, Y_train1)
Y_pred = logreg.predict(X_test1)
acc_log1 = round(logreg.score(X_train1, Y_train1) * 100, 2)
acc_log1
```

Out[23]:

```
91.55
```

Gaussian Naive Bayes (NB)

In [24]:

```
#NB
gaussian = GaussianNB()
gaussian.fit(X_train1, Y_train1)
Y_pred = gaussian.predict(X_test1)
acc_gaussian1 = round(gaussian.score(X_test1, Y_test1) * 100, 2)
acc_gaussian1
```

Out[24]:

```
9.23
```

Decision Tree (DT)

In [25]:

```
#DT
decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train1, Y_train1)
Y_pred = decision_tree.predict(X_test1)
acc_decision_tree1 = round(decision_tree.score(X_test1, Y_test1) * 100, 2)
acc_decision_tree1
```

Out[25]:

```
87.86
```

Random Forest (RF)

In [26]:

```
#RF
random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train1, Y_train1)
Y_pred = random_forest.predict(X_test1)
random_forest.score(X_train1, Y_train1)
acc_random_forest1 = round(random_forest.score(X_test1, Y_test1) * 100, 2)
acc_random_forest1
```

Out[26]:

```
92.0
```

Support Vector Machines (SVM)

In [27]:

```
#SVM
svc = SVC()
svc.fit(X_train1, Y_train1)
Y_pred = svc.predict(X_test1)
acc_svc1 = round(svc.score(X_test1, Y_test1) * 100, 2)
acc_svc1
```

Out[27]:

```
92.48
```

Sampling and trainig machine learning algorithms considering vehicle information

In [28]:

```
# splitting dataset into test and train with 1:3 ratio.
X_train, X_test,Y_train,Y_test = train_test_split(X, Y, test_size=0.33, random_state=99)
```

K-Nearest Neighbors (KNN)

In [29]:

```
#KNN

knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, Y_train)
Y_pred = knn.predict(X_test)
acc_knn = round(knn.score(X_test, Y_test) * 100, 2)
acc_knn
```

Out[29]:

```
90.81
```

Linear Regression (LR)

In [30]:

```
# LR
logreg = LogisticRegression()
logreg.fit(X_train, Y_train)
Y_pred = logreg.predict(X_test)
acc_log = round(logreg.score(X_train, Y_train) * 100, 2)
acc_log
```

Out[30]:

```
92.34
```

Gaussian Naive Bayes (NB)

In [31]:

```
# NB

gaussian = GaussianNB()
gaussian.fit(X_train, Y_train)
Y_pred = gaussian.predict(X_test)
acc_gaussian = round(gaussian.score(X_test, Y_test) * 100, 2)
acc_gaussian
```

Out[31]:

```
87.11
```

Decision Tree (DT)

In [32]:

```
# DT

decision_tree = DecisionTreeClassifier()
decision_tree.fit(X_train, Y_train)
Y_pred = decision_tree.predict(X_test)
acc_decision_tree = round(decision_tree.score(X_test, Y_test) * 100, 2)
acc_decision_tree
```

Out[32]:

```
86.87
```

Random Forest (RF)

In [33]:

```
# RF

random_forest = RandomForestClassifier(n_estimators=100)
random_forest.fit(X_train, Y_train)
Y_pred = random_forest.predict(X_test)
random_forest.score(X_train, Y_train)
acc_random_forest = round(random_forest.score(X_test, Y_test) * 100, 2)
acc_random_forest
```

Out[33]:

```
93.33
```

Support Vector Machines (SVM)

In [34]:

```
#SVM
svc = SVC()
svc.fit(X_train, Y_train)
Y_pred = svc.predict(X_test)
acc_svc = round(svc.score(X_test, Y_test) * 100, 2)
acc_svc
```

Out[34]:

```
92.32
```

Scores wihtout considering Vehicle Information

In [35]:

```
print("Machine Learning algorithm scores without Vehicle Info")
models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes',
              'Decision Tree','Support Vector Machines'],
    'Score': [acc_knn1, acc_log1,
              acc_random_forest1, acc_gaussian1,acc_decision_tree1,acc_svc1]})
models.sort_values(by='Score', ascending=False)
```

```
Machine Learning algorithm scores without Vehicle Info
```

Out[35]:

	Model	Score
5	Support Vector Machines	92.48
2	Random Forest	92.00
1	Logistic Regression	91.55
0	KNN	90.16
4	Decision Tree	87.86
3	Naive Bayes	9.23

Scores considering Vehicle Informaition

In [36]:

```
print("Machine Learning algorithm scores with Vehicle Info")
models = pd.DataFrame({
    'Model': ['KNN', 'Logistic Regression',
              'Random Forest', 'Naive Bayes',
              'Decision Tree','Support Vector Machines'],
    'Score': [acc_knn, acc_log,
              acc_random_forest, acc_gaussian,acc_decision_tree,acc_svc]})
models.sort_values(by='Score', ascending=False)
```

```
Machine Learning algorithm scores with Vehicle Info
```

Out[36]:

	Model	Score
2	Random Forest	93.33
1	Logistic Regression	92.34
5	Support Vector Machines	92.32
0	KNN	90.81
3	Naive Bayes	87.11
4	Decision Tree	86.87

The results showed that adding vehicle characteristics to a machine learning algorithm in forecasting accident severity had a minor effect on model accuracy. The results show a high level of accuracy. Because this is a multilabel classification, the accuracy metric computes the number of labels predicted that perfectly match the associated set of labels.

In [37]:

```
# Confusion matrix with random forest
from sklearn.metrics import classification_report, confusion_matrix
x,y = ds1.loc[:,ds1.columns != 'accident_severity'], ds1.loc[:,'accident_severity']
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3,random_state = 1)
rf = RandomForestClassifier(random_state = 3)
rf.fit(x_train,y_train)
y_pred = rf.predict(x_test)
cm = confusion_matrix(y_test,y_pred)
print('Confusion matrix: \n',cm)
print('Classification report: \n',classification_report(y_test,y_pred))
y_test.value_counts()
```

```
Confusion matrix:
 [[   0    0    2]
 [   0   31  330]
 [   0   67 4070]]
Classification report:
               precision    recall  f1-score   support

           1       0.00      0.00      0.00         2
           2       0.32      0.09      0.14       361
           3       0.92      0.98      0.95      4137

    accuracy                           0.91      4500
   macro avg       0.41      0.36      0.36      4500
weighted avg       0.88      0.91      0.89      4500
```

Out[37]:

```
3    4137
2     361
1       2
Name: accident_severity, dtype: int64
```

In [38]:

```
# Confusion matrix with random forest of ds2
from sklearn.metrics import classification_report, confusion_matrix
x,y = ds2.loc[:,ds2.columns != 'accident_severity'], ds2.loc[:,'accident_severity']
x_train,x_test,y_train,y_test = train_test_split(x,y,test_size = 0.3,random_state = 1)
rf = RandomForestClassifier(random_state = 3)
rf.fit(x_train,y_train)
y_pred = rf.predict(x_test)
cm = confusion_matrix(y_test,y_pred)
print('Confusion matrix: \n',cm)
print('Classification report: \n',classification_report(y_test,y_pred))
y_test.value_counts()
```

```
Confusion matrix:
 [[   4    0   24]
 [   0   54  290]
 [   1    6 4121]]
Classification report:
               precision    recall  f1-score   support

           1       0.80      0.14      0.24        28
           2       0.90      0.16      0.27       344
           3       0.93      1.00      0.96      4128

    accuracy                           0.93      4500
   macro avg       0.88      0.43      0.49      4500
weighted avg       0.93      0.93      0.90      4500
```

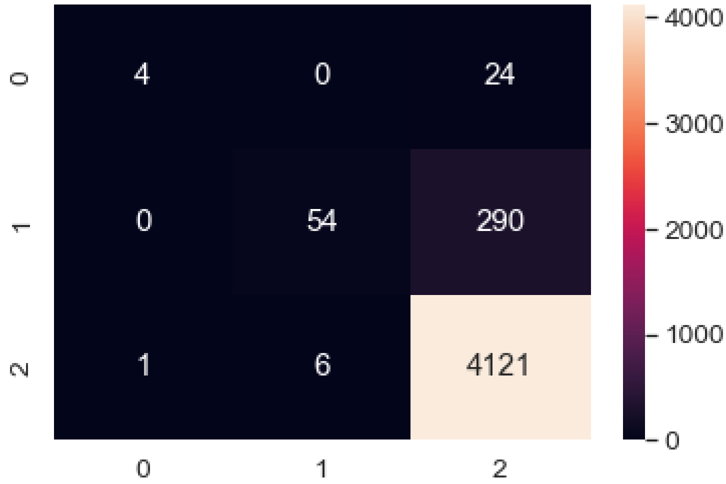
Out[38]:

```
3    4128
2     344
1      28
Name: accident_severity, dtype: int64
```

Although nearly all of the predictions would concern the majority class, a model may forecast the value of the majority class for all predictions and produce a high accuracy, except naïve base. When the classes are severely unbalanced, precision-recall is a helpful indicator of prediction success. Precision measures the relevance of the results, whereas recall measures how many really relevant items are returned. Another statistic is the F1 score, which is a measure of the correctness of a test. When calculating the score, it takes into account the test's precision p and recall r and delivers a result that strikes a balance between the two.

In [39]:

```
sns.heatmap(cm,annot=True,fmt="d")
plt.show()
```



CONCLUSION

It is noticeable that adding car information, such as vehicle type, engine capacity(cc), and age of vehicle improves accuracy, as most algorithms perform well. The random forest model increased by 1.34 points over its prior score of 91.94, which did not account for vehicle features. When the classes are highly unbalanced, precision-recall is a good metric of prediction success. Precision measures the relevance of the results, whereas recall measures how many really relevant items are returned. In conclusion it is observed that the vehicle parameter do contribute to predict the severity of the accident .