CA4010 Data Warehousing and Data Mining

Predicting if a car accident is fatal

Student Name	Email	Student No.
Emily McGivern	emily.mcgivern3@mail.dcu.ie	16305506
Camilla Boyle	camilla.boyle36@mail.dcu.ie	16732949

Plagiarism Declaration

I/We declare that this material, which I/We now submit for assessment, is entirely my/our own work and has not been taken from the work of others, save and to the extent that such work has been cited and acknowledged within the text of my/our work. I/We understand that plagiarism, collusion, and copying are grave and serious offences in the university and accept the penalties that would be imposed should I engage in plagiarism, collusion or copying. I/We have read and understood the Assignment Regulations. I/We have identified and included the source of all facts, ideas, opinions, and viewpoints of others in the assignment references. Direct quotations from books, journal articles, internet sources, module text, or any other source whatsoever are acknowledged and the source cited are identified in the assignment references. This assignment, or any part of it, has not been previously submitted by me/us or any other person for assessment on this or any other course of study.

Name(s): Camilla Boyle, Emily McGivern

Date: 18/12/19

1. Introduction

1.1. Idea and Dataset Description

Our aim was to predict if a car accident would be fatal. We chose a dataset with a large number of attributes and rows which we hoped would help us in making our analysis and prediction. The dataset we used came from kaggle. It was downloaded in .CSV format. It contained four different files: Accident file, Vehicle File, Casualty file and Lookup file. The dataset contained a large number of rows and attributes that we could use to make our prediction. We used attributes from 3 of the 4 files. The Lookup file served to explain what some of the values in the .CSV file represented.

2. Dataset Preparation

2.1. Pre-Processing

Once we downloaded our .CSV files, we needed to combine them into one single .CSV file. We did this using a Python script. Next, we needed to remove attributes that would not help us make our prediction. Upon inspection, we realised that some factors did not affect the likelihood of a fatality. For example, we removed attributes for longitude and latitude due to the fact that we had a district attribute which indicates location along with an attribute to distinguish between urban or rural area.

We had a number of Null values in our dataset which also needed to be removed so as to not skew our results. If we had left them in, our prediction may have been inaccurate or incorrect. After our workshop, we realised that we still had too many attributes to make an accurate prediction. There were some attributes which would not offer us useful information for making our prediction. Furthermore, we knew that we would have to change the format of some of our attributes as there were too many values, such as date and time.

2.2. Attribute Selection

Our dataset came with a very large amount of attributes. However, for our prediction, not all would be useful. Despite having cut out some attributes for our workshop, we realised that we still needed to choose more useful attributes which would aid us in our prediction. After further investigation, we chose to use the following attributes:

Accident_Severity	Number_of_Vehicles	Road_Type	Speed_limit
Light_Conditions	Weather_Conditions	Day_of_Week	1st_Point_of_Impact
Sex_of_Driver	Age_Band_of_Driver	time_of_day	Road_Surface_Conditions
Season	Urban_or_Rural_Area		

2.3. Classification

The next stage in the process was to decide between classification and clustering for making our prediction. We felt that classification was more suitable than clustering as we had a known number of labelled classes. However, we needed to classify some of our columns such as date, time and speed limit which were not currently in classes. For speed limit, the values of the attribute ranged from about 12 km/h to about 120km/h. We broke it down into 4 different classes, 0 - 30, 31 - 60, 61 - 90 and 91 - 120. We felt that these were the best classes for speed as they represent very slow, moderate, fast and very fast.

Adding classes to Speed_limit attribute

```
d.loc[d['Speed_limit'] <= 30, 'Speed_limit'] = 1
```

```
d.loc[(d['Speed_limit'] > 30) & (d['Speed_limit'] <= 60), 'Speed_limit'] = 2
```

```
d.loc[(d['Speed_limit'] > 60) & (d['Speed_limit'] <= 90), 'Speed_limit'] = 3
```

```
d.loc[(d['Speed_limit'] > 90) & (d['Speed_limit'] <= 120), 'Speed_limit'] = 4
```

We chose to classify time in a similar way. We broke down the time attribute by whether the time fell in the morning, afternoon, evening or night. This gave us 4 labelled classes which could be used to make our prediction. With our date attribute, we chose to classify the date by season. We did this because both weather and light conditions during months in the same season are very similar. We removed the date attribute and replaced it with Season, which has values from 1 - 4 representing Spring, Summer, Autumn and Winter.

Creating labelled classes for Season based on date

```
In [40]: d.loc[(d['Season'] >= 3) & (d['Season'] <= 5), 'Season'] = 1
```

```
In [41]: d.loc[(d['Season'] >= 6) & (d['Season'] <= 8), 'Season'] = 2
```

```
In [42]: d.loc[(d['Season'] >= 9) & (d['Season'] <= 11), 'Season'] = 3
```

```
In [43]: d.loc[(d['Season'] >= 12) & (d['Season'] <= 2), 'Season'] = 4
```

3. Algorithms

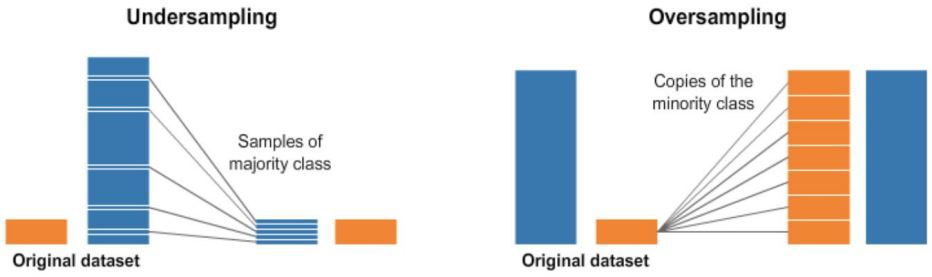
After deciding to do classification, we then had to choose some algorithms to help us in making our prediction. We looked into a number of algorithms which could be used on our dataset. First, we consulted the module notes and considered the algorithms there.

The following section will discuss each algorithm and the process we used to implement them on our dataset. To carry out these algorithms, we used Python and a number of different libraries and tools such as Pandas, SkiLearn and Jupyter Notebook.

3.1. Imbalanced Classification Problem

On our first round of implementing our algorithms on our training data, we noticed that our accuracy scores were coming out at between 97% and 99%. These seemed extremely high. After doing some research, we realised that the values are so high because we have an imbalanced classification problem occurring. Of the 817,186 rows we have, only 8098 of them represent fatal accidents. This is because non-fatal accidents are far more common than fatal accidents. One of the categories represents the overwhelming majority of the data, as just over 99% are non-fatal. We had to assess how to solve this issue. We came across a number of solutions to our problem.

1. Over-Sampling - Add copies of instances from the minority class
2. Under-Sampling - Delete instances from the majority class



Under- and Over-Sampling

Upon consideration, we chose to remove instances from our majority class. This meant that we ran the risk of discarding some useful information however, we hoped that due to the size of our dataset we would still retain enough information to enable us to make our prediction. We had to look into Under-Sampling techniques to ensure that we carried it out correctly. To carry it out, we used SkiKit Learns resample tool. By using this, we ended up with an equal amount of both classes. Immediately we saw an improvement in results and could proceed with working on our prediction. After we had carried this out, we could proceed with running various algorithms against our dataset.

```
df_majority_downsampled = resample(df_majority,
                                   replace=False,
                                   n_samples=8098,
                                   random_state=123)
```

3.2. Naive Bayes

Naive Bayes are a family of classifiers which determine the probability of an outcome given a set of conditions. In our case, the outcome is that an accident is fatal or that it is not fatal. We have a number of conditions in the form of attributes in our dataset. We did further research on the topic of Naive Bayes and found references to 3 different Naive Bayes classifiers which we could implement: Bernoulli, Multinomial and Gaussian.

```
bnb = BernoulliNB()
bnb.fit(X_train, y_train)
prediction = bnb.predict(X_test)
print (accuracy_score(y_test, prediction))
```

```
0.4961926322288537
```

```
gnb = GaussianNB()
gnb.fit(X_train, y_train)
prediction = gnb.predict(X_test)
print (accuracy_score(y_test, prediction))
```

```
0.692529327022021
```

```
mnb = MultinomialNB()
mnb.fit(X_train, y_train)
prediction = mnb.predict(X_test)
print (accuracy_score(y_test, prediction))
```

```
0.6369623379296151
```

The results of each algorithm are shown above. The accuracy score of the Bernoulli naive Bayes is almost 50%. This means it is as likely to predict correctly as it is to predict incorrectly. As Bernoulli Naive Bayes predicts best with boolean features and we did not have exclusively boolean features this result is not surprising. With the Gaussian naive Bayes, we saw a large increase in accuracy, reaching nearly 70%. This is a good accuracy score but we were still interested in testing other algorithms to see if there were better performances. There was a slight drop in accuracy with the Multinomial naive Bayes score, as it came to 64%. While this is not low, it is not quite accurate enough to be acceptable.

3.3. k-Nearest Neighbour

The k-Nearest Neighbour algorithm is a machine learning algorithm that can be used to solve classification and regression problems. In classification, the output is a class membership. With this algorithm, a class is determined based on the class of surrounding neighbours. With the k-Nearest Neighbour algorithm, the accuracy was 76%. It was an interesting algorithm to use as it is simple but effective in performing predictions. We were happy with this result as we had expected this algorithm to perform well. The confusion matrix also gives us a useful breakdown of the results of the algorithm. It is interesting to be able to compare the number of true positives versus false positives and the same for negative results. It means that the algorithm is very transparent in its results and is easier to understand how and why it performed as well as it did.

Classification Report

	precision	recall	f1-score	support
1	0.72	0.83	0.77	2411
2	0.80	0.69	0.74	2448
accuracy			0.76	4859
macro avg	0.76	0.76	0.76	4859
weighted avg	0.76	0.76	0.76	4859

Confusion Matrix

```
TP - True Negative 1995
FP - False Positive 416
FN - False Negative 768
TP - True Positive 1680
Accuracy Rate: 0.7563284626466351
Misclassification Rate: 0.2436715373533649
```

3.4. Decision Tree

As we had proceeded with classification, we decided to use a decision tree to construct a model for our dataset. A decision tree tries to make a prediction by taking into account every scenario for an attribute. We used SkiKit Learn's Decision Tree Classifier class with Jupyter notebook to run our Decision Tree Classifier. The result of our algorithm came out at 78%, which is our highest accuracy rating. This is a very acceptable level of accuracy. We had originally expected the decision tree to have the highest level of accuracy for our dataset due to the nature of our dataset and how the algorithm works. As a decision tree continuously splits according to certain parameters, this method seemed to line up with our dataset and how it was classified. Before even running the algorithm, we could assume that it would work well.

```
prediction = dtc.predict(X_test)
print (accuracy_score(y_test, prediction))
```

```
0.7773204363037662
```

3.5. Random Forest Classifier

We also decided to run a random forest classifier algorithm against our data. A random forest classifier creates a set of decision trees from a randomly selected subset of training data. It then aggregates the results from different decision trees to decide the final class of an object. We chose to implement this algorithm as we thought that if we had a combination of decision trees being examined, it would give us an interesting result. We also wanted to examine the feature importance indication. The accuracy of this algorithm was 67%. This is not as successful a score as our previous Decision Tree.

```
from sklearn.ensemble import RandomForestClassifier
```

```
clf = RandomForestClassifier(n_estimators=100, max_depth=2, random_state=0)
```

```
clf.fit(X, y)
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
            max_depth=2, max_features='auto', 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, n_estimators=100, n_jobs=None,
            oob_score=False, random_state=0, verbose=0, warm_start=False)
```

```
RandomForestClassifier(bootstrap=True, class_weight=None, criterion='gini',
                       max_depth=2, max_features='auto', 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, n_estimators=100,
                       n_jobs=None, oob_score=False, random_state=0, verbose=0,
                       warm_start=False)
```

```
prediction = clf.predict(X_test)
```

```
print (accuracy_score(y_test, prediction))
```

```
0.6799753035604034
```

Feature Ranking

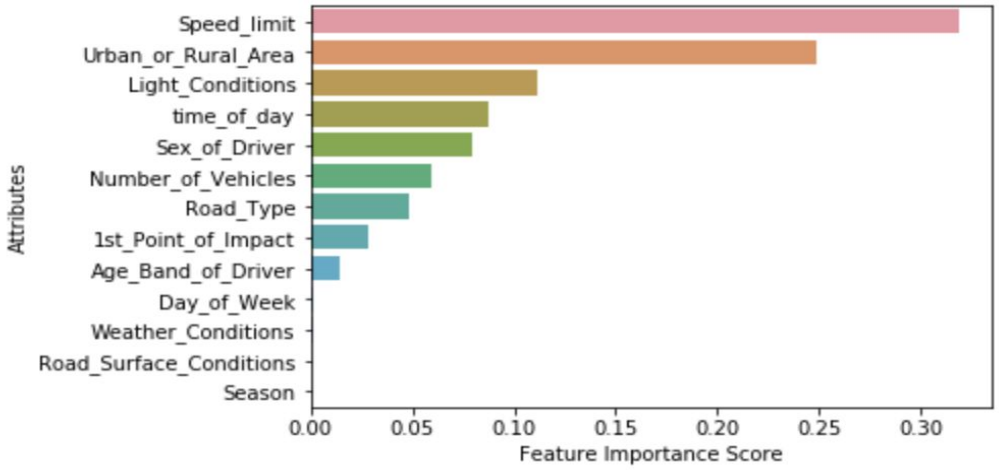
The last algorithm we decided to run against our dataset was a feature ranking algorithm. We did this in conjunction with our random forest classifier to give us an idea of where the result came from and why. The following table shows the ranking of features importance, from 1 to 13. This shows the importance of features on a classification task. This was extremely interesting, as it gave us an indication of which features were contributing the most into making our prediction. It also allowed us to examine our own assumptions about our dataset to see if they were correct. The results were definitely different to what we originally expected in terms of what features were important and what were not. As these results are just in relation to the random forest classifier, we cannot say they are definitely correct but they are very useful in helping us understand our dataset further.

```
feature_imp = pd.Series(clf.feature_importances_,index=dff.columns).sort_values(ascending=False)
feature_imp
```

```
Speed_limit                 0.319072
Urban_or_Rural_Area         0.248499
Light_Conditions            0.111843
time_of_day                 0.087637
Sex_of_Driver               0.079570
Number_of_Vehicles          0.059758
Road_Type                   0.048143
1st_Point_of_Impact         0.028770
Age_Band_of_Driver          0.014269
Day_of_Week                 0.001122
Weather_Conditions          0.001056
Road_Surface_Conditions     0.000154
Season                      0.000107
dtype: float64
```

```
import matplotlib.pyplot as plt
import seaborn as sns
%matplotlib inline

sns.barplot(x=feature_imp, y=feature_imp.index)
plt.xlabel('Feature Importance Score')
plt.ylabel('Attributes')
plt.show()
```



4. Results

The following table shows the accuracy scores calculated for each algorithm that we implemented. In each case, the split was 70:30 training data to test data. We had originally planned to implement a Decision Tree classification algorithm, however, we chose to experiment with other algorithms to see if we could make any other interesting discoveries. The decision tree classifier ended up returning the highest accuracy for predicting if a car accident was fatal. We had assumed that this would be the ideal algorithm for our dataset due to the nature of the data.

Algorithm	Bernoulli	Gaussian	Multinomial	k-Nearest Neighbour	Decision Tree	Random Forest Classifier
Accuracy	0.50	0.69	0.64	0.76	0.77	0.68

Alongside this, the results of the Random Forest Classifier also ended up being extremely useful in validating our assumptions about our dataset. The Feature Importance ranking that was produced from the Random Forest Classifier gave us a list of the dataset attributes ranked in order of importance for making a prediction. When we first chose this dataset, we did some basic investigation on our dataset to determine what we thought would be the main attributes to cause a fatal accident. By simply looking at the attributes we assumed that speed limit, the number of vehicles involved and both light and weather conditions would definitely be obvious attributes which would help in making a prediction. From these results, our assumptions about speed limit and light conditions were correct. However, number of vehicles was lower down the list than we expected, and we were also surprised about far down the list weather conditions turned out to be in terms of importance.

On reflection, there were many things we would do differently if we were to do this again. We realised later that there were other workshops which might have been more useful to us as even after doing our own workshop, we still had more pre-processing tasks to do that we had not anticipated. We also feel that our prediction may have been a bit too broad for the size of our dataset. This is why we ran into issues with imbalanced classification. As we had so many features in our dataset, we had a large range of choice of predictions we could make. We would possibly narrow down our prediction for the next time. We also did not realise the importance of some features. Next time, we would examine each feature to see their general effect as opposed to just disregarding them if we feel it is not relevant. It did not have a major effect on our prediction results this time, but it would be interesting to see the results if we had left some features in.

5. Conclusion

When we first started this project, we did not realise just how important data cleaning and sorting would be. We progressed from the removal of null attributes and those that were not relevant to our prediction, to adding classes for attributes such as season and time of day. Once this was done, alongside the resampling, we could progress with finding the most suitable algorithm and model to use. From here, we were able to investigate which classifications were most accurate and using feature importance we were able to establish which attributes have the greatest effect on the ability to predict a fatality.

What we found most interesting was seeing the varying results of different algorithms on our dataset. We found that the decision tree classifier algorithm is most accurate for our

dataset, but we learned a great deal from researching each one and discovering where each may be useful in the future. We also learned that our assumptions about our dataset were not always accurate. For example, we assumed that bad accidents occur in poor weather conditions, however, this was not the case. While small accidents occur more frequently in poor weather conditions, fatal accidents do not.

Overall, we feel as though we have learned a lot from this project. With the nature of it, you learn from your mistakes a huge amount. It gave us the opportunity to understand why some algorithms worked better than others and how important some features can be. We also learned about how the size of a dataset can greatly affect your prediction and this must be taken into account.

6. References

6.1. https://scikit-learn.org/stable/modules/generated/sklearn.utils.resample.html

6.2. https://elitedatascience.com/imbalanced-classes

6.3. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html

6.4. https://scikit-learn.org/stable/auto_examples/ensemble/plot_forest_importances.html

6.5. https://medium.com/machine-learning-101/chapter-5-random-forest-classifier-56dc7425c3e1

6.6. https://towardsdatascience.com/what-to-do-when-your-classification-dataset-is-imbalanced-6af031b12a36