Accident Severity Prediction using Seattle Accident Dataset

Applied Data Science Capstone Project

IBM - Coursera

Project report

Karthik J Ghorpade

September 2020

Contents

Introduction | Problem Understanding

Photo by **mali maeder** from **Pexels**

Vehicular accidents, no doubt, cause damage to people involved in the collision and also damage to property. The severity of harm/damage caused by such accidents is dependent on a large number of factors. Attempts to study existing data documenting collisions may lead to insights into common factors which may have a part to play in determining the severity (property damage or injury to humans) caused by the accident. Effectively, the problem narrows down to a binary classification of the severity of the collision, based on external features.

The dataset being used is the Seattle Accident dataset provided by Seattle PD, which has collection of collision data from 2004.

Business Problem | Application Perspective

- The insights gained by such a study would help Seattle Police authority in making changes to some of the external factors in reducing the severity of the accident. This could also help the public in being cautious while driving when some common factors may be evident.
- An additional area where this prediction problem could help would be in narrowing down accident prone locations in Seattle.

Data Collection and Exploration

The data being used for this ML based prediction problem is the Seattle Accident Dataset, which is available in a .csv format as 'Data-Collisions.csv'. There are 194673 accident/collision instances have 38 features/columns each, containing recordings of possible factors such as road condition, if the vehicle was speeding, weather, location, junction type etc. One of the columns contains the severity class of the accident – '1' signifying property damage while '2' signifies human injury. This column would serve as the labels in the machine learning binary (class '1' or '2') classification problem.

```
data.columns
```

```
Index(['SEVERITYCODE', 'X', 'Y', 'OBJECTID', 'INCKEY', 'COLDETKEY', 'REPORTNO',
       'STATUS', 'ADDRTYPE', 'INTKEY', 'LOCATION', 'EXCEPTRSNCODE',
       'EXCEPTRSNDESC', 'SEVERITYCODE.1', 'SEVERITYDESC', 'COLLISIONTYPE',
       'PERSONCOUNT', 'PEDCOUNT', 'PEDCYLCOUNT', 'VEHCOUNT', 'INCDATE',
       'INCDTTM', 'JUNCTIONTYPE', 'SDOT_COLCODE', 'SDOT_COLDESC',
       'INATTENTIONIND', 'UNDERINFL', 'WEATHER', 'ROADCOND', 'LIGHTCOND',
       'PEDROWNOTGRNT', 'SDOTCOLNUM', 'SPEEDING', 'ST_COLCODE', 'ST_COLDESC',
       'SEGLANEKEY', 'CROSSWALKKEY', 'HITPARKEDCAR'],
      dtype='object')
```

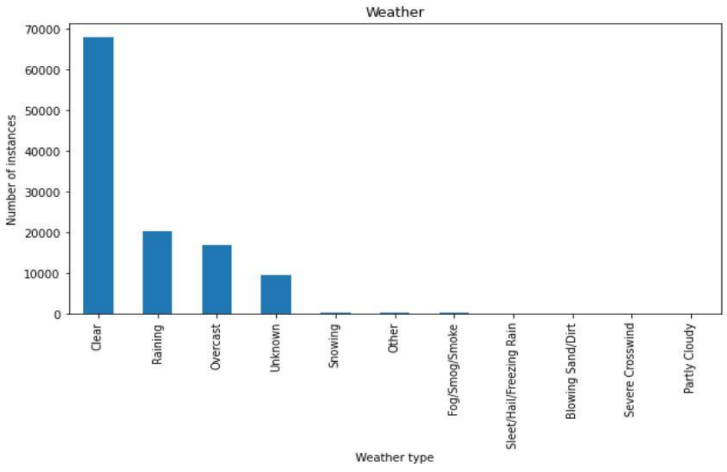
The different columns for each collision instance

As most of the columns here do not provide useful information, only a subset of the columns are chosen for the final ML classification task.

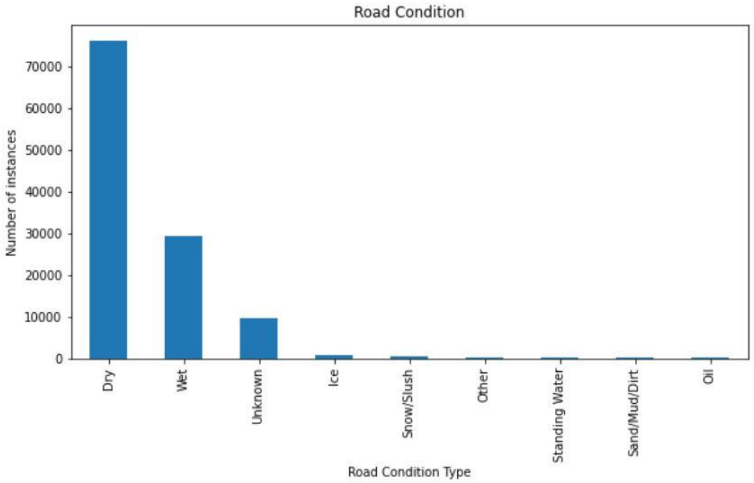
Exploration - Histograms

Observing some of the intuitively important seeming features from the dataset.

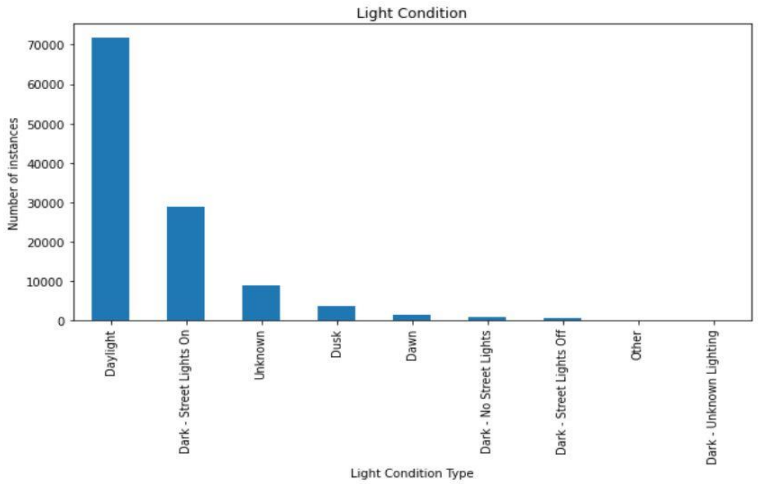
1. Weather Feature:



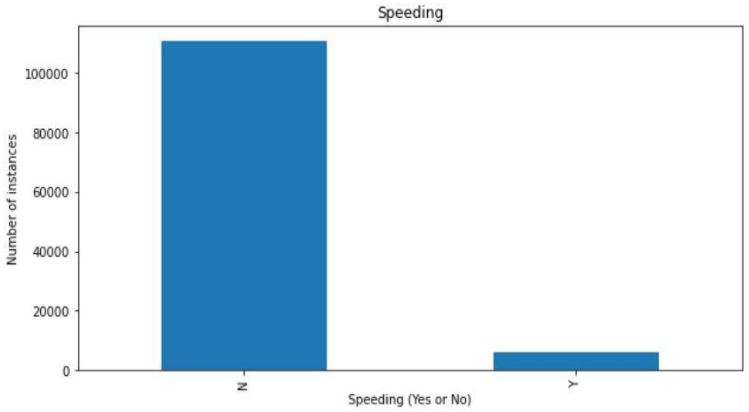
2. Road Condition Feature:



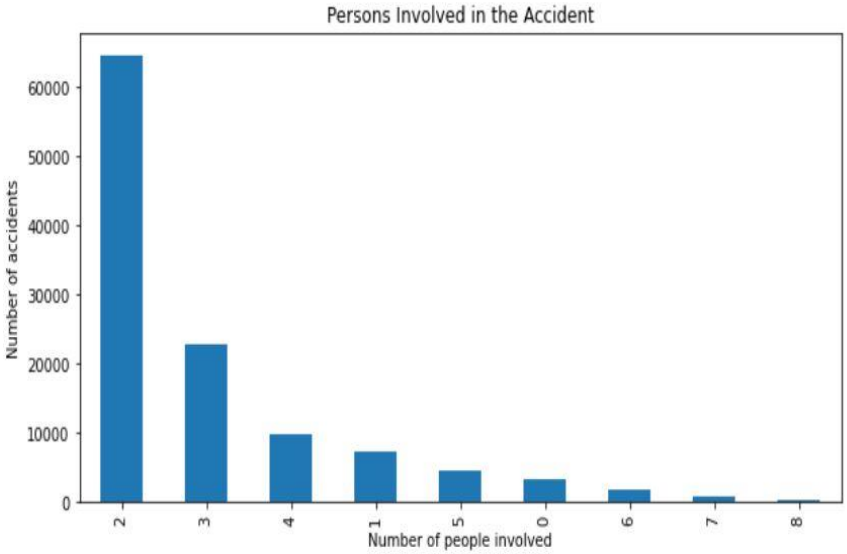
3. Lighting Condition Feature:



4. Speeding Feature:



5. Number of people involved in the collision/accident:



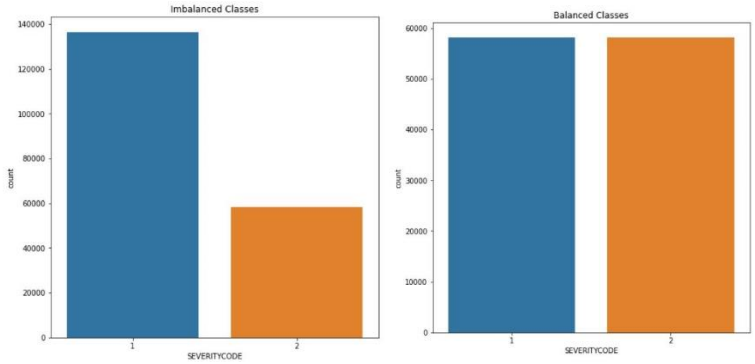
It can be observed that most of the features have a dense concentration over a particular narrow range.

Data Preprocessing – Cleaning and Preparation

Not all of the features in the dataset are useful in the classification problem. The data available in each of the column may also not be present in desired numerical format which can be fed in as input in any classification algorithm. Hence, there is a need for preprocessing (cleaning, labelling and normalizing) of data.

Balancing the Imbalanced Dataset

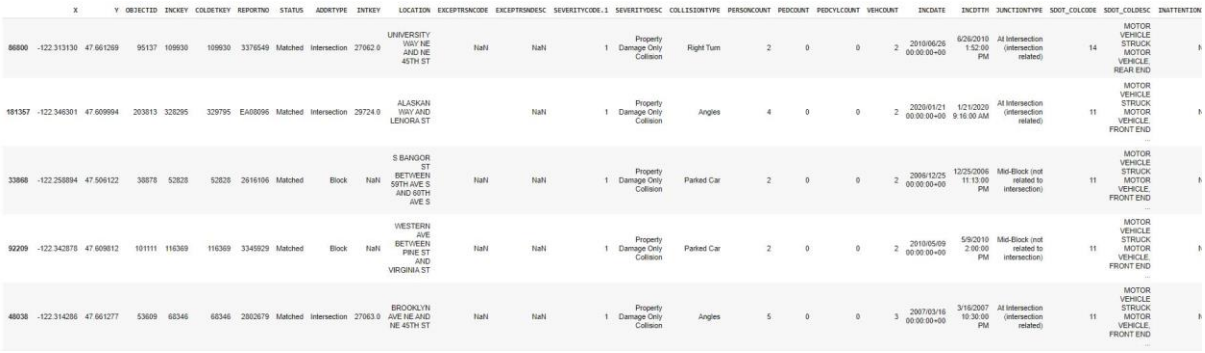
As the two target classes in the dataset are imbalanced



Dataset before and after down-sampling

Data Cleaning

It can be observed that the data has a lot of text and categorical data. These are not suitable to be fed as inputs to any ML classification algorithm.



	X	Y	OBJECTID	INCKEY	COLDETKEY	REPORTNO	STATUS	ADDRTYPE	INTKEY	LOCATION	EXCEPTRSNCODE	EXCEPTRSNDESC	SEVERITYCODE.1	SEVERITYDESC	COLLISIONTYPE	PERSONCOUNT	PEDCOUNT	PEDCYLCOUNT	VEHCOUNT	INCDATE	INCDTTM	JUNCTIONTYPE	SDOT_COLCODE	SDOT_COLDESC	INATTENTION
86800	-122.313130	47.661269	95137	109930	109930	3376549	Matched	Intersection	27062.0	UNIVERSITY WAY NE AND NE 45TH ST	NaN	NaN	1	Property Damage Only Collision	Right Turn	2	0	0	2	2010/06/26 00:00:00+00	6/26/2010 1:52:00 PM	At Intersection (intersection related)	14	MOTOR VEHICLE STRUCK MOTOR VEHICLE, REAR END	
181357	-122.346301	47.609994	203813	328295	329795	EA08096	Matched	Intersection	29724.0	ALASKAN WAY AND LENORA ST		NaN	1	Property Damage Only Collision	Angles	4	0	0	2	2020/01/21 00:00:00+00	1/21/2020 9:16:00 AM	At Intersection (intersection related)	11	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END	
33868	-122.258894	47.506122	38878	52828	52828	2616106	Matched	Block	NaN	S BANGOR ST BETWEEN 59TH AVE S AND 60TH AVE S	NaN	NaN	1	Property Damage Only Collision	Parked Car	2	0	0	2	2006/12/25 00:00:00+00	12/25/2006 11:13:00 PM	Mid-Block (not related to intersection)	11	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END	
92209	-122.342878	47.609812	101111	116369	116369	3345929	Matched	Block	NaN	WESTERN AVE BETWEEN PINE ST AND VIRGINIA ST	NaN	NaN	1	Property Damage Only Collision	Parked Car	2	0	0	2	2010/05/09 00:00:00+00	5/9/2010 2:00:00 PM	Mid-Block (not related to intersection)	11	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END	
48038	-122.314286	47.661277	53609	68346	68346	2802679	Matched	Intersection	27063.0	BROOKLYN AVE NE AND NE 45TH ST	NaN	NaN	1	Property Damage Only Collision	Angles	5	0	0	3	2007/03/16 00:00:00+00	3/16/2007 10:30:00 PM	At Intersection (intersection related)	11	MOTOR VEHICLE STRUCK MOTOR VEHICLE, FRONT END	

i) There are also a lot of 'NaN' values under most columns. Replacing them with suitable values.
e.g.:

```
[ ]  data['WEATHER'].isnull().values.any()
```

```
True
```

```
[ ]  data['WEATHER'].fillna("Unknown", inplace = True)
```

ii) The categorical values are label encoded.
e.g.:

```
[ ]  from sklearn.preprocessing import LabelEncoder
     label_encoder = LabelEncoder()
     label_encoder_weather = label_encoder.fit(data['WEATHER'])
     label_encoded_weather = label_encoder.transform(data['WEATHER'])
```

```
[ ]  data['WEATHER'] = label_encoded_weather
```

iii) Picking the relevant columns/features for the classification problem as many of the columns may not be very useful.

Features chosen

Road condition, Lighting condition, Weather, Speeding, Location, Junction type, Address type, Vehicle count, Person count and Inattention index.

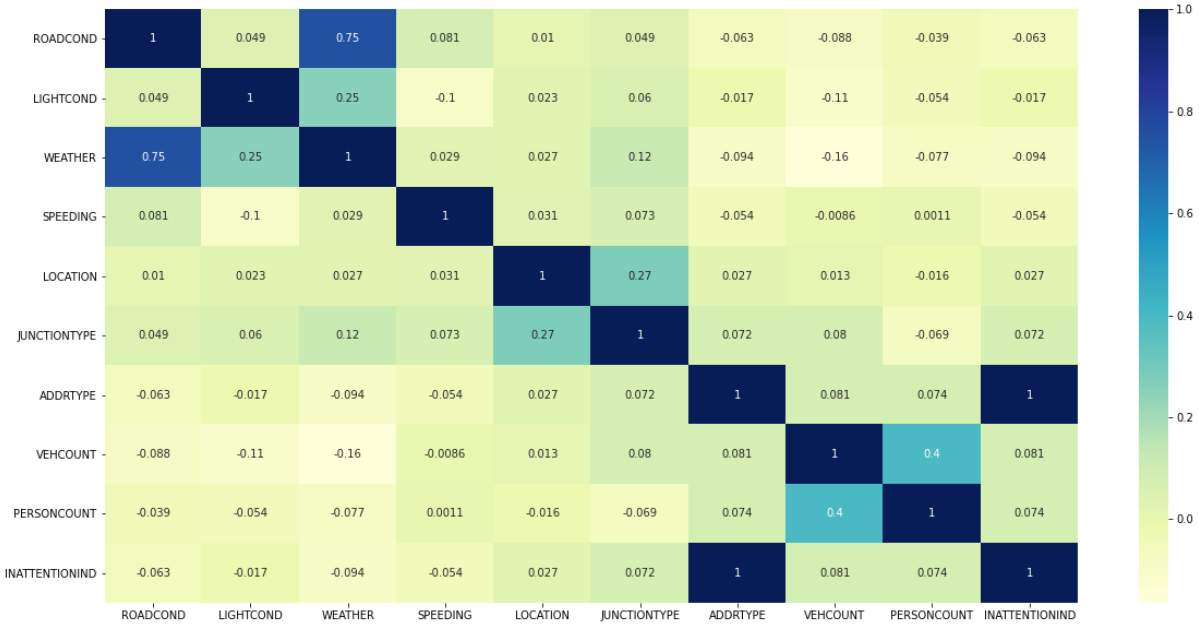
Final Preprocessed Data

	ROADCOND	LIGHTCOND	WEATHER	SPEEDING	LOCATION	JUNCTIONTYPE	ADDRTYPE	VEHCOUNT	PERSONCOUNT	INATTENTIONIND
86800	0	5	1	0	18973	1	0	2	2	0
181357	8	5	4	0	8413	1	0	2	4	0
33868	0	5	1	0	16517	4	0	2	2	0
92209	0	5	1	0	19617	4	0	2	2	0
48038	0	2	1	0	9472	1	0	3	5	0
...	...	...	...	...	...	...	...	...	...	...
9603	0	5	1	0	10145	4	0	5	7	0
73706	0	5	1	0	11700	2	0	1	3	0
138284	8	2	10	0	19760	1	0	2	2	0
107442	0	5	1	0	9249	3	0	2	3	0
73575	8	2	6	0	7890	1	0	1	2	0

116376 rows × 10 columns

The data is now cleaned and all the values are numerical and hence suitable to be fed as input to ML algorithms.

Checking for Data Correlation



There is a significant correlation between Weather and Road Condition features observable in the heatmap. Noticeable correlations between Vehicle count and Person count, Location and Junction type features.

Train-test split of data

Splitting the data into train and test data. 33% of the data is taken to be test data while 66% is used for training the algorithm.

```
Shape of X_train :  (77971, 10)
Shape of X_test  :   (38405, 10)
Shape of y_train :  (77971,)
Shape of y_test  :   (38405,)
```

This data can now be used to input to various ML algorithms and check for model performance based on various metrics. The model with best metrics would be chosen for the classification and prediction.

Modeling

- **K-Nearest Neighbors Classifier**

  Performing hyperparameter tuning to find the best parameters. Training of the model is performed using the best hyperparameters.

```
[ ]  classifier = GridSearchCV(KNeighborsClassifier(), hyperparameters)
     classifier.fit(X_train, y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=KNeighborsClassifier(algorithm='auto', leaf_size=30,
                                            metric='minkowski',
                                            metric_params=None, n_jobs=None,
                                            n_neighbors=5, p=2,
                                            weights='uniform'),
             iid='deprecated', n_jobs=None,
             param_grid={'leaf_size': [25, 30, 35],
                         'n_neighbors': [15, 20, 25, 30]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
[ ]  classifier.best_params_
```

```
{'leaf_size': 25, 'n_neighbors': 25}
```

```
[ ]  classifier = KNeighborsClassifier(leaf_size = 25, n_neighbors = 25)
```

```
[ ]  Knn_classifier = classifier.fit(X_train,y_train)
```

Trained model is saved for evaluation phase which is carried out in the later stage.

- **Decision Tree Classifier**

Similarly, performing hyperparameter tuning on Decision Tree Classifier to find the best parameters followed by training of the model using the best hyperparameters.

```
[ ]  hyperparameters = {"max_depth": [6,7,8,9,10],
                        "max_features": [5,6,7,8,9],
                     "criterion": ["gini", "entropy"]}
```

```
[ ]  classifier = GridSearchCV(DecisionTreeClassifier(), hyperparameters)
     classifier.fit(X_train, y_train)
```

```
GridSearchCV(cv=None, 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=None,
                                              splitter='best'),
             iid='deprecated', n_jobs=None,
             param_grid={'criterion': ['gini', 'entropy'],
                         'max_depth': [6, 7, 8, 9, 10],
                         'max_features': [5, 6, 7, 8, 9]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
[ ]  classifier.best_params_
```

```
{'criterion': 'entropy', 'max_depth': 8, 'max_features': 8}
```

```
[ ]  classfier = DecisionTreeClassifier(criterion = 'gini', max_depth =  8, max_features = 8)
```

```
[ ]  DTClassifier = classifier.fit(X_train,y_train)
```

Trained model is saved for evaluation phase which is carried out in the later stage.

- **Random Forest Classifier**

Lastly, performing hyperparameter tuning on Random Forest Classifier to find the best parameters followed by training of the model using the best hyperparameters.

```
[ ] hyperparameters = {'n_estimators': [200,300,400],
                       'max_depth': [3,4,5]}
```

```
[ ] classifier = GridSearchCV(RandomForestClassifier(max_features = 5), hyperparameters)
    classifier.fit(X_train, y_train)
```

```
GridSearchCV(cv=None, error_score=nan,
             estimator=RandomForestClassifier(bootstrap=True, ccp_alpha=0.0,
                                              class_weight=None,
                                              criterion='gini', max_depth=None,
                                              max_features=5,
                                              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=100, n_jobs=None,
                                              oob_score=False,
                                              random_state=None, verbose=0,
                                              warm_start=False),
             iid='deprecated', n_jobs=None,
             param_grid={'max_depth': [3, 4, 5],
                         'n_estimators': [200, 300, 400]},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

```
[ ] classifier.best_params_
```

```
{'max_depth': 5, 'n_estimators': 400}
```

```
[ ] classfier = RandomForestClassifier(max_depth = 5, max_features = 5, n_estimators = 400)
```

```
[ ] RFClassifier = classifier.fit(X_train, y_train)
```

Trained model is saved for evaluation phase which is carried out in the later stage.

It can be noticed that for all the three classifiers, GridSearchCV with default cross-validation = 5, is used for hyperparameter tuning.

As all the models are trained and stored separately, evaluation of their performances can be checked for and is documented in the next phase.

Evaluation

Performance of the models is evaluated using performance metrics such as precision, recall, f1-score, accuracy, and confusion matrix.

- **K-Nearest Neighbors Classifier**

```
y_pred = Knn_classifier.predict(X_test)
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           1       0.64      0.63      0.64     19202
           2       0.64      0.65      0.64     19203

    accuracy                           0.64     38405
   macro avg       0.64      0.64      0.64     38405
weighted avg       0.64      0.64      0.64     38405
```

```
print(f1_score(y_test,y_pred))
```

```
0.6365501999579036
```

```
print(confusion_matrix(y_test,y_pred))
```

```
[[12097  7105]
 [ 6709 12494]]
```

The performance of KNN classifier is best when the leaf size = 25 and the number of nearest neighbors = 25. The accuracy of classification is 64%.

- **Decision Tree Classifier**

```
y_pred = DTClassifier.predict(X_test)
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           1       0.69      0.60      0.64     19202
           2       0.65      0.72      0.68     19203

    accuracy                           0.66     38405
   macro avg       0.67      0.66      0.66     38405
weighted avg       0.67      0.66      0.66     38405
```

```
print(f1_score(y_test,y_pred))
```

```
0.6428789096046762
```

```
print(confusion_matrix(y_test,y_pred))
```

```
[[11603  7599]
 [ 5292 13911]]
```

The performance of Decision Tree classifier is best when the max. tree depth = 8, max. features = 8 and the criterion = ‘gini’. The accuracy of classification is 66%.

- **Random Forest Classifier**

```
y_pred = RFClassifier.predict(X_test)
```

```
print(classification_report(y_test,y_pred))
```

```
              precision    recall  f1-score   support

           1       0.69      0.58      0.63     19202
           2       0.64      0.74      0.69     19203

    accuracy                           0.66     38405
   macro avg       0.67      0.66      0.66     38405
weighted avg       0.67      0.66      0.66     38405
```

```
print(f1_score(y_test,y_pred))
```

```
0.6309763355533513
```

```
print(confusion_matrix(y_test,y_pred))
```

```
[[11132  8070]
 [ 4951 14252]]
```

The performance of Random Forest classifier is best when the number of estimators = 400 and max. depth = 5. The accuracy of classification is 66%.

Tabulated Result

```
Results:
+---------------------+----------+-----------+--------+---------+
|     Classifier      | Accuracy | Precision | Recall | F1Score |
+---------------------+----------+-----------+--------+---------+
| K-Nearest Neighbors |   0.64   |   0.64    |  0.63  |  0.64   |
|    Decision Tree    |   0.66   |   0.69    |  0.6   |  0.64   |
|    Random Forest    |   0.66   |   0.69    |  0.58  |  0.63   |
+---------------------+----------+-----------+--------+---------+
```

Tabulated Result

Conclusion

1. The Seattle Accident dataset has many missing values and handling of missing data is an important step before any classification algorithm is used upon such data.

2. Data cleaning constitutes the handling missing data as well as replacing 'NaN' values with appropriate substitution. The data is imbalanced in terms of the class labels and is balanced using down-sampling of the major class datapoints.

3. Not all features in the Seattle Accident data are important and most of the features are dropped as they are not relevant for the classification problem. For e.g., the Incident number column bears no relevance towards the Severity classification of the incident. Therefore, only a few features out of the 37 available features are being used.

4. The categorical and text features are label encoded, as the text and categorical data cannot be directly input to a classification algorithm.

5. Data normalization is performed. A classification algorithm is used as the problem at hand is a binary classification of accident severity (Class 1 or Class 2.)

6. From the final results obtained, it can be concluded that although there is some correlation between the features (such as weather, road condition, etc.,) and the severity of the accident but there is no concrete evidence of a pattern for classification, as signified by the model performance results.