Predicting Crash Severity and Fatalities for NZ Road Accidents Final Report

I. Definition

Project Overview

Car crashes and road accidents could be considered an old topic. Yet, with the progress of the technology included in in cars and their new capabilities, it is ever more important to have tools and means available to mitigate their occurrences, as well as their implications and consequences for the people involved. Thanks to the advancement of technological and analytical tools in the last two decades we are now able to better understand how crashes happen. This enables the transport, security and emergency agencies all around the world to have different (predictive) models for quickly analyzing crashes when they happen and dispatch an appropriate response swiftly.

On a personal note, I'm looking to move to New Zealand. I was drawn to this problem while looking for an interesting dataset and problem that I could use to build a portfolio; which I then intend to use for networking purposes with professionals from NZ in my search for job opportunities.

The dataset that we will use comes directly from the *New Zealand Transport Agency* and is made available through the website *data.govt.nz;* a repository of open datasets related to all kind of activities throughout New Zealand and published by the central government. The dataset is available in many formats, we will use the CSV version. It contains data from car crashed since January 1st of the year 2000 until the present day and it's automatically updated on a quarterly basis.

I have included the version of the datasets with the data corresponding to 2018Q2 to the github repository for this project. The latest version can be found in its corresponding landing page here, or can be directly download here. In addition, the NZ Transport Agency has made available a definition for each feature included in the data here.

It's important to note, as the description states, that: "not all crashes are reported to the NZ Police. The level of reporting increases with the severity of the crash. Due to the nature of non-fatal crashes, it is believed that these are under-reported".

Problem Statement

Predicting the severity of a car crash is no easy task. And even when possible, precision levels will vary significantly depending on the data available and how well the system or model has been defined. However, if the dataset's features are clearly defined and if there's a thorough description of how this data is collected –as is our case– we have much better chances of arriving at a usable model.

In this particular dataset, the severity for a crash is classified under four different categories: non-injury, minor, serious and fatal.

Our original goal was twofold. First, we wanted to predict the severity of a given crash sample. And second if classified as *fatal*, we wanted to estimate the number of fatalities.

However, due to time constraints and the data at hand we were only able to predict crash severities just a little bit below par from our benchmark values.

We explain how we got to this result and also set out a path to solve different problems we've found while working on the data and how to improve the results.

To achieve this, we will first do an exhaustive exploration of the data. This will provide us with valuable insights about the features we have available, what values they take and if/how they correlate with one another. This will serve also as a first round of feature selections towards the final model.

Then we will implement PCA to see if there's some latent feature we could leverage in a more direct way for our purposes.

This will lead us to the actual classifier(s) where we will use the original data and the newly found insights to train a model that we hope will be able to predict the severity.

Finally, we will explain and give details on what are the possible next steps to improve the overall performance and interpretability of the final model.

All files for the project are available in my personal Github profile here. And there's also a Tableau workbook that was used for doing the initial EDA together with an *EDA* notebook available in the repo.

The workbook is available for download on my Tableau Public profile here. It is a packaged workbook with different data sources, worksheets and dashboard.

The repository has a *README*; *requirements.txt*; two jupyter notebooks, one for named *EDA* and another one named *Classifiers; a data* folder with all the necessary input data and partial results; a *saved_models* folder with different models that were trained and saved to disk to enable working on the during different sessions; and an *images* folder with image filed exported from Tableau's workbook used to visualize different aspects of the dataset and the models that were trained.

The *EDA* notebook contains all the work that was done to understand the dataset at the sample and feature level. It also has sections dedicated to data sanitization and a section for dimensionality reduction.

The *Classifiers* notebook contains all the code and work that corresponds to training of the different models; the preprocessing that was required; and the optimization that was done.

The two notebooks have a narrative of themselves, accompanied with explanatory text and imaged so that the anyone can read easily through them.

Metrics

Considering our problem is a multiclass classification one, we will use an F1 score to evaluate the performance of the different models that we will train.

The value for the score will in turn be the average of the F1 scores for each class. So, we will be using a *one vs all* approach.

If we have classes *N, M, S* and *F*; for each model that we train we will have four corresponding F1 scores. The average of these four scores will make up the final F1 score for the model.

At the same time, the F1 score for each class will be the harmonic mean between *Precision* and *Recall.* This is because the F1 score is actually an F beta score with the parameter beta set to a value of *1*. Where *Precision* is a measure of the model's ability to correctly predict a given sample; and *Recall* is a measure of the model's ability to not miss a relevant case –for the class in question that is.

However, we won't create any specific code for this as *sklear* has it already implemented as the default behavior for the algorithms we've chose and a multiclass classification setting.

Using an F score with beta equal to 1provides a good balance between *Precision* –by considering the false positives– and *Recall* –by considering the false negatives. This is important because we want our model to be able to pick as much cases as possible (for each class); and to make as few mistakes as possible.

II. Analysis

Data Exploration

The dataset we will be using is an official one published by the NZ Transport Agency. It contains all crashes that have been reported since January 1st, 2000 and it is updated quarterly. The version we will be using includes data up to 2018Q2 included.

The dataset has an initial total of 655,697 samples. Each with 89 different features. The feature we want to predict is *crashSeverity* and it is a categorical feature that takes the values *N, M, S* and *F* to indicate that the crash was of class *Non-Injury*, *Minor*, *Serious* or *Fatal*, respectively.

The remaining features are both numerical and categorical. Additionally, there's also two geo spatial features (*X* and *Y*) used to indicate longitude and latitude of the crash site. Plus, an *OBJECTID*.

The github repository incudes two files called "*Crash_Analysis_System_CAS_data.pdf"* and "*data/features_description.tsv"*. The first one is a printed PDF from the NZTA website. It has a complete list of all features in the dataset and their meaning. The second file is an expanded catalog created for this project specifically. For each feature, it summarizes its meaning and adds the type of feature it is, and the data type used to parse its values in the notebooks.

We suggest consulting first the second file and resorting to the PDF when necessary.

Next, we provide a brief description of both categorical and numerical features. Followed by the insights and results found when analyzing the distribution of values for each feature and the correlation between pair of features.

Categorical Features

The dataset has many categorical features. All these features take a small number of values. Also, they are all nominal features. This means that there's no natural order to their values. Which makes it difficult to transform them in order to be used as input for *Sci-kit Learn* since it can only handle numerical values. So, as we explain in further detail later on, we will resort to a *One Hot Encoding* of the dataset –after cleaning it, of course.

These categorical features are used to indicate different attributes and condition about the crash. For example, the feature *regionDesc* correspond to the country's region where the crash happened. The feature *roadCurvature* is used to indicate the level of curvature of the road at the site of the crash. The feature *weatherA* and *weatherB* are used to indicate two different types of weather condition at the moment of the crash.

There are also derived features whose values are derived from one or more categorical features. For example, the feature *darkLight* is used to group the different values of another feature –*light*– into the labels *light* or *dark.* Another example is the feature *intersectionMidBlock* that maps the joint values of features *intersection* and *junction_type* into the values i*ntersection* and *Midblock*.

All these derived features will be removed from the dataset as they are redundant and create unnecessary correlation between features which is harmful to our goals. Also, we will remove some other features that create data leakage. We explain this in the following sections.

Numerical Features

Most of the numerical features in the dataset are counters that indicate the number of objects involved in the crash. For example, the feature *bridge* indicates how many times a bridge, tunnel, the abutments or handrails were struck in the crash. Similarly, the features *fatalCount*, *seriousInjuryCount* and *minorInjuryCount* indicate how many people got injured in the crash, regardless of the severity. So, for example, a crash of class *F* will have a value for *fatalCount* greater than zero and could also have values for *seriousInjuryCount* and *minorInjuryCount* also greater than zero. On the other hand, a crash of class *M* can only have *minorInjuryCount* greater than zero. If it had *seriousInjuryCount* or *fatalCount* greater than zero, then it wouldn't be of class *M*.

Besides from these counters (there are many more) we find three features related to speed limit. These features are: *speedLimit*, *temporarySpeedLimit* and *advisorySpeed*. Their meaning is evident. However, as we will show next, only the feature *speedLimit* will end up being usable.

Finally, there is also a numerical *crashYear* and *crashFinancialYear* used to indicate both the year of the crash (e.g. 2017) and the financial year of the crash (e.g. 2017/2018).

Data cleaning

As a first step in the analysis of the data, we have gone through a thorough exploration of each categorical feature and the numerical features that are not counters. The full details of this analysis can be found on sections 2, 3 and 4 of the first notebook. Here, we present the insights and results produced by this analysis.

Removal of non-Relevant Features

The first things we've done is remove some features that –after considering carefully their meaning as explained by the PDF– we understand are not relevant to our problem. Following is the list of features and the reason why we've decided to remove each one from the dataset.

- *X, Y, OBJECTID*: although these features are useful to uniquely identify a crash and we have use them in the Tableau workbook, they are not of much use when it comes to the final model and making predictions.

- *crashYear* and *crashFinancialYear*: these features could be of value for a historical and seasonal analysis through time series techniques, for example. But again, They're not relevant to the final model. We want the model to be able to analyze and predict the severity of a crash regardless of when it happened. We should also bear in mind the use case for the model. If its purpose is to be a tool for emergency agencies towards a quicker and better response; the feature vector that the model receives would correspond to crashes that have just happened. Following this logic, the year of the crash becomes irrelevant.

- *tlaID*: this is the ID of the *Territorial Local Authority*. Since we will use the feature *tlaName* for better understanding the data, we will disregard this one as it becomes redundant.

- *areaUnitID*: by reading the definition of this feature, it's not clear enough what it represents or how it might be relevant towards our goal. Therefore, we will also disregard it in the name of data sanity.

- meshblockID, easting, northing, crashRSRP, crashDirectionDescription, crashDistance, crashRPDescriptio n, crashRPNewsDescriptioncrashRPDirectionDescription outdatedLocationDescription: although each of these features have value for different purposes, they correspond to reference data used by the NZ Transport Agency for reporting and other administrative purposes. But they do not correspond to data from the crash itself.

- *fatalCount*, *seriousInjuryCount*, *minorInjuryCount*: these features are provided by the NZTA after the crash has been processed. Their values are not available at the time of the crash. At the same time, their values are completely correlated to our target variable, naturally. We remove them to avoid data leakage.

- *darkLight*, *intersectionMidBlock*: these features are derived from others. Therefore, they are highly correlated, and we shouldn't use them.

Feature Distribution

After removing the features mentioned above we have filled in all missing value with placeholder *###*. Then, we explored the values for each of the remaining categorical and numerical features (excluding counters) and treated this placeholder value when necessary. Following are different insights we've uncovered (again, the full analysis is available on the notebook).

crashSeverity

We found the following distribution of values:

N: 71.24%; *M*: 22.30%; *S*: 5.54%; *F*: 0.92%

it is important to note here –as the NZ Transport Agency warns– that the reporting of crashes increases with their severity. Meaning that all *Fatal* crashes have been reported most likely, but at the same time there will be *Non-Injury* and *Minor* crashes that weren't reported. This decision falls somewhat to the person in charge of responding to the crash and reporting it.

However, the most important thing to notice is how these classes are distributed and the imbalance between them. Since these are the classes we want to predict, and because the dispatched response would be very different depending on our prediction, we definitely need to be very careful about how we measure the performance of our model.

To this end, we will rely on Precision, Recall and F1 score as performance metrics and we will analyze their values for each class.

cornerRoadSideRoad

We need to clarify here, that the name of this feature in the dataset corresponds to the feature referred to as Crash *Road Side Road (CR_RD_SIDE_RD)* from the PDF. In the PDF, it says that the possible values are either *1*or *2*.

The dataset however, has 2 samples with the value *0* –which comes directly from the dataset– and 3 sample with the value *###* –which was our way to denote missing values. In any case, these are samples with values that are not included in the feature's definition.

Looking at the crash severity for these 5 samples we found that they are all of class *N*; which is the majority class, and the one responsible for most of the imbalance between classes. So, we simply have dropped these sample from the dataset.

This is the approach we have taken for all samples that present an inconsistent value for some feature with regard to the feature's definition.

directionRoleDescription, roadLane, roadMarkings, roadSurface, numberOfLanes, trafficContro

For all these features we have repeated the approach mentioned above.

crashRPSH

This feature is intended to indicate the *State Highway* on which the crash happened. It is used in tandem with *crashSHDescription* that indicates whether the crash happened on an SH or not (possible values are *Yes* and *No*).

The first thing we noticed when exploring the feature's values is that there were a lot of samples with missing or inconsistent values: 71.5% of samples had a value of *0* and another 12% were missing values (which are shown in the notebook with our placeholder *###I).* Together, these samples make up 83% of the whole dataset and we needed to treat them somehow.

In doing so, we looked separately at the samples that had *crashSHDescription=No* the samples that had *crashSHDescription=Yes*. This is because for cases for the first group, it wouldn't make sense to any a valid value and we could simply fill it with our placeholder.

In fact, we found that for 99.88% of samples in this first group, the value for *crashRPSH* was *0* or *###*. As a result, we set a value of *N/A* for all samples in this group.

On the other hand, when we looked at the second group, we found that 41% of samples had a value of *###*. So, we explore the values for the feature *crashLocation1* for these specific samples and found that they had a value of either "*SH 1N"* or "*SH 1S"*. As a result, we assigned a value of *1* to this subset of samples.

Finally, because of the size of remaining samples with values *###* and *0*, and their distribution among *crashSeverity* classes we chose to remove them from the dataset.

advisorySpeedLimit and temporarySpeedLimit

When we explored these features, we found that more than 98% of samples had a value *0*. Which doesn't make any sense. So, we decided to drop the features altogether.

Feature Correlation

Our next step was to explore the correlation between feature. To do this, we computed a Chi2 test of independence for each pair of categorical features.

Unfortunately, we weren't ab le to find a pair of features that were independent. In fact, all feature turned out to be dependent on every other feature. This can be seen both in the notebook and the Tableau workbook. A visualization of the dependency of every pair of features is shown in the next section.

Dimensionality Reduction

Finally, we tried to uncover some latent features by applying PCA with 10 components to the cleaned dataset – after transforming it through *One Hot Encoding*. We plotted all sampled using the first two PCs which accounted for more than 97% of total variability and saw that the crashes aligned on a few columns (as shown in the viz on the next section). However, these columns concentrate crashes of all classes. Meaning that the first two PCs, which account for more than 97% of total variability, are not able to create a separation between the classes we wanted to predict.

Since there were some features with numerical values (the counter) we applied a *MinMax* scaler to the dataset and repeated the PCA. But results were even worse than before; as the viz in the next section shows.

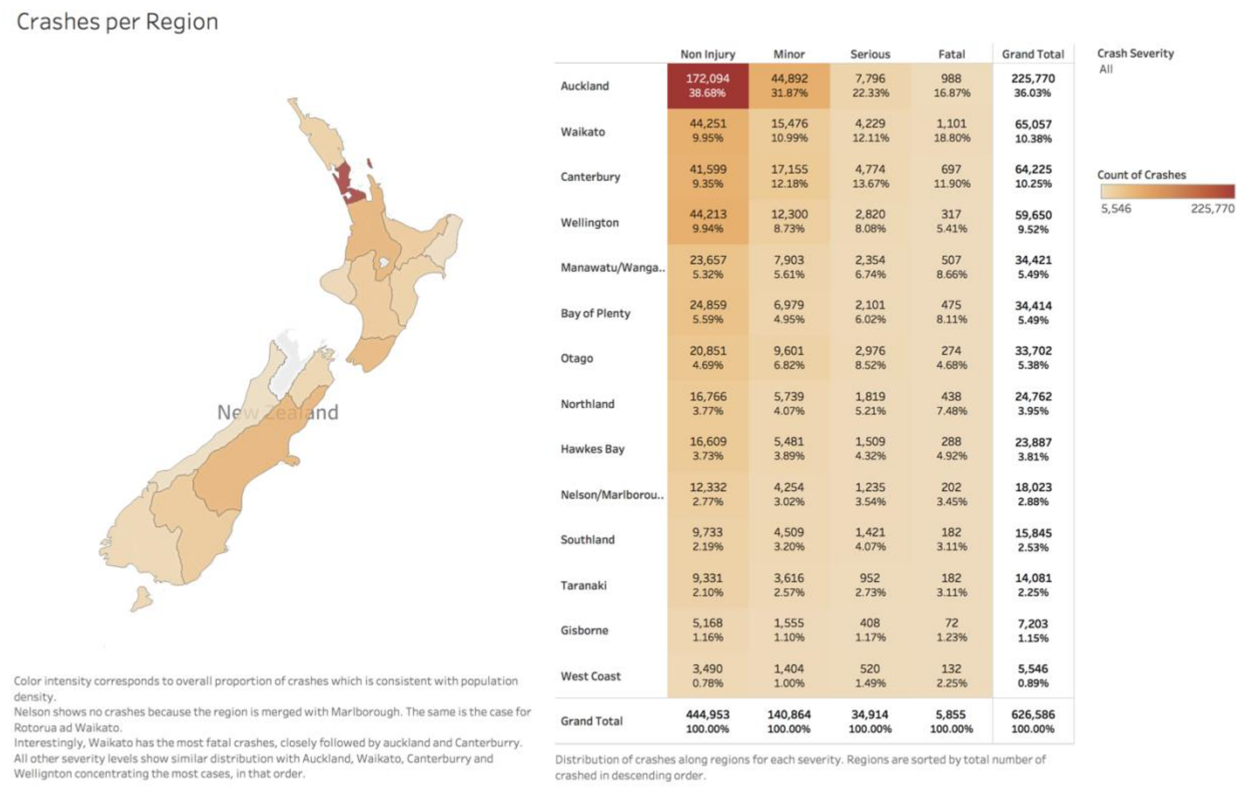
Exploratory Visualization

We defer the visual exploration of the dataset to a Tableau workbook published on Tableau Public and available for download here.

The workbook has some navigation controls that will allow for a deeper exploration of the data and it includes three dashboards.

The first one called *Crashes per Region* presents the total number of crashes per region.

Interestingly, Waikato has the most fatal crashes, closely followed by Auckland and Canterbury. All other severity levels show similar distribution with Auckland, Waikato, Canterbury and Wellington showing the highest concentrations, in that order.



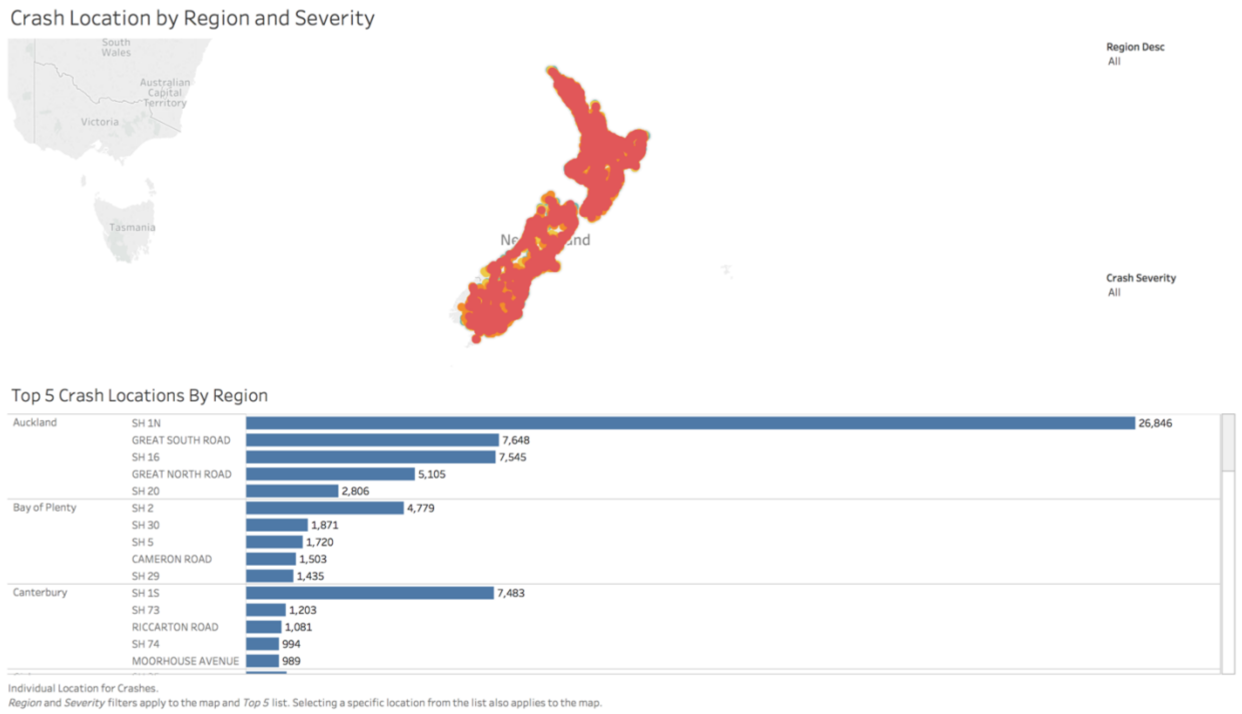
Crashes per Region

	Non Injury	Minor	Serious	Fatal	Grand Total
Auckland	172,094 38.68%	44,892 31.87%	7,796 22.33%	988 16.87%	225,770 36.03%
Waikato	44,251 9.95%	15,476 10.99%	4,229 12.11%	1,101 18.80%	65,057 10.38%
Canterbury	41,599 9.35%	17,155 12.18%	4,774 13.67%	697 11.90%	64,225 10.25%
Wellington	44,213 9.94%	12,300 8.73%	2,820 8.08%	317 5.41%	59,650 9.52%
Manawatu/Wanga..	23,657 5.32%	7,903 5.61%	2,354 6.74%	507 8.66%	34,421 5.49%
Bay of Plenty	24,859 5.59%	6,979 4.95%	2,101 6.02%	475 8.11%	34,414 5.49%
Otago	20,851 4.69%	9,601 6.82%	2,976 8.52%	274 4.68%	33,702 5.38%
Northland	16,766 3.77%	5,739 4.07%	1,819 5.21%	438 7.48%	24,762 3.95%
Hawkes Bay	16,609 3.73%	5,481 3.89%	1,509 4.32%	288 4.92%	23,887 3.81%
Nelson/Marlborou..	12,332 2.77%	4,254 3.02%	1,235 3.54%	202 3.45%	18,023 2.88%
Southland	9,733 2.19%	4,509 3.20%	1,421 4.07%	182 3.11%	15,845 2.53%
Taranaki	9,331 2.10%	3,616 2.57%	952 2.73%	182 3.11%	14,081 2.25%
Gisborne	5,168 1.16%	1,555 1.10%	408 1.17%	72 1.23%	7,203 1.15%
West Coast	3,490 0.78%	1,404 1.00%	520 1.49%	132 2.25%	5,546 0.89%
Grand Total	444,953 100.00%	140,864 100.00%	34,914 100.00%	5,855 100.00%	626,586 100.00%

Color intensity corresponds to overall proportion of crashes which is consistent with population density.
Nelson shows no crashes because the region is merged with Marlborough. The same is the case for Rotorua ad Waikato.
Interestingly, Waikato has the most fatal crashes, closely followed by auckland and Canterbury.
All other severity levels show similar distribution with Auckland, Waikato, Canterburry and Wellignton concentrating the most cases, in that order.

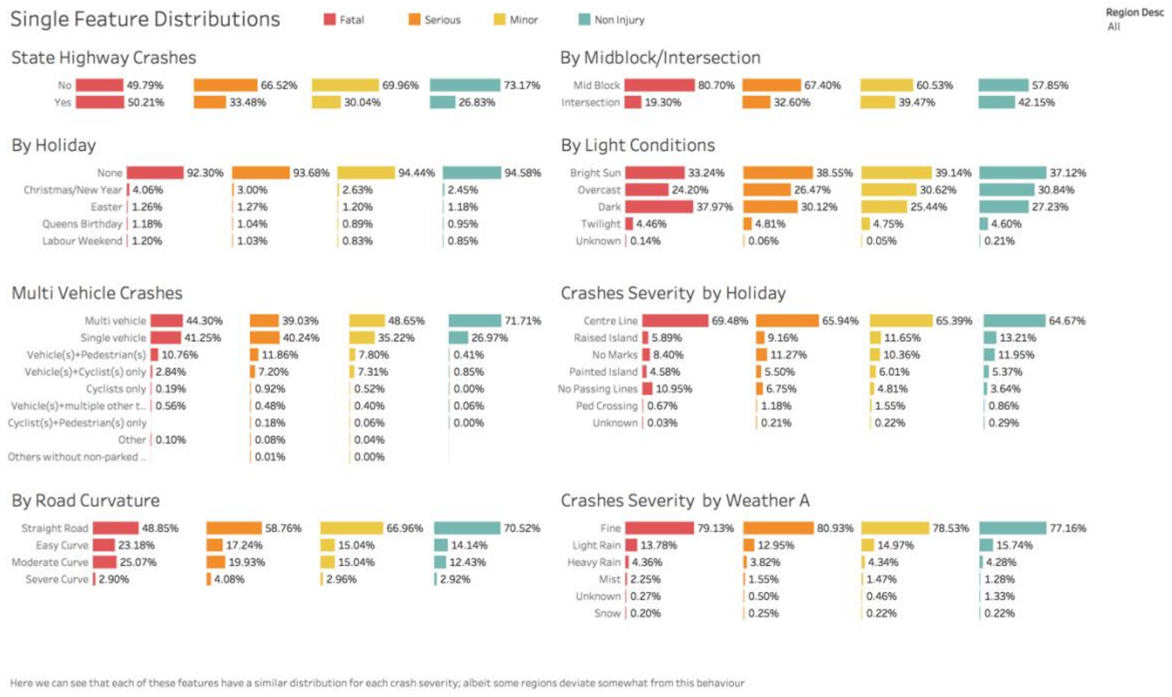
Distribution of crashes along regions for each severity. Regions are sorted by total number of crashed in descending order.

The second dashboard marks each crash in a map, colored by its severity and also presents –for each region– the top 5 locations with the greatest number of crashes. The map and the list are linked to offer a more interactive exploration.



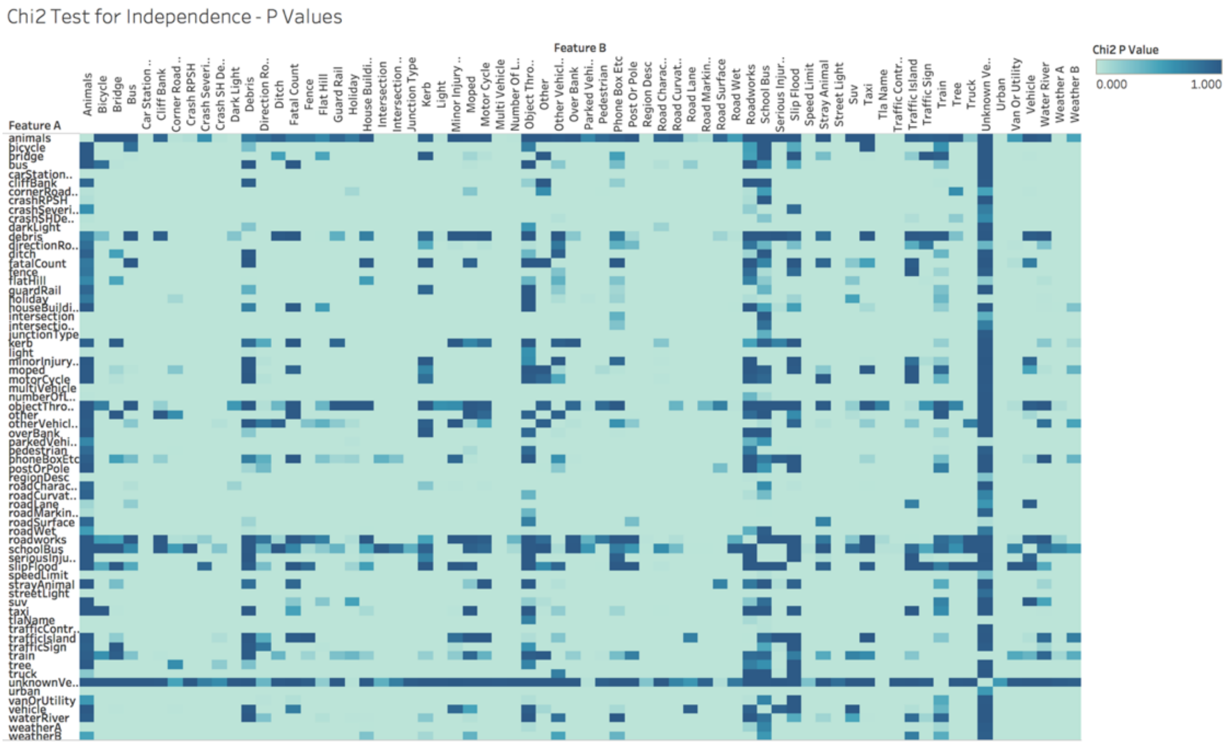
Individual Location for Crashes.
Region and *Severity* filters apply to the map and *Top 5* list. Selecting a specific location from the list also applies to the map.

The third dashboard shows the distribution of 8 different feature with regard to Crash Severity. Showing that– for each of these features– the distribution is very similar across crash Severity levels. This can be visualized by noting how similar are the 4 color bars (one for each crash severity level), in each feature's chart. Additionally, there's a region filters that allows to look at a specific region. This filter can be used to also see that all regions behave more or less the same.

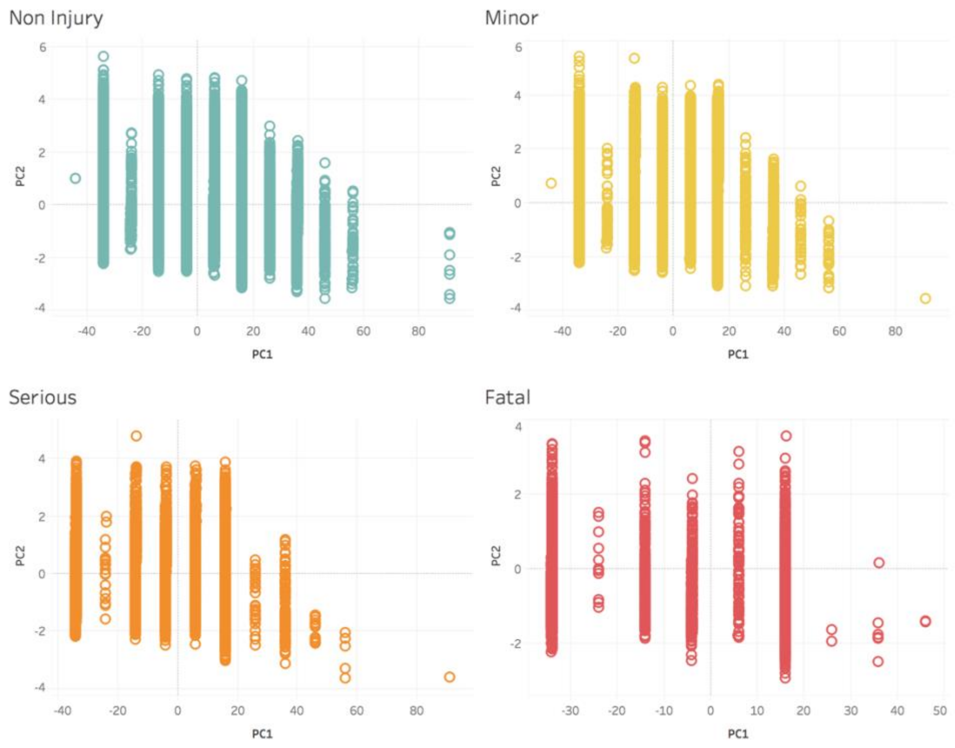


These three dashboards provide a great overview on the behavior of the dataset. Although we were note able to identify a specific characteristic that could help us understand what drives a given crash severity level; we do notice that there's a common trend both across crash severity levels and regions.

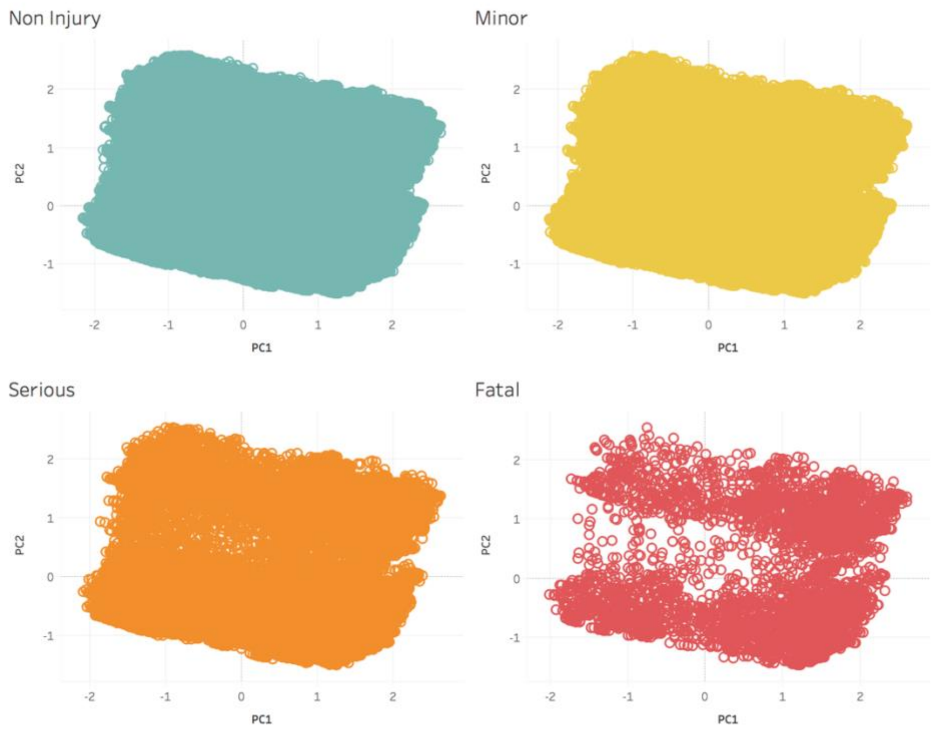
This trend motivated us to explore the correlation for every pair of features by means of a Chi2 test of independence. Doing so we found that every feature was dependent on every other feature as shown by this matrix of the corresponding p values:



And also, to try to reduce the dimensionality by applying PCA –after applying One Hot Encoding to the categorical features. However, the results showed that event two first 2 PCs –which account for more than 97% of the total variability– were able to create a separation between classes:



What's more, after scaling the data through MinMax, the results were even less interpretable and the first two PCs accounted only for 23% of total variability:



Algorithms and Techniques

Considering that most of the features in the dataset are categorical and that it is not clear which ones are the most meaningful or relevant, we will focus on decision trees as base/weak learner and train two ensemble models; one for bagging and one for boosting. Specifically, we will train and optimize a Random Forest and an AdaBoost GBM.

Random Forest is a great and powerful tool. It leverages the power of decision trees and adds different levels of randomness to overcome their tendency to overfit. This randomness happens when sampling or bootstrapping the instances used to train each tree in the ensemble –which makes it a bagging algorithm– and again, when a random sample of features are used at each split of each node of each tree. This generates a strong learner able to produce more accurate and stable predictions that any of the weak learners it is made of.

AdaBoost is a boosting algorithm. It is an ensemble of weak learners trained sequentially where each learner or stump is trained with a special focus on samples that were wrongly classified by the previous weak learner. Thus, boosting the performance achieved so far.

These two algorithms are good option for a multiclass classification problem. However, Random Forest is sensible to class imbalance and AdaBoost is sensible to noise and outlier. All these particularities are important, and we will address them by training both models on three variations of the dataset: the original (full) dataset, an undersampled variation and an oversampled variation. These variations and the way we've produced them are explained in detail in the next section.

Our approach will be the same for each of these two algorithms.

First, we will train a model with all default values –for each variation of the dataset– to get a baseline. Both on model parameters (like number of max depth of trees in the forest) and performance metrics.

Then we will do a grid search using the variation that produced the best results and with relevant hyperparameter values. The grids of parameters and values will be determined by analyzing the results of the baseline model.

Finally, we will take the best model for each algorithm and compare their performance.

Benchmark

To evaluate the performance of the trained models we will use two reference values.

The first one is the performance of an ANN that was trained for our problem specifically specified in this paper:

Severity Prediction of Traffic Accident Using an Artificial Neural Network

2016 - Sharaf Alkheder; Madhar Taamneh; Salah Taamneh

The abstract of the paper describes a final ANN with a test accuracy of 74.6% that we will for reference.

The second reference will be twofold. We will train a basic Naïve Bayes to baseline our ensemble models and use it for comparison. Also, we will use the baseline ensemble models trained with all default values as a second point of comparison against the result of a grid search.

For this second reference, we will use the average scores across all classes as well score values at the class level.

The accuracy and averaged F1 score for the Naïve Bayes and each variation of the dataset are as follows:

	Accuracy	F1-Score
Original variation	0.6339	0.6445
Undersampled variation	0.4198	0.3830
Oversampled variation	0.4296	0.3909

The performance of the baseline models is discussed in the *Results* section below.

All the work done form this point forward is available on the second notebook available in the github repository.

Data Preprocessing

As it was mentioned in the *Data Analysis* section above, we have already explored each feature and accounted for any possible missing or inconsistent value. And we have removed all features we consider to be irrelevant to our problem.

All remaining features in the dataset are either counter or categorical features.

The counters don't require any special treatment. They are all features that have small integer values.

All categorical features have been binarized through *One Hot Encoding*. This is a necessary step for SKLearn to be able to take the dataset as input. Although Random Forest can work with categorical features, SKLearn doesn't.

After having an OHE version of the original dataset, we proceeded to generate the under and oversample variations.

For this step we have relied on the *imblearn library.*

The undersampled variation simply has, for each class, as many samples as the minority class –in this case class *F*

The oversample variation has been created by following two steps. First, we have reduced the two majority classes by taking a random sample of each (20% of class *N* and 60% of class *M*). This was done to reduce the overall size of the result dataset. Then using the SMOTE method, we created synthetic samples for all classes except the majority one. Ending up with the same number of samples as the majority class.

The undersampled variation of the dataset contains 5,855 samples for each class. As many as the number of samples of class *F*.

The oversampled variation of the dataset contains 88,989 samples for each class. As many as the number of samples in the (sampled 20%) majority class.

Finally, although we have used PCA to uncover latent features, the results we obtain were not useful to our goal. Therefore, this preprocessing step is not necessary prior to training the models.

Implementation and Refinement

For the two chosen algorithms (RF and AdaBoost). We have followed the same process:

1. Train one baseline model (with default parameters) using a 25/75 train/test split for each variation of the dataset. The result of this step is one trained model for the original variation; one for the undersampled variation; and one for the oversampled variation.

2. Analyze performance metrics for the baseline models and select the model and variation that performs best. In this step we have consider different metric. Specifically, we have compared overall accuracy – remember that one of our benchmarks is on accuracy– averaged Precision, Recall and F1 score across all target classes and the same metrics at the class level. This was done this way because although we want to achieve a high F1 score, we are in a multiclass situation and so, we want the score to be representative of all classes. And considering the natural imbalance of class distribution, this becomes even more relevant.

3. Do a grid search using the best performing variation and a grid of parameters derived from the characteristics of the best performing baseline model. The grid search will choose the best estimator based on the average F1 score from a 3-fold cross validation.

The grid search for RF was done with the following grid of parameters:

- 'max_depth': [70, 80, 90, 100, 110, 120]
- 'min_samples_split': [2, 50, 100, 500]
- 'n_estimators': [20, 50, 100, 200]

The grid search for AdaBoost was done with the following grid of parameters:

- 'learning_rate': [.3, .5, 1, 1.3, 1.5]
- 'n_estimators': [100, 200, 300, 400, 500]

4. Analyze the predictions and performance of the best performing model from the search.

The details of this process can be seen in the second notebook under section 4.

III. Results

The results obtained by following the process described above were very interesting. In this section we discuss first the performance metrics for the baseline RF models. Then we do the same for the AdaBoost baseline models.

Next, we discuss bot grid searches. First the one for RF and then the one for AdaBoost.

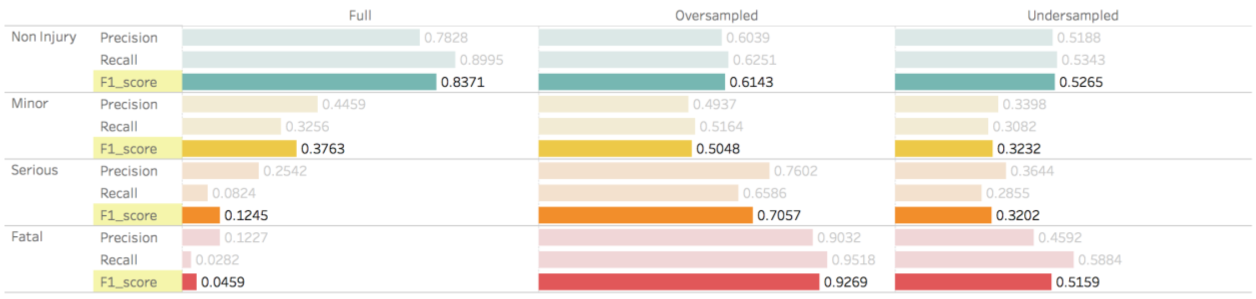
Finally, we compare the results of each grid search and its corresponding best estimators.

Random Forest Baseline Models

The Accuracy and averaged F1 score for the RF baselines are as follows:

	Accuracy	F1-Score
Original variation	0.7164	0.6861
Undersampled variation	0.4283	0.4207
Oversampled variation	0.6878	0.6879

The in-class performance metrics are as follows:



As we can see, the best accuracy is obtained with the original variation closely followed by the oversampled one. However, the best F1 score is obtained with the oversampled variation, although only slightly better the one obtained with the original variation.

Also, by looking at the performance metrics at the class level, we clearly see that the average F1 score of the original variation is mostly due to the score for class *N* while the scores for the other classes are much lower. On the other hand, the F1 score for the oversample variation is generated by F1 score at the class level that are much more balanced between each other. So, the F1 score of 0.6879 for the oversampled variation is much more representative of what is going on at the class level.

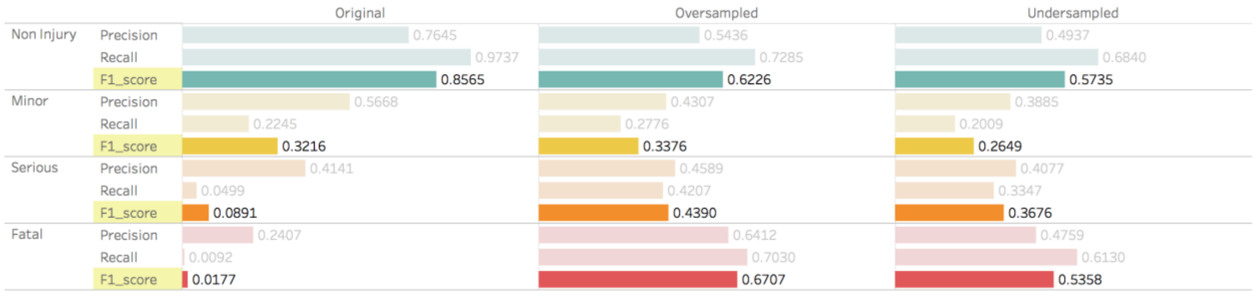
Moreover, for each class in the oversample variation, the F1 score turns out to be a very good balance between Precision and Recall. All these details make the oversample variation the best candidate on which to perform the grid search.

AdaBoost Baseline Models

The Accuracy and averaged F1 score for the AdaBoost GBM baselines are as follows:

	Accuracy	F1-Score
Original variation	0.7443	0.6853
Undersampled variation	0.4565	0.4341
Oversampled variation	0.5326	0.5175

The in-class performance metrics are as follows:

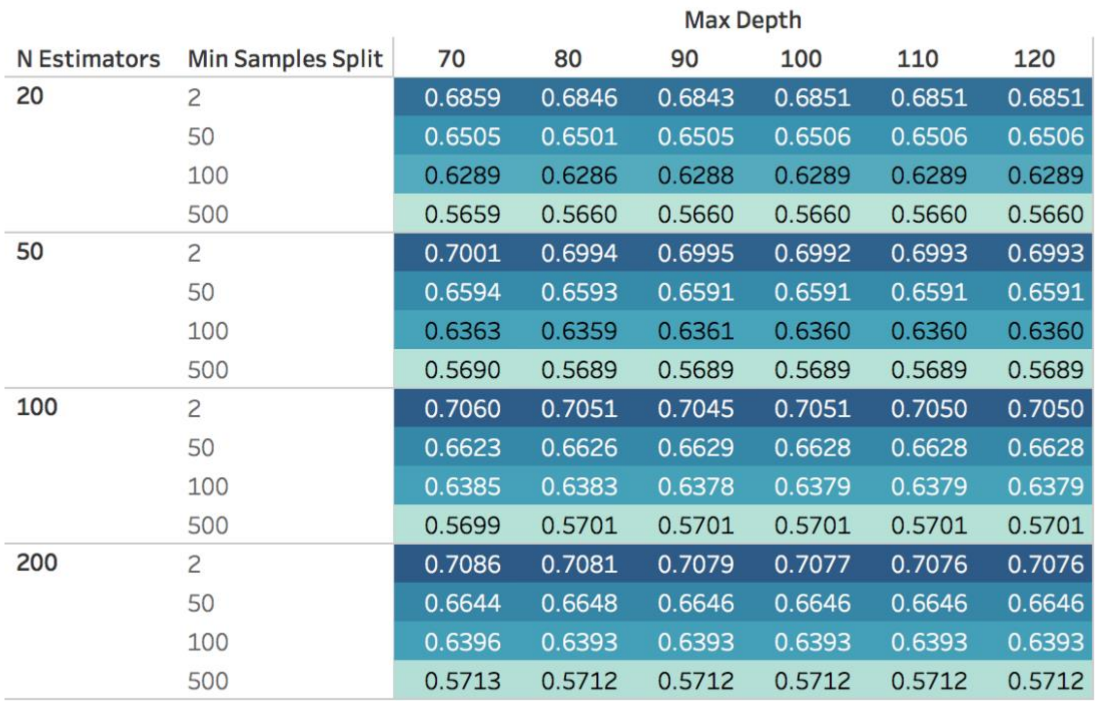


In this case, both the best accuracy and averaged F1 score is obtained with the original variation. Nevertheless, as with the random forest, this is mostly driven by the score for class *N*. And since we want to have a score that is evenly representative of each class we can't choose this variation.

Instead, we will choose the oversampled variation that, although has a much lower value, its behavior across classes is very similar as the case for RF.

Random Forest Grid Search

Following are the average F1 score for each combinator of parameters in the grid:



		Max Depth					
N Estimators	Min Samples Split	70	80	90	100	110	120
20	2	0.6859	0.6846	0.6843	0.6851	0.6851	0.6851
	50	0.6505	0.6501	0.6505	0.6506	0.6506	0.6506
	100	0.6289	0.6286	0.6288	0.6289	0.6289	0.6289
	500	0.5659	0.5660	0.5660	0.5660	0.5660	0.5660
50	2	0.7001	0.6994	0.6995	0.6992	0.6993	0.6993
	50	0.6594	0.6593	0.6591	0.6591	0.6591	0.6591
	100	0.6363	0.6359	0.6361	0.6360	0.6360	0.6360
	500	0.5690	0.5689	0.5689	0.5689	0.5689	0.5689
100	2	0.7060	0.7051	0.7045	0.7051	0.7050	0.7050
	50	0.6623	0.6626	0.6629	0.6628	0.6628	0.6628
	100	0.6385	0.6383	0.6378	0.6379	0.6379	0.6379
	500	0.5699	0.5701	0.5701	0.5701	0.5701	0.5701
200	2	0.7086	0.7081	0.7079	0.7077	0.7076	0.7076
	50	0.6644	0.6648	0.6646	0.6646	0.6646	0.6646
	100	0.6396	0.6393	0.6393	0.6393	0.6393	0.6393
	500	0.5713	0.5712	0.5712	0.5712	0.5712	0.5712

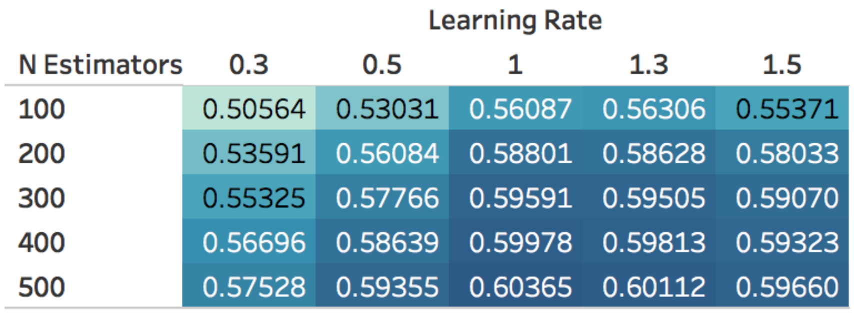
The best estimator was trained with parameters *max_depth=70*; *n_estimators=200*; and *min_samples_split=2*.

It's interesting to note that for each *n_estimator* and *max_depth* we considered, the highest scores are for *min_samples_split=2*. This becomes evident by looking at the color stripes from the viz –plus looking at the actual score values, of course.

When we make predictions on the test set (for the oversampled variation) using the best estimator, we got an accuracy of 0.7227 and an F1 score of 0.7231.

AdaBoost Grid Search

Following are the average F1 score for each combinator of parameters in the grid:



	Learning Rate				
N Estimators	0.3	0.5	1	1.3	1.5
100	0.50564	0.53031	0.56087	0.56306	0.55371
200	0.53591	0.56084	0.58801	0.58628	0.58033
300	0.55325	0.57766	0.59591	0.59505	0.59070
400	0.56696	0.58639	0.59978	0.59813	0.59323
500	0.57528	0.59355	0.60365	0.60112	0.59660

Looking at the CV results for the averaged F1 score of the three-fold validation, we find values between 0.50564 and 0.60365. With a maximum at *n_estimator=500* and *learning_rate=1*.

In this case, the interesting part comes from noticing that for each values of *n_estimators*, the best test score is achieved with a learning rate of 1. This in a way, makes sense. The algorithm itself is already assigning weights to correctly and incorrectly classified sampled so that the next weak learner is trained so as to improve the misclassifications from the previous one.

For this best estimator, the accuracy on the test set is 0.6089 and the averaged F1 score is 0.6058.

Random Forest and AdaBoost Compared

Finally, after optimizing each model through a grid search, we are in a position to compare between them and with the benchmark.

Since we've trained both algorithms using the oversampled variation in order to overcome the imbalance of classes, we show the performance metrics only for that variation:

	Accuracy	F1-Score
Naïve Bayes	0.4296	0.3909
Random Forest Baseline	0.6878	0.6879
Random Forest Grid Search	0.7227	0.7231
AdaBoost Baseline	0.5326	0.5175
AdaBoost Grid Search	0.6089	0.6058

For both metrics, the best performing model is the Random Forest after doing the grid search. In a way, this was expected. At least the fact that the best performing model would be one optimized, in this case, through a grid search.

But perhaps the most interesting thing to note, is the difference of approach between the two algorithms.

While AdaBoost tries at each step to improve the misclassifications made by the ensemble of previous weak learner –that are stump that make predictions based on a single feature– Random Forest is a deterministic method that grows trees by finding the split that would produce the greatest reduction of impurity.

By doing this, Random Forest develops very precise or overfitted trees. But by adding many trees with a bag of samples, and by bootstrapping the features at each split, it overcomes the overfitting of individual trees, creating a very strong model.

The fact that the overall Accuracy score is only 0.72 indicated that the available data lack predictable power to bring this metric above the first benchmark we chose of 74.6% accuracy for the ANN.

IV. Conclusions

The initial goal of this project was to train a model that would be able to predict the severity. While working on this problem and dataset I learned many things about working with categorical data, multiclass classification and class imbalanced.

I has to explore deeply and thoroughly these features to make sure I wasn't introducing any bias into the model. I had to remove featured because they produced data leakage and had to treat both missing and inconsistent values for different features.

Understanding the relevance of the different features was particularly hard for me because of their categorical nature. The results obtained by applying PCA actually provided more question rather than answers. But as I kept working on the training of the models I understood what could be happening and stopped trying to understand every bit of the data.

The end result of the work done is a Random Forest model comprised of 200 trees with a max depth of 70 levels and a min sample split parameter of 2. Which makes each tree very deep and specific; and most surely overfitted to the bag of samples it was trained on.

Together, the ensemble of trees is able to produce a mere 0.72 accuracy and 0.72 F1 score. While the F1 score is balanced across classes –meaning that the F1 score for each class has a similar value to the averaged one– the overall performances are below par with the ANN benchmark we chose at the beginning of the project.

This makes our model relevant, but not usable. At least not for the purpose it was originally intended. Namely, to be used by the different NZ emergency agencies to predict the severity of a crash a dispatch an appropriate response.

Nevertheless, there is much room for improvements. For example, a few other algorithms that could prove useful are SVM, XGBoost and LGMB. These could also be combined –including the Random Forest.

We could also train a binary classifier for each class to have a more customized one-vs-all approach. And also try other evaluation methods like AUC/ROC and Precision/Recall curves.

In this case we could implement a voter for the final classifier. And we could also leverage a deeper analysis of predicted probabilities for each class. Helping understand better what are the most representative charactersitics of each class.

In parallel with all the above, we could explore the path of feature engineering, through other dimensionality reduction techniques and unsupervised learning that could help uncover hidden structure in the data.

For the imbalance nature of the dataset, some cost sensivity algorithm could be very helpful.

And last but not least, we could try other libraries that have other strong and weak point. Like H2O for example, which can handle categorical features as is intended for parallel computing which would help with the total training time of the models.

On a personal note, I am partly satisfied with the results, very satisfied with the work done and what I've learned along the way. So, I don't consider I'm done with this project and dataset and look forward to implementing some of the improvements mentioned above.

I am thankful to my mentor, James Lee; the people behind the nanodegree and my fellow classmate with whom I've had valuable conversations and shared knowledge.

Thank you.