Real-Time Prediction of Severity of Car Accidents on Traffic in the US

Vaishnavii P Paramashivam
677711866

Ramnath Ramachandran
670658005

Abstract

With the increase in urbanization process the rate of traffic accidents has also increased in recent decades, causing significant life and property losses. Predicting the severity of these accidents on traffic is a crucial problem to improving transportation, public safety, as well as finding safe routes each year. In this report, we focus on predicting the impact of car accidents on traffic and also identifying the hotspot locations of accident locations in the US. However, the problem is also challenging due to the imbalanced classes, uneven distributions of various concentrations of population as it is real-time big data which has been collected from the time period 2016 to 2019 using several APIs which provide streaming traffic event data. We've formulated the prediction problem as a multi-class classification problem and we've evaluated four models namely Support Vector Machines, Decision Tree, Multinomial Logistic Regression and KNN Classification. To tackle the class imbalance, we've adapted appropriate evaluation metrics which takes the no of samples in each class into consideration. SVM generally performed better than other models. We found the hotspot locations of car accidents to be states like California followed by Texas and New York.

1.Introduction

Traffic accidents are a major issue for public safety. On average 6 million car accidents occur in the US each year. Around 1.25 million people are killed in car accidents each year. The ability to predict the impact of these car accidents on traffic will provide more insights that will help the transportation administrators and individual travelers. The insights which we've obtained by predicting the severity of such accidents on traffic include identifying important weather factors, factors of road system (like bumps, potholes, traffic signal post.,etc) that contribute to the occurrence of these accidents. A potential application of such a technique would be identifying real-time safe route recommendations for commuters and drivers. The rapid development of data collection techniques had led to the availability of big real-time datasets. We've obtained our dataset from the data science community Kaggle where the real-time traffic data has been collected using several data providers, including two APIs which provide streaming traffic event data which makes predicting traffic accidents more realistic. However, this problem statement is very challenging due to the following issues. (1) Class imbalance. The majority of the accidents causing least impact on traffic has not been reported and hence recorded, causing the classes to be severely imbalanced. (2) Spatial heterogeneity, i.e., the model parameters may differ from place to place. For example, factors causing traffic accidents in large cities with comparatively dense population and lower speed limits might be very different from those in rural areas with a low density of population but higher speed limits, causing the model to be not very

accurate. (3) The relationship between environmental factors like weather conditions and accidents might be complex and non-linear, which disregards the use of simple linear models and requires the usage of appropriate machine learning models. In this paper, we present our explorations on effective ways to improve accident traffic prediction results, which is an essential step towards building robust and reliable accident traffic models. We have considered a multi-class classification problem. We've collected a dataset, which includes all the motor vehicle crashes in the US between 2016 and 2019, detailed road network, and hourly weather data such as rainfall, air temperature, etc. We conduct experiments on four classification models, i.e., Support Vector Machine, Decision Tree, Multinomial Logistic Regression and KNN.

We highlight our work as follows:

1. We've identified the temporal factors, weather factors and road factors that cause the accidents.
2. We've experimented four classification Support Vector Machine, Decision Tree, Multinomial Logistic Regression and KNN to predict the severity of accidents on traffic in which SVM generally performed better than the other models.
3. We've tackled the class imbalance problem by adapting appropriate evaluation metric that accounts for class imbalance.

2.Related Work

1. Predicting Traffic Accidents Through Heterogeneous Urban Data: A Case Study - Zhuoning Yuan, Xun Zhou, Tianbao Yang, James Timerius
2. Moosavi, Sobhan, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, and Rajiv Ramnath. "A Countrywide Traffic Accident Dataset.", arXiv preprint arXiv:1906.05409 (2019).
3. Moosavi, Sobhan, Mohammad Hossein Samavatian, Srinivasan Parthasarathy, Radu Teodorescu, and Rajiv Ramnath. "Accident Risk Prediction based on Heterogeneous Sparse Data: New Dataset and Insights." In proceedings of the 27th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems, ACM, 2019.

3. Model

We compare four classification models, namely Support Vector Machine(SVM), Decision Tree (DT) , Multinomial Logistic Regression (MNLM) and k-Nearest Neighbours (kNN). For implementing all the models we use scikit, Python's extensive library for Machine learning.

3.1 Baseline Model - K-Nearest Neighbours Classifier

The KNN algorithm is a method for classifying objects based on closest training examples in the feature space. KNN is a type of instance-based learning or lazy learning where the function is only approximated locally and all computation is deferred until classification. The KNN is the fundamental and simplest classification technique when there is little or no prior knowledge about the distribution of the data. This rule simply retains the entire training set during learning and assigns to each query a class represented by the majority label of its k-nearest neighbors in the training set. The Nearest Neighbor rule (NN) is the simplest form of KNN when K = 1.

The similarity or distance between two objects (observations in our case) plays an important role in many tasks of classification or grouping of data. The distance function chosen for our problem is Euclidean distance, which is defined as:

$$d(X, Y) = \sqrt{\sum_{i=1}^{n} d_i(x_i, y_i)}$$

In this method, each sample should be classified similarly to the surrounding samples. Therefore, if the classification of a sample is unknown, then it could be predicted by considering the classification of its nearest neighbor samples.

Given an unknown sample and a training set, all the distances between the unknown sample and all the samples in the training set can be computed. The distance with the smallest value corresponds to the sample in the training set closest to the unknown sample. Therefore, the unknown sample may be classified based on the classification of this nearest neighbor.

The performance of a KNN classifier is primarily determined by the choice of K as well as the distance metric applied. The estimate is affected by the sensitivity of the selection of the neighborhood size K, because the radius of the local region is determined by the distance of the Kth nearest neighbor to the query and different K yields different conditional class probabilities. If K is very small, the local estimate tends to be very poor owing to the data sparseness and the noisy, ambiguous or mislabeled points. In order to further smooth the estimate, we can increase K and take into account a large region around the query.

The pseudocode representation for kNN:

Data: D = {(x_i , c_i) , for i = 1 to m}, where x_i = ($v^{(i)}_1$, $v^{(i)}_2$, ..., $v^{(i)}_n$) is an observation that belongs to class c_i
Data: x = (v_1, v_2, ..., v_n) data to be classified
Result: class to which x belongs
distances ← ∅;
for y_i in D do
    d_i ← d(y_i , x);
    distances ← distances ∪ {d_i};
end Sort distances = {d_i , f or i = 1 to m} in ascending order;
Get the first K cases closer to x, $D^K{}_x$;
class ← most frequent class in $D^K{}_{x;}$

How to select a suitable neighborhood size K is a key issue that largely affects the classification performance of KNN. As for KNN, the small training sample size can greatly affect the selection of the optimal neighborhood size K and the degradation of the classification performance of KNN is easily produced by the sensitivity of the selection of K. Generally speaking, the classification results are very sensitive to two aspects: the data sparseness and the noisy, ambiguous or mislabeled points if K is too small, and many outliers within the neighborhood from other classes if K is too large.

3.2 Multinomial Logistic Model

The MNLM is a statistical method used to predict the probability of class relationship on a predicted variable constructed on several predictor variables. The expected variable in question is nominal and for which there are more than two categories while the predictor variables can be either dichotomous or continuous. The method is used to predict nominal response variables by representing the log odds of the responses are represented as a linear grouping of the explanatory variables. The MNLM is an upgrade version of binary logit regression that tolerates two or more categories of the outcome variable. Like binary logit regression, the MNLM applies maximum likelihood estimation to appraise the chance of categorical membership. MNLMs have restrictive assumptions of independence, normality, and multicollinearity.

In multinomial logistic regression the target y is a variable that ranges over more than two classes; we want to know the probability of y being in each potential class c ∈ C, p(y = c|x).

The multinomial logistic classifier uses a generalization of the sigmoid, called softmax the softmax function, to compute the probability p(y = c|x). The softmax function takes a vector z = [z1,z2,...,zk] of k arbitrary values and maps them to a probability distribution, with each value in the range (0,1), and all the values summing to 1. Like the sigmoid, it is an exponential function.

For a vector z of dimensionality k, the softmax is defined as:

$$\text{softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{k} e^{z_j}} \quad 1 \le i \le k$$

The softmax of an input vector z = [z1,z2,...,zk] is thus a vector itself:

$$\text{softmax}(z) = \left[\frac{e^{z_1}}{\sum_{i=1}^{k} e^{z_i}}, \frac{e^{z_2}}{\sum_{i=1}^{k} e^{z_i}}, \ldots, \frac{e^{z_k}}{\sum_{i=1}^{k} e^{z_i}}\right]$$

The denominator Pk i=1 e zi is used to normalize all the values into probabilities. Thus for example given a vector:

z = [0.6,1.1,−1.5,1.2,3.2,−1.1]

the result softmax(z) is

[0.055,0.090,0.0067,0.10,0.74,0.010]

Again like the sigmoid, the input to the softmax will be the dot product between a weight vector w and an input vector x (plus a bias). But now we'll need separate weight vectors (and bias) for each of the K classes. The Softmax output for a class c is:

$$p(y = c|x) = \frac{e^{w_c \cdot x + b_c}}{\sum_{j=1}^{k} e^{w_j \cdot x + b_j}}$$

Like the sigmoid, the softmax has the property of squashing values toward 0 or 1. Thus if one of the inputs is larger than the others, it will tend to push its probability toward 1, and suppress the probabilities of the smaller inputs.

3.3 Decision Trees

Decision trees (DTs) have been applied as well to analyze the severity of accidents. DTs provide a very useful model to identify the causes of accidents because they can be easily interpreted and, most importantly, decision rules can be easily extracted from them. These rules can be used by road safety analysts to identify the main causes of accidents.

A decision tree (DT) is a predictive model that can be used for both classification and regression tasks. In our case, the class variable has only two states, which means that it is a discrete variable. Hence, in this work, DTs were used to represent classification problems.

DTs are commonly used in the literature because they are hierarchical structures presented graphically. Therefore, they are quite interpretable models. DTs are also simple and transparent. Within a DT, each node represents a predictor variable and each branch represents one of the states of this variable. Each node is associated to the most informative attribute that has not already been chosen from the path that goes from the root to this node. To select the most informative variable, a specific split criterion (SC) is used. A terminal node (or leaf) represents the expected value of the class variable depending on the information contained in the set used to build the model, i.e., the training data set. This leaf node is created with the majority class for the partition of the data set corresponding to the path from the root node to that leaf node.

Procedure DT(X)
1. RS = ∅
2. If X = ∅ then Exit
3. Select a SC
4. Use the SC to sort the attributes {RXi}, i = 1,2, . . . , m
5. For each i = 1, . . . , m
    6. Build DTi calling BuildTree(RXi , γ)
    7. Extract RSi from each DTi
    8. RS = RS ∪ RSi
9. Select another split criterion and go to step4. If there is no SC left then **Exit**

DTs are built descending in a recursive way. The process starts in the root node with the full training data set. A feature is selected depending on the split criterion. The data set is then split into several smaller datasets (one for each state of the split variable). This procedure is applied recursively to each subset until the information gain cannot be increased in any of them. In this way, we obtain the terminal nodes.

The split criterion (SC) that we have considered here are Information gain (IG) and Gini index.

The first step in the Decision tree method is to choose a split criterion. Let us suppose we have Data X and there are m features and that RXi is number i in the importance ranking according to the split criterion (SC) for i = 1, . . . , m. Then, RXi is used as the root node to build DTi for i = 1, . . . , m. For each of the m DTs, once the root node has been fixed, we build the rest of the tree following the same process used for building DTs. After building DTi , the corresponding rule set RSi is extracted ∀i = 1, . . . , m. All the rule sets RSi i = 1, . . . , m are added to the final rule set, RS. The process just described is repeated for each SC.

A more systematic explanation of the entire procedure is:

3.3.1 Split Criteria

In Information Theory, Entropy measures the homogeneity of samples. Completely homogeneous samples have the minimum possible entropy of 0, whereas equally divided samples #1-4 between two labels have the entropy of 1. For any subset of data S, it is evaluated by:

Entropy(S) = $-\sum_{l \in Labels(S)} p^{(l)} log_2 p^{(l)}$

Information Gain (IG) is another popular metric to decide the best split. Assuming a split divides S into {S1, ..., SK},

IG= Entropy(S) -$\sum_{k=1}^{K} w_k$ Entropy(S_k)
where ($w_k = \frac{|S_k|}{S}$)

Gini(P) = $\sum_{i=1}^{n} w_k(1 - w_k)$ = 1- $\sum_{i=1}^{n} {w_k}^2$
where ($w_k = \frac{|S_k|}{S}$)

The Gini Index is another function that can be used as an impurity function. It is a variation of the Gini Coefficient, which is used in economics to measure the dispersion of wealth in a population

3.4 Support Vector Machines

Support vector machine (SVM) is a kernel based novel pattern classification method that is significant in many areas like data mining and machine learning. A unique strength is the use of kernel function to map the data into a higher dimensional feature space. In training SVM, kernels and its parameters have very vital role for classification accuracy. Therefore, a suitable kernel design and its parameters should be used for SVM training. Support Vector Machine (SVM) has been a very successful supervised algorithm for solving twoclass and multi-class recognition problems. For our problem, we present certain kernel functions for multiclass support vector machines and propose the appropriate and optimal kernel for one-versus-all (OAA) multi-class support vector machines. SVM is usually implemented through the so-called **soft-margin** SVM because of its attractive characteristics listed as follows:

(1) It possesses good generality under the principle of structural risk minimization;
(2) It can deal with non-linear problems by kernel methods;
(3) It is robust to noisy instances after introducing slack variables;
(4) It produces sparse solutions since the optimal hyper-plane depends only on support vectors;
(5) It guarantees convergence.

Robust performance of SVM requires a properly adjusted parameter in kernel functions, such as σ in the Gaussian radial basis function (RBF) and the degree in the polynomial kernel.

3.4.1 Soft-Margin Support Vector Machine

This section reviews the main ideas behind the soft-margin SVM. For our problem, we limit the use of SVM to the soft-margin SVM classification. We now describe kernel methods briefly. **Kernel methods** are useful for the successful application of SVM. Kernel methods are a set of approaches to mapping data in the original feature space into the kernel space without ever knowing the mapping function F explicitly. Kernel functions define inner product spaces (Hilbert spaces) in the following way:

$K(x_i, x_j) = < \varphi(x_i), \varphi(x_j) >$

Powered by a proper kernel, SVM is enabled to deal with not only linear problems but also non-linear problems. In Section 3, several classical kernels and their normalized versions are introduced. Given a training dataset U containing N instance-label pairs {xi , yi }, where $y_i \in \{1, -1\}$ represents labels of two classes. SVM seeks an optimal hyper-plane $f(\varphi(x_i)) = w^T \varphi(x) + b = 0$ in the kernel space by maximizing the margin width between $f(\varphi(x_i)) = \pm 1$, where **w** is a weight vector, and **b** is a scalar. The margin width equals 2/||w|| in SVM. The goal of maximizing the margin width is equivalent to the following optimization problem:

$$w^*, b^*, \varepsilon^* = \arg min_{w,b} (\frac{1}{2} ||w||^2 + C \sum_{i=1}^{N} \varepsilon_i)$$

subject to

$$y_i.f\big(\varphi(x_i)\big) \geq 1-\varepsilon_i; \varepsilon_i \geq 0; C > 0; \\ w \epsilon R^n; i = 1,2,..n$$

where C is the regularization parameter; and ε_i is the slack variable that is introduced for instances violating the edges of the margin. The optimization problem is further transformed to the following equivalent dual problem:

$$\alpha^* = \arg max_\alpha L(\alpha) = \\ \sum_{i=1}^{N} \alpha_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\big[\alpha_i \alpha_j y_i y_j K(x_i, x_j)\big]$$

Subject to $\sum_{i=1}^{N}(y_i \alpha_i) = 0; 0 \leq \alpha_i \leq C; \alpha = \{\alpha_i\}^N; i = 1,2..N$

After α* is available by solving above equation, w* and b* are computed by:

$$w^* = \sum_{i=1}^{N}[{\alpha_i}^* y_i \varphi(x_i)] = \sum_{t=1}^{P}[{\alpha_t}^* y_t \varphi(x_t)]$$

$$b^* = \frac{1}{p}\sum_{t=1}^{P}[y_t - {\alpha_t}^* y_t K(x_t, x_t)]$$

where p is the total number of non-zero elements in a*. Those instances that have non-zero elements in a* are called support vectors. So p is the number of support vectors. The predicted label for an unknown instance x is determined by:

$$\hat{y} = f(x) = Sign(\sum_{t=1}^{P}[{\alpha_t}^* y_t K(x_t, x_t)] \\ + \frac{1}{p}\sum_{t=1}^{P}[y_t - {\alpha_t}^* y_t K(x_t, x_t)])$$

3.4.2 Kernel Functions and Similarity Measures

Kernel function is the most important component in kernel based techniques,such as SVM. Several kernel functions are developed according to the Hilbert-Schmidt theory and Mercer condition. The three commonly used kernels are Linear , Gaussian and Polynomial kernel. For our Multi-class model we implemented Parameter tuning to find the best kernel function suitable for our model.

4 Experimental Results

4.1 Explanation of Dataset

4.1.1 Origin of Dataset

The data has been obtained from the data science community Kaggle where the real-time traffic data has been collected using several data providers, including two APIs which provide streaming traffic event data which makes predicting traffic accidents more realistic. It is a real-time traffic data which has been collected between the years 2016-2019.

4.1.2 Features and Observations

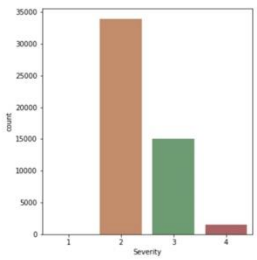
The dataset consists of 49 features and 50000 observations. The features consist of factors like geographical information, accident time, weather conditions and other accident-related statistics and findings. The below image consists of datatype of each feature.

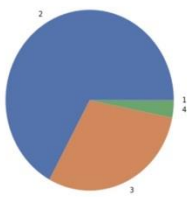
```
ID                  : Factor w/ 50564 levels "A-1000012","A-100003",..:
Source              : Factor w/ 3 levels "Bing","MapQuest",..: 2 1 1 2 1
TMC                 : num  201 NA NA 201 NA 241 241 201 NA 201 ...
Severity            : int  2 3 2 2 2 3 2 3 2 2 ...
Start_Time          : Factor w/ 50460 levels "2016-02-09 07:41:01",..: 1
End_Time            : Factor w/ 50494 levels "2016-02-09 08:11:01",..: 1
Start_Lat           : num  30.3 30 41.8 40.1 44.4 ...
Start_Lng           : num  -97.7 -90.1 -87.9 -75.1 -123.8 ...
End_Lat             : num  NA 30 41.8 NA 44.4 ...
End_Lng             : num  NA -90 -87.9 NA -123.8 ...
Distance.mi.        : num  0 1.57 0 0 1.67 ...
Description         : Factor w/ 46946 levels "# 2 lane blocked due to ac
5 32498 37576 10518 31588 23377 ...
Number              : num  101 NA NA NA 23843 ...
Street              : Factor w/ 17437 levels " 108th St NE",..: 16023 75
Side                : Factor w/ 2 levels "L","R": 1 2 2 2 1 2 2 2 2 2 ..
City                : Factor w/ 4516 levels "","Abbeville",..: 166 2767
County              : Factor w/ 1014 levels "Abbeville","Acadia",..: 923
State               : Factor w/ 48 levels "AL","AR","AZ",..: 41 17 13 30
Zipcode             : Factor w/ 20004 levels "","01005-9392",..: 14890 1
Country             : Factor w/ 1 level "US": 1 1 1 1 1 1 1 1 1 1 ...
Timezone            : Factor w/ 5 levels "","US/Central",..: 2 5 2 3 5 3
Airport_Code        : Factor w/ 1353 levels "","K04W","K0CO",..: 108 110
Weather_Timestamp   : Factor w/ 40397 levels "","2016-02-09 07:58:00",..
Temperature.F.      : num  84 77 80.1 53 49 46.9 45 82 59 83 ...
Wind_Chill.F.       : num  NA 77 NA 53 49 NA 37.6 NA NA 83 ...
Humidity...         : num  67 96 43 86 97 44 53 54 93 82 ...
Pressure.in.        : num  29.9 29.9 30.1 30 29.8 ...
Visibility.mi.      : num  10 10 10 4 10 10 10 10 8 10 ...
Wind_Direction      : Factor w/ 25 levels "","Calm","CALM",..: 12 3 6 4
Wind_Speed.mph.     : num  6.9 0 10.4 9 3 9.2 17.3 8.1 3.5 9 ...
Precipitation.in.   : num  NA 0 NA 0 0 NA NA NA NA 0 ...
Weather_Condition   : Factor w/ 66 levels "","Blowing Dust / Windy",..:
Amenity             : Factor w/ 2 levels "False","True": 1 1 1 1 1 1 1 1
Bump                : Factor w/ 2 levels "False","True": 1 1 1 1 1 1 1 1
Crossing            : Factor w/ 2 levels "False","True": 1 1 1 2 1 1 1 1
Give_Way            : Factor w/ 2 levels "False","True": 1 1 1 1 1 1 1 1
Junction            : Factor w/ 2 levels "False","True": 1 2 1 1 1 1 1 2
No_Exit             : Factor w/ 2 levels "False","True": 1 1 1 1 1 1 1 1
Railway             : Factor w/ 2 levels "False","True": 1 1 1 1 1 1 1 1
Roundabout          : Factor w/ 2 levels "False","True": 1 1 1 1 1 1 1 1
Station             : Factor w/ 2 levels "False","True": 1 1 1 1 1 1 1 1
Stop                : Factor w/ 2 levels "False","True": 1 1 1 1 1 1 1 1
Traffic_Calming     : Factor w/ 2 levels "False","True": 1 1 1 1 1 1 1 1
Traffic_Signal      : Factor w/ 2 levels "False","True": 1 1 2 1 1 2 1 1
Turning_Loop        : Factor w/ 1 level "False": 1 1 1 1 1 1 1 1 1 1 ...
Sunrise_Sunset      : Factor w/ 3 levels "","Day","Night": 3 3 2 2 3 3 2
Civil_Twilight      : Factor w/ 3 levels "","Day","Night": 3 3 2 2 3 3 2
Nautical_Twilight   : Factor w/ 3 levels "","Day","Night": 2 3 2 2 3 3 2
Astronomical_Twilight: Factor w/ 3 levels "","Day","Night": 2 3 2 2 3 3 2
```

4.1.3 Label Imbalance

Since the data has been collected in real-time, the accidents causing least impact on traffic has been less reported when compared to more severe impact accidents, hence the data is highly imbalanced. The target feature is 'Severity' which has four categories 'Severity1','Severity2','Severity3''Severity4', with Severity1 showing the least impact of accidents on traffic and Severity4 showing the highest impact of accidents on traffic. The below shows the distribution of target variable.

Severity 1	Severity 2	Severity 3	Severity 4
14	1560	15081	33909





4.2 Machine Learning Types and Approaches

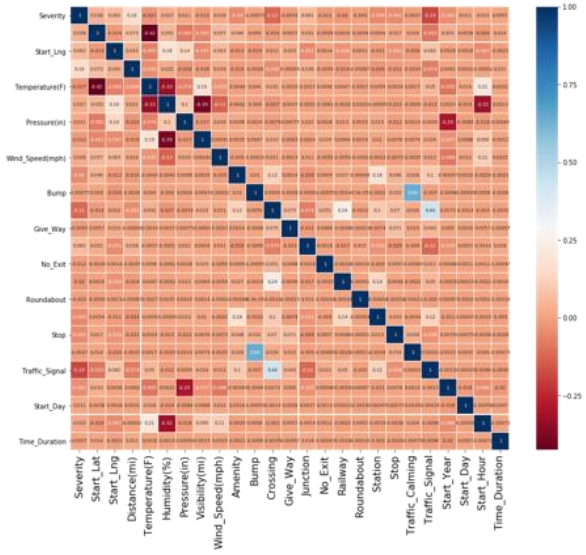
In this report, we'll be solving the problem statement as a supervised machine learning type (the task of learning a function that maps an input to an output based on example input-output pairs). Supervised learning type can be of two types – Classification and Regression. We will be following the Classification Supervised Learning Approach in our prediction. We've used four classification machine learning models in our analysis. The KNN classifier has been used as our baseline model.

KNN	LOGISTIC REGRESSION	DECISION TREE	SVM
BASELINE MODEL	MODEL 1	MODEL 2	MODEL 3

4.3 Experimental Processes

4.3.1 Derived Features

From the StartTime and EndTime features, the Year, Month, Day and Hour of the accident are derived as features. Also, Time Duration (in min) and Time Duration (in sec) are derived by finding the difference in time between End Time and Start Time of an accident. We then found the correlation of these features with other features by using correlation heat map.



4.3.2 Missing Value Treatment

The features that had more than 30000 missing

values out of 50000 were removed.

Variables Removed

Wind_Chill(F) – Wind Chill at time of accident

Precipitation – Precipitation in air at time of accident

End_Lat – End Latitude value of the accident location

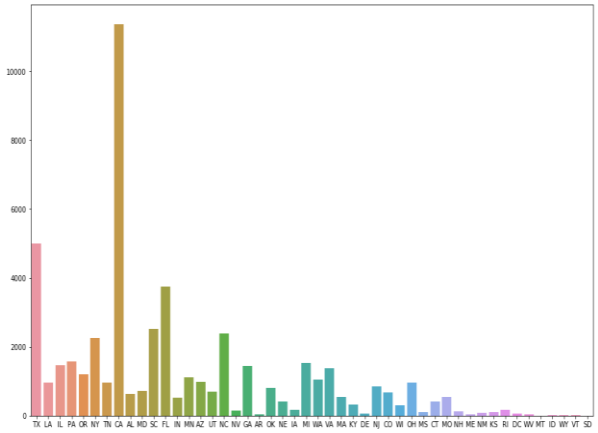
End_Lon - End Longitude value of the accident location

Number – Street Number of the accident location

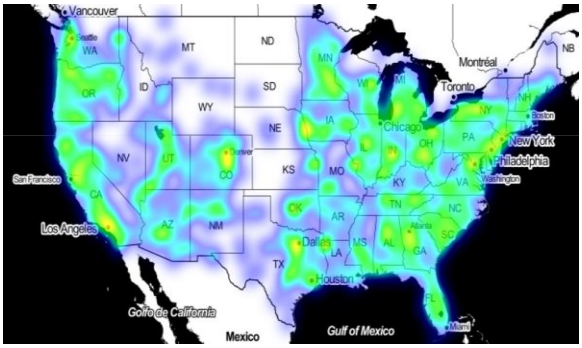
Mean and Mode imputation were done for continuous and categorical features respectively that had missing values fewer than 30000, typically ranging from 36 to 12000.

4.3.3 Identifying the hotspot locations

We did an extensive analysis in identifying the hotspot locations and we found that California had more accidents followed by Texas and New York. The below image shows the distribution of accidents in the 49 states

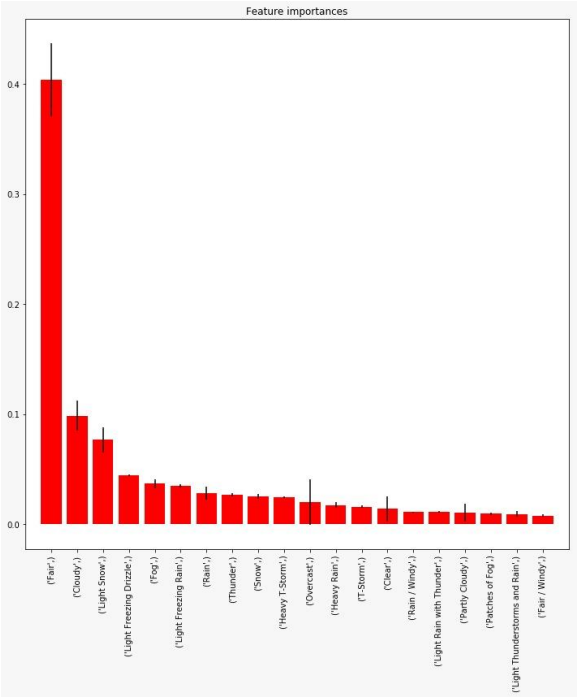


We used the technique of heat map to further identify the accident-prone zones in these states which can be seen below



4.3.4 Identifying the weather factors causing the road accidents

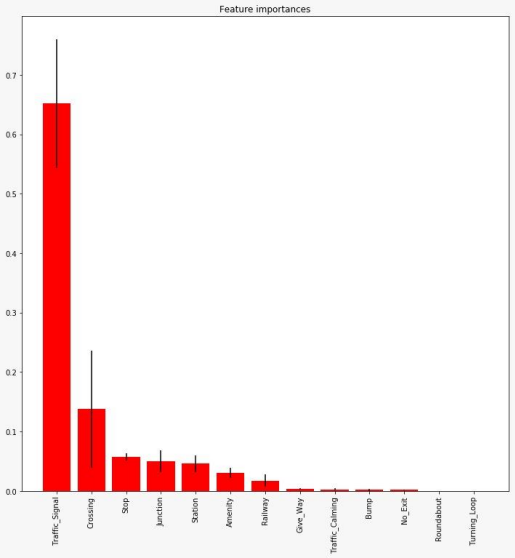
The weather factors that cause majority of accidents can be seen in the below plot.



From the above plot, we can see that Fair and Cloudy weather causes the most accidents. This can be neglected as it does not provide any information regarding the weather conditions that cause accidents and could attribute to the fact that the weather has fair or cloudy weather on most days. Hence, we can find that weather conditions like ‘Light Snow’ followed by ‘Light Freezing drizzle’ and ‘Fog’ are the major

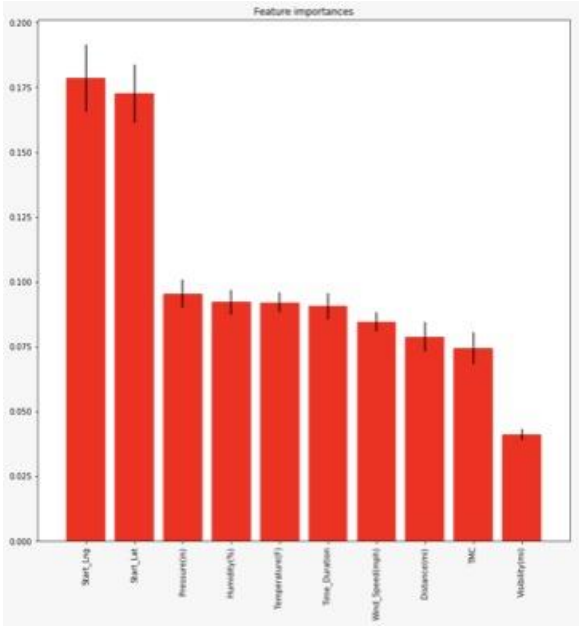
weather factors that contribute to road accidents in the US.

4.3.5 Identifying the road factors causing the accidents



From the above plot, we can see that Traffic Signal, Crossing, a Stop and Junction are the primary factors causing an accident with Traffic signals being the strongest influencer.

4.3.6 Identifying other factors causing the accidents



From the above plot, we can see that the Start Longitude and End Longitude which is basically the location of the accident followed by Pressure, Humidity, Temperature are other environmental factors contributing to road accidents in the US.

4.3.7 Baseline Model

We chose the KNN classifier as our baseline model and to achieve better results we used three classification models Decision Tree, Multinomial Logistic Regression and SVM.

4.3.4 Model Improvement

1. Multinomial Logistic Regression

2. Decision Tree

3. SVM

To improve the performance of the decision tree, parameter tuning was performed using K-fold cross validation using GridSearch functionality. The pseudocode for the Grid Search can be seen below.

```
for c in [2^-5, 2^-3,...,2^15]:
 for g in [2^-15, 2^-13, ..., 2^3]:
  for train, test in partition:
    model = svm_train(train, c, g)
    score = svm_predict(test, model)
    cv_list.insert (score)
  scores_list.insert(mean(cv_list),c,g)
print max(scores_list)
```

Pseudocode for Cross Validation

Nested k-fold Cross-Validation

1. Define set of hyper-parameter combinations, ***C***, for current model. If model has no hyper-parameters, ***C*** is the empty set.
2. Divide data into *K* folds with approximately equal distribution of cases and controls
3. (**outer loop**) For fold k_i in the *K* folds:
    1. Set fold k_i as the test set
    2. Perform automated feature selection on the remaining *K-1* folds
    3. For parameter combination c in ***C***:
        1. (**inner loop**) For fold k_j in the remaining *K-1* folds:
            1. Set fold k_j as the validation set
            2. Train model on remaining *K-2* folds
            3. Evaluate model performance on fold k_j
        2. Calculate average performance over *K-2* folds for parameter combination *c*
    4. Train model on *K-1* folds using hyper-parameter combination that yielded best average performance over all steps of the **inner loop**
    5. Evaluate model performance on fold k_i.
4. Calculate average performance over *K* folds

Parameters tuned for Decision Tree
Max Depth of tree and criteria for splitting attribute parameters were tuned using k-fold cross validation(k=5). The optimal parameters were

- **Depth of Tree - 5**
- **Criteria – 'gini'**

Parameters tuned for SVM
The kernel to be used in SVM, the cost of misclassification parameter C and Kernel parameter Gamma were tuned using k-fold cross validation. The optimal parameters were

- **Kernel – RBF kernel**
- **Value of C – 1000**
- **Value of Gamma – e-08**

Multinomial Logistic Regression, Decision Tree and SVM were then trained with the optimal parameters using Scikit-Learn Library in Python.

4.4 Evaluation Metric, Model Output & Error Analysis

Since the data has imbalance in labels, accuracy cannot be considered as an evaluation metric. Also, in this scenario of predicting the severity of accidents on traffic both precision and recall are equally important. Hence, we considered the following four evaluation metrics.

Precision: This evaluation measure talks about how precise/accurate your model is out of those predicted positive. Precision is preferred as an evaluation metric when the costs of false positive is high.

$$Precision = \frac{True\ Positive}{True\ Positive + False\ Positive}$$

Recall: This evaluation measure calculates how many of the Actual Positives a model captures through labelling it as True Positive. Recall is preferred as an evaluation metric when the costs of false negative is high.

$$Recall = \frac{True\ Positive}{True\ Positive + False\ Negative}$$

F1-Score: When there is a high cost associated with false positive and false negative, a measure, F1 Score is used which is a function of Precision and Recall. In other words, F1 score is a good choice if we need to seek a balance between precision and recall and if there is uneven class distribution.

$$F1 = 2 * \frac{Precision * Recall}{Precision + Recall}$$

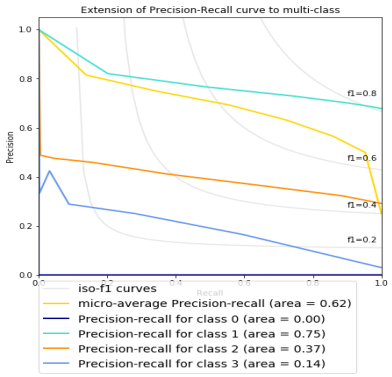
PR Curve – AUC / Average Precision: The PR Curve is a curve that combines precision (PPV) and Recall (TPR) in a single visualization. For every threshold, you calculate PPV and TPR and plot it. The higher on y-axis your curve is the better your model performance. The PR Curve is used when we want to achieve high precision and high recall.

The four models were evaluated using these four metrics and the results obtained on test set can be seen as below

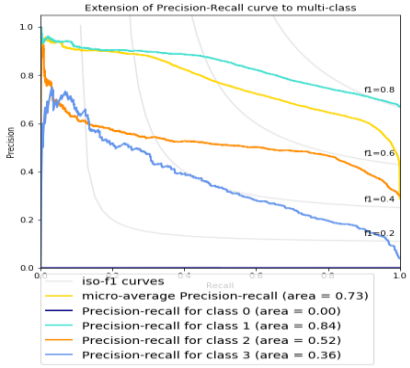
	BASELINE MODEL - KNN	MULTINOMIAL LOGISTIC REGRESSION	DECISION TREE	SVM
PRECISION	67	83	83	100
RECALL	69	69	70	67
F1 SCORE (WEIGHTED)	72.3	74	76.1	80.3
PR-AUC(AVG)	62	73	74	71

The PR-AUC Curves for the four classes of the four models can be seen below

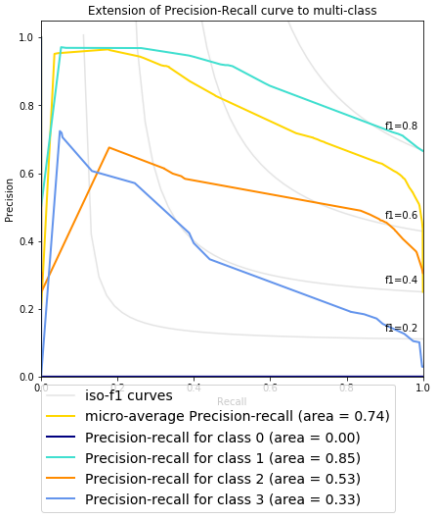
KNN Classifier



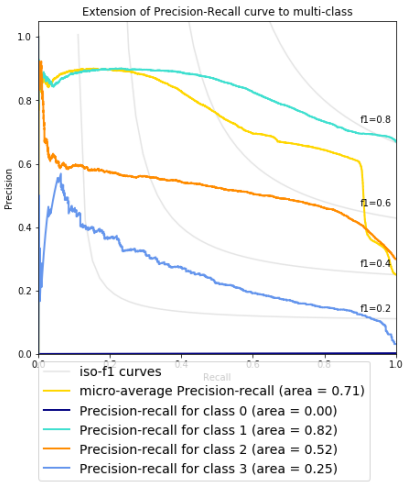
Multinomial logistic Regression



Decision Tree

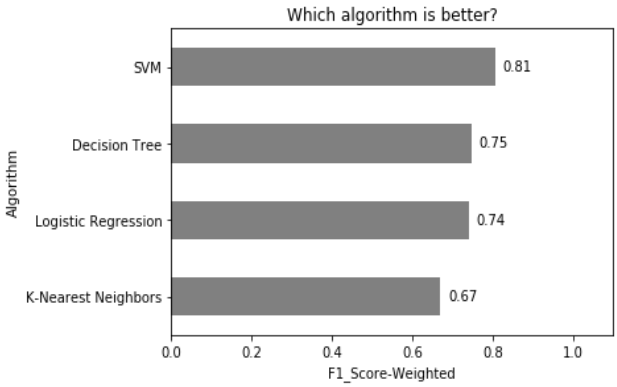


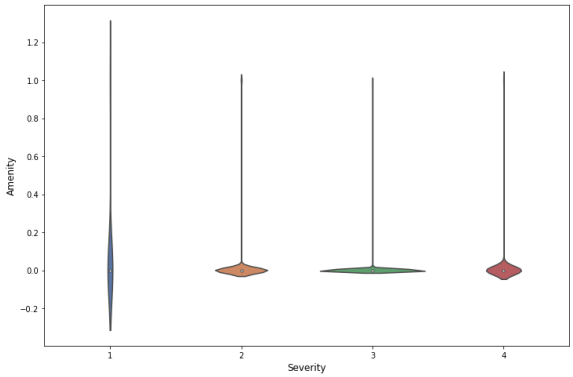
SVM

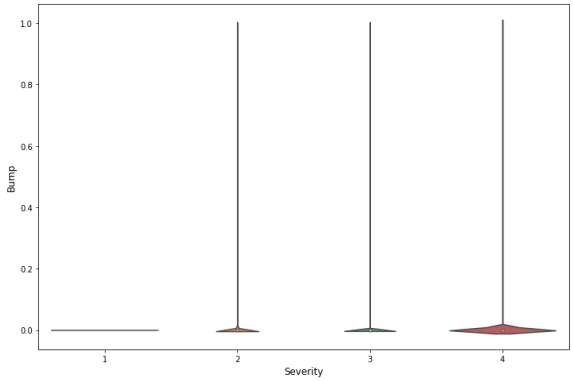


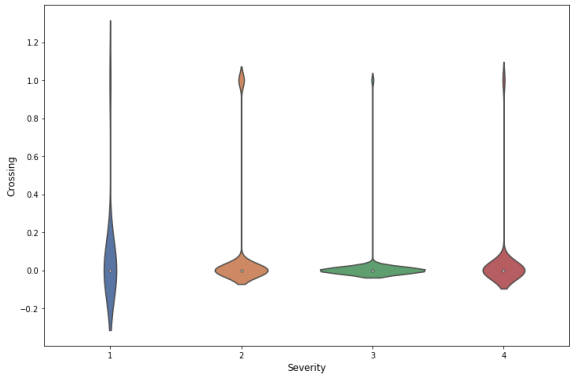
Label Imbalance Problem

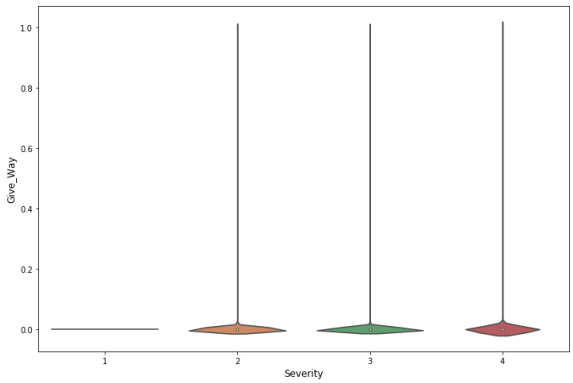
Though these four evaluation metrics focuses on achieving high precision and high recall , only **weighted F1-Score** accounts for the label imbalance problem in our dataset, by taking into account the no of samples present in each class into account. Hence, we've chosen F1-Score as our final evaluation metric. Considering which SVM performed better than the other models and achieved a weighted F1-Score of 80% on the test set. The below plot shows the comparison of four models in terms of weighted F1-Score.











Conclusion

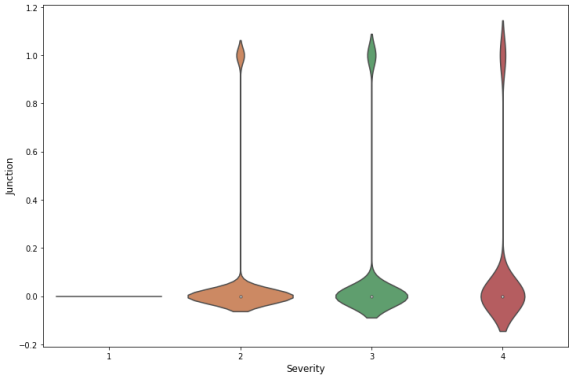
In this report we investigated the problem of real-time prediction of severity of car accidents on traffic using real-time traffic data between 2016 to 2019, addressing an important problem to transportation and public safety. This problem is very challenging due to the imbalanced classes, the spatial heterogeneity and its non-linear separable nature. We formulated the problem as a multi-class classification problem. In this report, we have also successfully identified the hotspot locations of accidents, various factors accounting for these accidents and the average time is taken to clear the accidents. We developed several classification machine learning models and evaluated the performance of these models using various evaluation metrics. Results show that our proposed approach (SVM) had a significantly improved weighted F1-Score of 80% when compared to our baseline model KNN which had an F1-Score of 67%.

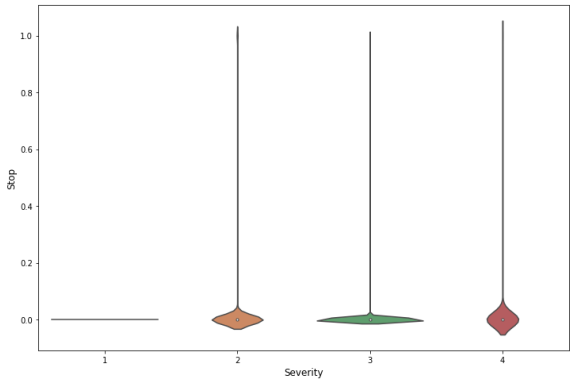
For future work, we would like to explore advanced machine learning techniques such as ensemble methods, XGBoost and Neural Networks to predict the severity of traffic on accidents in real-time. We also plan to investigate approaches to predict casualty analysis.

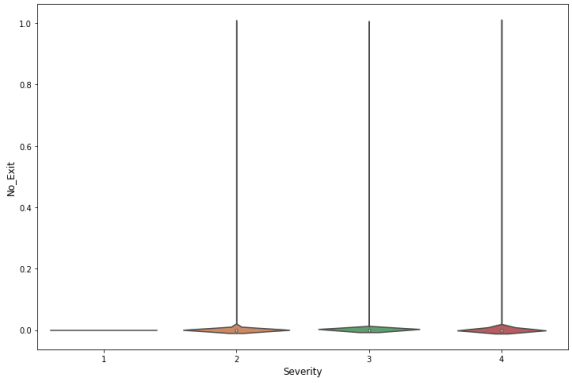
Appendix

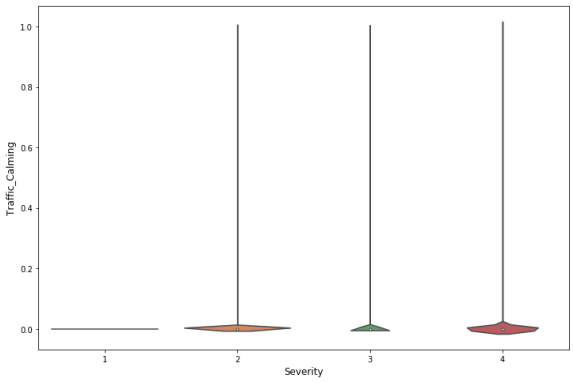
EDA Plots

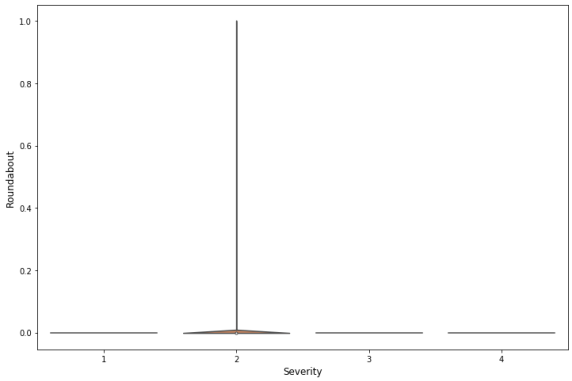
Factors of Road System vs Severity of Car Accidents on Traffic

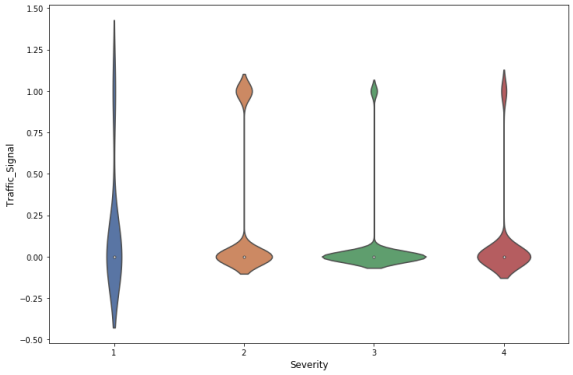


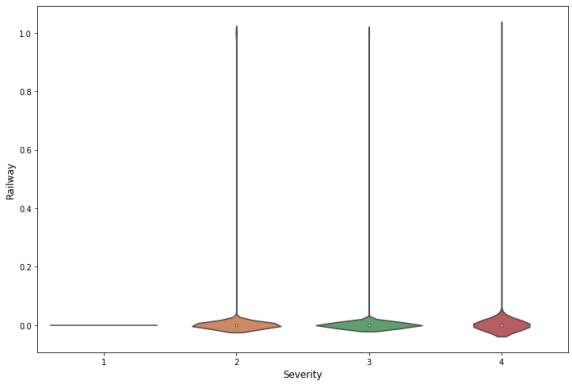




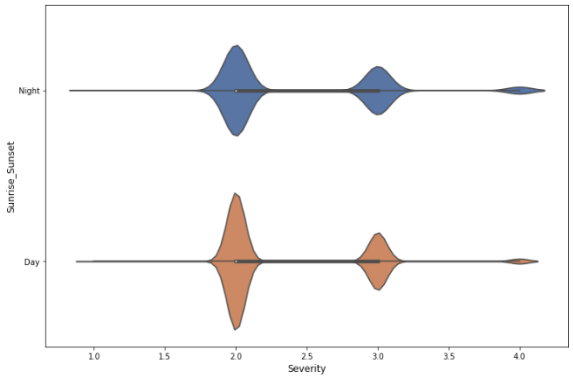


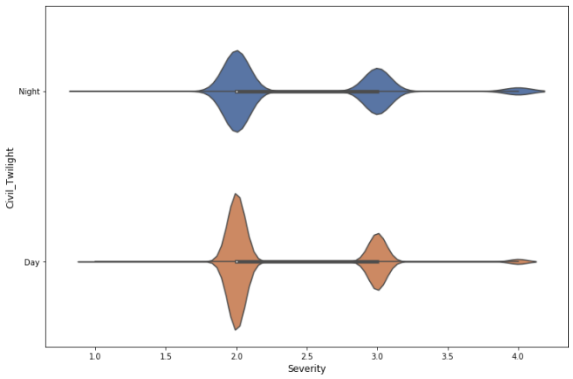


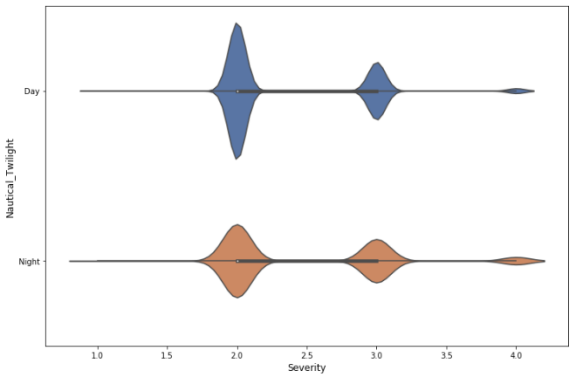


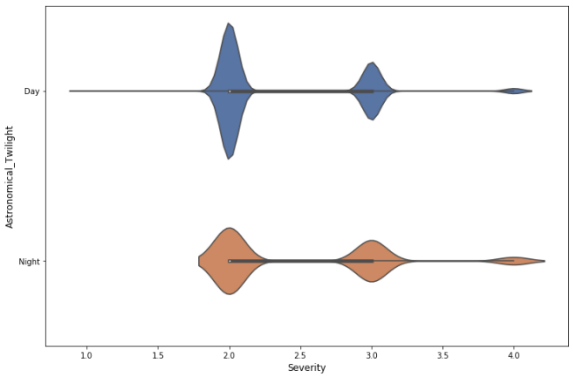


Accident Severity vs Days & Nights

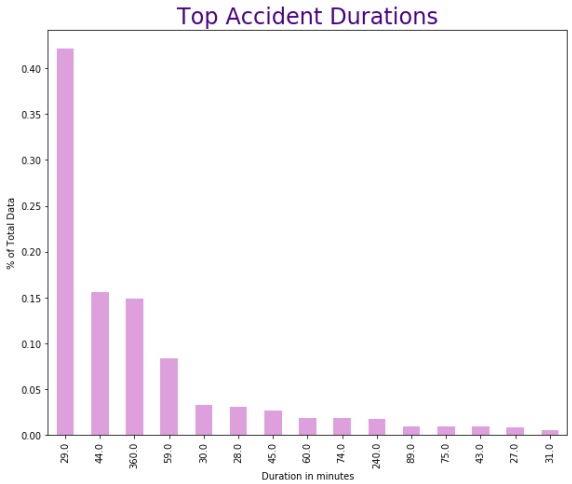








Time taken to clear accidents vs %of Total Data



Source Code

Code for Parameter Tuning – decision tree

#PARAMETER TUNING FOR DECISION TREE

```
df=df_nona
import warnings
warnings.filterwarnings("ignore")
from sklearn.metrics import f1_score
from sklearn.metrics import make_scorer
# load libraries
from sklearn import decomposition, datasets
from sklearn import tree
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV, cross_val_score
from sklearn.preprocessing import StandardScaler


target='Severity'


# Create arrays for the features and the response variable

# set X and y
y = df[target]
X = df.drop(target, axis=1)
```

```
    # Load the iris flower data
    #dataset = datasets.load_iris()
    #X = dataset.data
    #y = dataset.target

    # Create an scaler object
    sc = StandardScaler()

    # Create a pca object
    #pca = decomposition.PCA()

    # Create a logistic regression object with an
L2 penalty
    decisiontree = tree.DecisionTreeClassifier()

    # Create a pipeline of three steps. First,
standardize the data.
    # Second, tranform the data with PCA.
    # Third, train a Decision Tree Classifier on
the data.
    pipe = Pipeline(steps=[('sc', sc),
                           ('decisiontree', decisiontree)])

    # Create Parameter Space
    # Create a list of a sequence of integers from
1 to 30 (the number of features in X + 1)
    #n_components =
list(range(1,X.shape[1]+1,1))

    # Create lists of parameter for Decision Tree
Classifier
    criterion = ['gini', 'entropy']
    max_depth = [1,2,3,4,5]

    # Create a dictionary of all the parameter
options
    # Note has you can access the parameters of
steps of a pipeline by using '__'
    parameters =
dict(decisiontree__criterion=criterion,

decisiontree__max_depth=max_depth)

    # Conduct Parameter Optmization With
Pipeline
    # Create a grid search object
    clf = GridSearchCV(pipe, parameters)

    # Fit the grid search
    clf.fit(X, y)

    # View The Best Parameters
    print('Best Criterion:',
clf.best_estimator_.get_params()['decisiontree__
criterion'])
    print('Best max_depth:',
clf.best_estimator_.get_params()['decisiontree__
max_depth'])
    #print('Best Number Of Components:',
clf.best_estimator_.get_params()['pca__n_comp
onents'])
    print();
print(clf.best_estimator_.get_params()['decisiont
ree'])

    # Use Cross Validation To Evaluate Model
    CV_Result = cross_val_score(clf, X, y, cv=5,
n_jobs=-
1,scoring=make_scorer(f1_score,average='weig
hted'))
    print(); print(CV_Result)
    print(); print(CV_Result.mean())
    print(); print(CV_Result.std())
```

Code for importing libraries

```
# Import KNeighborsClassifier from
sklearn.neighbors
from sklearn.neighbors import
KNeighborsClassifier

# Import DecisionTreeClassifier from
sklearn.tree
from sklearn.tree import DecisionTreeClassifier

from sklearn.linear_model import
LogisticRegression

from sklearn.model_selection import
train_test_split
from sklearn.model_selection import
GridSearchCV
from sklearn.feature_selection import
SelectFromModel
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_curve, auc
```

Code for KNN Classifier

```
# Binarizing labels and Training - KNN
from sklearn.preprocessing import
```

```
label_binarize
from sklearn.multiclass import
OneVsRestClassifier
y = label_binarize(y, classes=[0,1,2,3])
n_classes = y.shape[1]
xcl_train,xcl_test,ycl_train,ycl_test=train_test_s
plit(X,y,test_size=0.3)
finalKNN=OneVsRestClassifier(KNeighborsCla
ssifier(n_neighbors=6))
finalKNN.fit(xcl_train,ycl_train)

#Model prediction
pr_train=finalKNN.predict(xcl_train)
pr=finalKNN.predict(xcl_test)
y_score_train=finalKNN.predict_proba(xcl_trai
n)
y_score=finalKNN.predict_proba(xcl_test)

f1_train=metrics.f1_score(pr_train,ycl_train,ave
rage="weighted")
print('Weighted Training F1-Score - KNN:
{0:0.2f}'.format(f1_train))
f1_test=metrics.f1_score(pr,ycl_test,average="w
eighted")
print('Weighted Test F1-Score - KNN:
{0:0.2f}'.format(f1_test))
f1knn=f1_test
fscore_lst.append(f1knn)
precision_train=metrics.precision_score(pr_train
,ycl_train,average="weighted")
print('Weighted Training Precision - KNN:
{0:0.2f}'.format(precision_train))
precision_test=metrics.precision_score(pr,ycl_te
st,average="weighted")
print('Weighted Test Precision - KNN:
{0:0.2f}'.format(precision_test))
recall_train=metrics.recall_score(pr_train,ycl_tr
ain,average="weighted")
print('Weighted Training Recall - KNN:
{0:0.2f}'.format(recall_train))
recall_test=metrics.recall_score(pr,ycl_test,aver
age="weighted")
print('Weighted Test Recall - KNN:
{0:0.2f}'.format(recall_test))
```

Code for Logistic Regression

```
# Binarizing labels and Training - Logistic
regression
from sklearn.preprocessing import
label_binarize
from sklearn.multiclass import
OneVsRestClassifier
y = label_binarize(y, classes=[0,1,2,3])
n_classes = y.shape[1]
xcl_train,xcl_test,ycl_train,ycl_test=train_test_s
plit(X,y,test_size=0.3)
finallogreg=OneVsRestClassifier(LogisticRegre
ssion(random_state=0))
finallogreg.fit(xcl_train,ycl_train)

#Model prediction
pr_train=finallogreg.predict(xcl_train)
pr=finallogreg.predict(xcl_test)
y_score_train=finallogreg.predict_proba(xcl_trai
n)
y_score=finallogreg.predict_proba(xcl_test)

f1_train=metrics.f1_score(pr_train,ycl_train,ave
rage="weighted")
print('Weighted Training F1-Score - Logistic
regression: {0:0.2f}'.format(f1_train))
f1_test=metrics.f1_score(pr,ycl_test,average="w
eighted")
print('Weighted Test F1-Score - Logistic
regression: {0:0.2f}'.format(f1_test))
f1log=f1_test
fscore_lst.append(f1log)
precision_train=metrics.precision_score(pr_train
,ycl_train,average="weighted")
print('Weighted Training Precision - Logistic
regression: {0:0.2f}'.format(precision_train))
precision_test=metrics.precision_score(pr,ycl_te
st,average="weighted")
print('Weighted Test Precision - Logistic
regression: {0:0.2f}'.format(precision_test))
recall_train=metrics.recall_score(pr_train,ycl_tr
ain,average="weighted")
print('Weighted Training Recall - Logistic
regression: {0:0.2f}'.format(recall_train))
recall_test=metrics.recall_score(pr,ycl_test,aver
age="weighted")
print('Weighted Test Recall - Logistic
regression: {0:0.2f}'.format(recall_test))
```

Code for Decision Tree

```
# Binarizing labels and Training - Decision Tree
from sklearn.preprocessing import
label_binarize
from sklearn.multiclass import
OneVsRestClassifier
```

```
y = label_binarize(y, classes=[0,1,2,3])
n_classes = y.shape[1]
xcl_train,xcl_test,ycl_train,ycl_test=train_test_s
plit(X,y,test_size=0.3)
finalDT=OneVsRestClassifier(DecisionTreeClas
sifier(max_depth=5, criterion='gini',
random_state=1))
finalDT.fit(xcl_train,ycl_train)

#Model prediction
pr_train=finalDT.predict(xcl_train)
pr=finalDT.predict(xcl_test)
y_score_train=finalDT.predict_proba(xcl_train)
y_score=finalDT.predict_proba(xcl_test)

f1_dt_train=metrics.f1_score(pr_train,ycl_train,
average="weighted")
print('Weighted Training F1-Score - Decision
Tree: {0:0.2f}'.format(f1_dt_train))
f1_dt_test=metrics.f1_score(pr,ycl_test,average
="weighted")
print('Weighted Test F1-Score - Decision Tree:
{0:0.2f}'.format(f1_dt_test))
f1log=f1_dt_test
fscore_lst.append(f1log)
precision_train=metrics.precision_score(pr_train
,ycl_train,average="weighted")
print('Weighted Training Precision - Decision
Tree: {0:0.2f}'.format(precision_train))
precision_test=metrics.precision_score(pr,ycl_te
st,average="weighted")
print('Weighted Test Precision - Decision Tree:
{0:0.2f}'.format(precision_test))
recall_train=metrics.recall_score(pr_train,ycl_tr
ain,average="weighted")
print('Weighted Training Recall - Decision Tree:
{0:0.2f}'.format(recall_train))
recall_test=metrics.recall_score(pr,ycl_test,aver
age="weighted")
print('Weighted Test Recall - Decision Tree:
{0:0.2f}'.format(recall_test))
```

Code for SVM

```
# Binarizing labels and Training - SVM
from sklearn import svm
from sklearn.preprocessing import
label_binarize
from sklearn.multiclass import
OneVsRestClassifier
y = label_binarize(y, classes=[0,1,2,3])
n_classes = y.shape[1]
xcl_train,xcl_test,ycl_train,ycl_test=train_test_s
plit(X,y,test_size=0.3)
finalsvm=OneVsRestClassifier(SVC(C=1000,ke
rnel='rbf',gamma=1e-8))
finalsvm.fit(xcl_train,ycl_train)
#Model prediction
pr_train=finalsvm.predict(xcl_train)
pr=finalsvm.predict(xcl_test)
y_score_train=finalsvm.decision_function(xcl_tr
ain)
y_score=finalsvm.decision_function(xcl_test)

f1_train=metrics.f1_score(pr_train,ycl_train,ave
rage="weighted")
print('Weighted Training F1-Score - SVM:
{0:0.2f}'.format(f1_train))
f1_test=metrics.f1_score(pr,ycl_test,average="w
eighted")
print('Weighted Test F1-Score - SVM:
{0:0.2f}'.format(f1_test))
f1svm=f1_test
fscore_lst.append(f1svm)
precision_train=metrics.precision_score(pr_train
,ycl_train,average="weighted")
print('Weighted Training Precision - SVM:
{0:0.2f}'.format(precision_train))
precision_test=metrics.precision_score(pr,ycl_te
st,average="weighted")
print('Weighted Test Precision - SVM:
{0:0.2f}'.format(precision_test))
recall_train=metrics.recall_score(pr_train,ycl_tr
ain,average="weighted")
print('Weighted Training Recall - SVM:
{0:0.2f}'.format(recall_train))
recall_test=metrics.recall_score(pr,ycl_test,aver
age="weighted")
print('Weighted Test Recall - SVM:
{0:0.2f}'.format(recall_test))
```

Code for finding Important Features

```
train_y = df_nona['Severity'].values
x_cols = [col for col in dfroad.columns if col not
in ['Severity'] if dfroad[col].dtype=='bool']
train_col= dfroad[x_cols]

fearture_name = train_col.columns.values

from sklearn import ensemble
```

```
model =
ensemble.ExtraTreesRegressor(n_estimators=25
, max_depth=30, max_features=0.3, n_jobs=-1,
random_state=0)
model.fit(train_col,train_y)

#plot imp
importance = model.feature_importances_
std = np.std([tree.feature_importances_ for tree
in model.estimators_],axis=0)
indices = np.argsort(importance)[::-1][:20]

plt.figure(figsize=(12,12))
plt.title("Feature importances")
plt.bar(range(len(indices)), importance[indices],
color="r", yerr=std[indices], align="center")
plt.xticks(range(len(indices)),
fearture_name[indices], rotation='vertical')
plt.xlim([-1, len(indices)])
plt.show()
```