Accident Severity Prediction using Seattle Accident Dataset

Applied Data Science Capstone Project
IBM - Coursera

Karthik J Ghorpade
September 2020

An accident won't arrive with a bell on its neck

- Finnish Proverb

Introduction

- Vehicular accidents, no doubt, cause damage to people involved in the collision as well as property.
- Severity of the accident mainly depends on the harm caused to humans rather than the property
- Studies on existing collision data may lead to important insights into underlying causes which lead to accidents
- This project is an attempt to predict severity of a possible futuristic accident based on existing collection of collision data

Business Problem | Application

- The study performed in this project is on the Seattle Collision data
- Authorities such as Seattle PD can benefit from such a study in narrowing down causes of accidents and take steps to reduce severity of collisions
- The people of Seattle can use such a study to take cautionary measures
- Cities with geographical and structural similarities could also benefit

Data Collection and Exploration

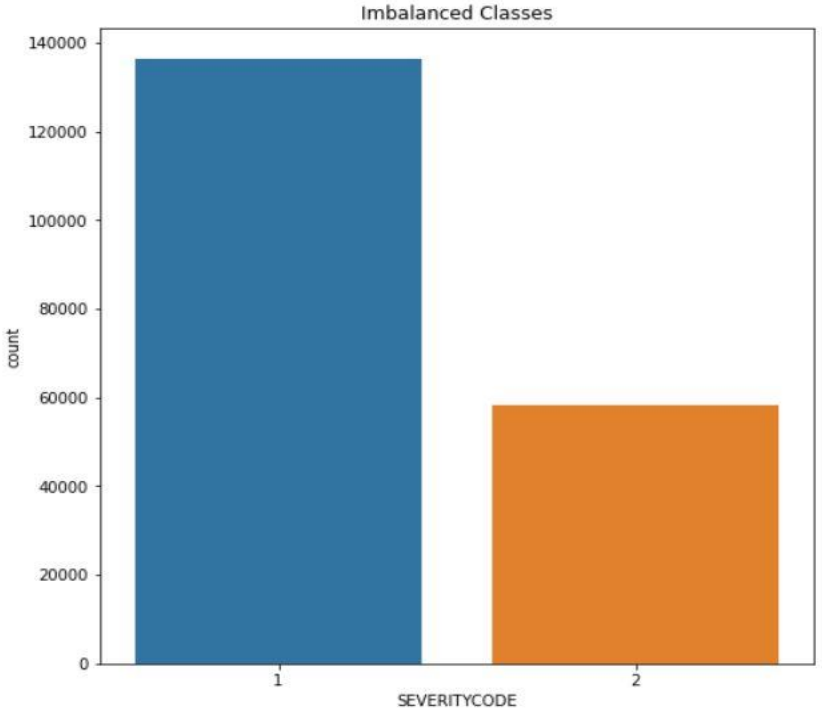
- Data made by Seattle GeoData – Collisions dataset
- 194763 instances of accidents – 37 attributes and class label attribute per instance
- Data obtained has many missing values – needs preprocessing
- The class label or the target class is used to classification:
  1. Class ‘1’ signifies low severity – property damage due to collision
  2. Class ‘2’ signifies high severity – harm to humans and possible property damage
- The classes are imbalanced – large number of Class ‘1’ data compared to Class ‘2’

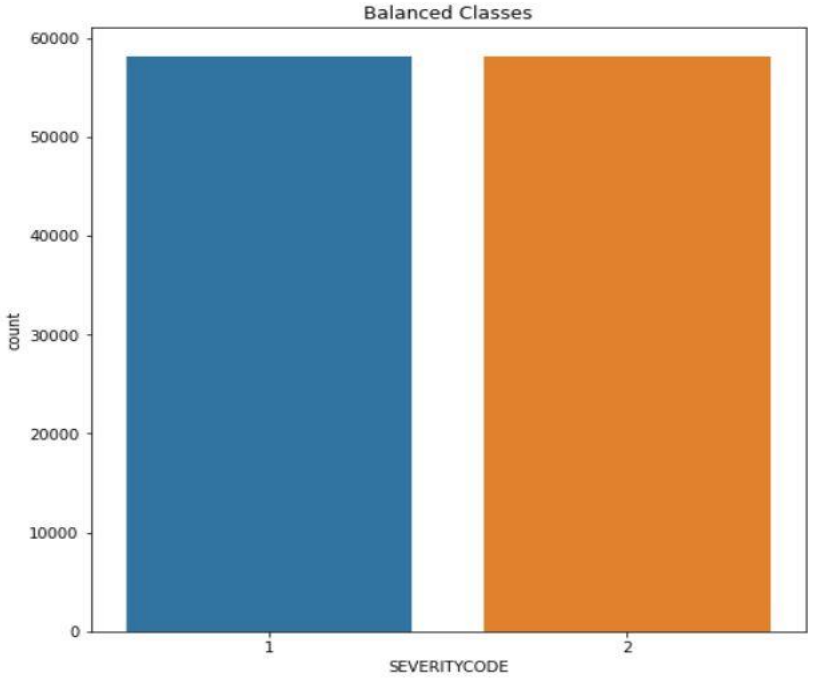
Data before cleaning

	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	N
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 ...	N
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 ...	N
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 ...	N
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 ...	N

Many NaN values and missing values.

Balancing Imbalanced Data





The classes are balanced by down-sampling the majority class i.e. Class ‘1’

Data Pre-processing

- Handling missing data
- Label encoding categorical data
- Column-wise data normalization
- Choosing the suitable features

Data after pre-processing

	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

Modelling

- Train-test splitting of data
- Choosing the appropriate algorithms
- Hyperparameter tuning
- Training with best parameters

K-Nearest Neighbours Classifier

```
[ ]  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)
```

Hyperparameter tuning and training with best parameters

Decision Tree Classifier

```
[ ]  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)
```

Hyperparameter tuning

```
[ ]  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)
```

Training with best parameters

Random Forest Classifier

```
[ ] 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)
```

Hyperparameter tuning

```
[ ] 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)
```

Training with best parameters

Evaluation – KNN 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]]
```

Performance Metrics

Evaluation – Dec. 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]]
```

Performance Metrics

Evaluation – RF 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]]
```

Performance Metrics

Consolidated Result | Conclusion

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

1. The accuracy of the models suggest that there is in fact 65% predictability of severity of accident based on collision and other attributes used in the model. This is very significant as these attributes could help in prediction and hence reduction of severity of possible accidents
2. 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.
3. Missing data handling, data preprocessing and feature selection are important steps involved in this project.
4. Advanced feature engineering techniques could provide concrete patterns for severity prediction