# Practical Machine Learning – Assignment 2

Name: Shane Quinn Date: 10/12/2020

Student Number: R00144107

## Abstract

For this project I looked at the Global Terrorism Database (GTD) [1], the dataset can be found on Kaggle [2]. The goal of this project was to develop a model that will predict a terrorist group name by features provided. I explored several different pre-processing techniques and machine learning models to find the best fitting model for this dataset. I also investigated robust validation strategies such as nested cross-fold and hyper parameter tuning methods such as grid search. A large portion of this report focuses on dealing with imbalance in the dataset.

## Introduction

GTD define terrorism as *“The threatened or actual use of illegal force and violence by a non-state actor to attain a political, economic, religious, or social goal through fear, coercion, or intimidation”* [1]. This dataset holds over 180,000 entries of acts of terrorism with 135 feature columns. The dataset holds both numerical and text data, however most categorical information is already encoded in numerical format.

The target value of this dataset is the column titled ‘gname’ which holds the terrorist group name associated with that terrorist attack.

## Research

A potential problem encountered when designing a predictive classification model is imbalance in the data set. Having a large number of entries for a majority class while few of a minority class can likely cause the minority class to perform poorly. In many cases the minority class performance is more important. We discuss model scoring systems in more detail in a later section, briefly however, model accuracy is not a good enough indication of model performance when dealing with an imbalanced dataset. In cases like this it is more beneficial to use metrics such as confusion matrices or F1 scoring.

For this project increasing the number of target classification values or terrorist groups, leads to a great deal of imbalance in the data set. There are a wide variety of methods to deal with imbalance in a dataset and in this section, I’ll describe the techniques investigated. A number of these methods are based around resampling which can be an effective strategy to combat imbalance [3]. While several methods described below are based on the generation of new samples.

Another method for dealing with imbalance in a dataset is cost-sensitive learning [4]. This does not involve the removal or generation of samples, the dataset is not modified. Instead, misclassifications are dealt with on a weighted bases, a minority class misclassification would cost more than a majority class misclassification.

Experimentation and evaluation of results are discussed in later sections.

### Random Over Sampling

Random over sampling is perhaps the simplest technique for dealing with imbalance. It is the process of randomly adding samples to a minority class. For a given class, a sample is picked at random and duplicated, this process is repeated ‘x’ number of times, by default ‘x’ is the difference between the number of entries in the majority class and the minority class we’re concerned with. This process is repeated for each minority class in our multi-classification problem.

The main consequence of random over sampling is causing the model to overfit [5].

### Random Under Sampling

Random under sampling is the process of randomly deleting samples from a majority class. For a given class, a sample is picked at random and deleted, this process is repeated ‘x’ number of times, by default ‘x’ is the difference between the number of samples in our majority class and the minimum number of samples in any class. This process is repeated for each class except the minority class(s) with the minimum number of samples.

If we are dealing with a multi-class problem where the majority class have 5000 samples and the minority class has 10, the majority class would have 4990 samples randomly deleted. This has the obvious drawback of removing a large amount of useful information. We can also set the desired number of samples for each class.

### SMOTE

Synthetic minority over-sampling technique (SMOTE) is a method of dealing with imbalance first described in [6]. SMOTE is the process of creating new points in feature space and is applied to the minority class(s). SMOTE identifies a sample in feature space and finds it’s K nearest neighbours of the same class. One of the nearest neighbours is selected at random and a point is generated arbitrarily between this point and our original point in feature space.

One of the drawbacks of SMOTE is it can create noisy instances between outliers and the rest of the data. This can be solved by combining SMOTE with an under-sampling method, discussed below.

### Tomek Links

Tomek links is a form of under sampling, it focusses on areas of overlap between the majority and minority classes. For a given sample in the majority class, if it is closer in feature space to a minority class it is considered a Tomek link and can be removed. This makes the difference between classes more distinguishable.

### Random Over and Under Sampling

This is a combination of random over sampling and random under sampling discussed above. The default over sampling implementation is to oversample the minority class to a point where it has the same number of samples as the majority class. When combining over and under sampling this must be considered, and default parameters must be altered so the majority and minority classes can meet in the middle.

### SMOTE and Tomek Links

A common implementation is to combine two of the methods described above. “The combination of SMOTE and under-sampling performs better than plain under-sampling” [6]. This method oversamples first using SMOTE and then prunes the overlapping area by removing the Tomek links.

### SMOTE and Edited Nearest Neighbours

This method is like the previous method in that it oversamples first using SMOTE, then we under-samples using edited nearest neighbours. Edited nearest neighbours identifies k nearest neighbours of the majority class in question (k = 3 by default). If the sample in question is then misclassified as a minority class, it is removed. This is then repeated for K = 1. [7]

Both SMOTE with Tomek Links and Edited Nearest Neighbours are attractive techniques for creating a more balanced dataset while minimizing noise from the samples generated using SMOTE.

### Adaptive Synthetic

Adaptive Synthetic (ADSYN) is like SMOTE in that it synthetically generates new samples for the minority class(es), both use a k nearest neighbours’ approach. ADSYN differs however in that it concentrates on generating more samples for classes that are hard to learn [8]. This method helps to adaptively shift learning boundaries, helping to increase the performance of class predictions that otherwise would perform poorly.

The algorithm decides how many samples to generate for a minority class by analysing its density. For example, classes that are more spaced out in feature space would have more samples generated than classes that have a denser population.

### Cost Sensitive Learning

This is the method of assigning varying costs to misclassifications in the form of a cost matrix. A cost matrix is a derivative of a confusion matrix

|  |  |  |  |
| --- | --- | --- | --- |
|  | Actual Negative | | Actual Positive |
| Predicted Negative | True Negative | False Negative | |
| Predicted Positive | False Positive | True Positive | |

Figure – Binary Confusion Matrix

Each entry in the confusion matrix is given a different cost, depending on its weight. Here we can heavily penalise a misclassification of a minority class. Not all machine learning algorithms work with cost sensitive learning. The implementation of a Decision Tree Classifier using cost sensitive learning is described below in the methodology section.

## Methodology

### Baseline

To establish a baseline the goal was to pre-process the dataset to a point where I could test several machine learning models to see which prove the most promising. I then concentrate on tuning these models.

#### Pre-processing

The original dataset has over 180,000 entries with 135 features, these features include numerical and text data. The first step of pre-processing I took was to identify the most prolific terrorist groups, seen in the table below

|  |  |
| --- | --- |
| Terrorist Group | Number of Entries |
| Unknown | 82782 |
| Taliban | 7478 |
| Islamic State of Iraq and the Levant (ISIL) | 5613 |
| Shining Path (SL) | 4555 |
| Farabundo Marti National Liberation Front (FMLN) | 3351 |
| Al-Shabaab | 3288 |
| New People's Army (NPA) | 2772 |
| Irish Republican Army (IRA) | 2671 |
| Revolutionary Armed Forces of Colombia (FARC) | 2487 |
| Boko Haram | 2418 |

Table – Most Prolific Terrorist Groups

As the goal of this project is to create a model which will predict the terrorist group name from a given feature vector, all entries where the terrorist group was ‘Unknown’ are not helping us create a model. That is not to say that these features do not contain meaningful information which could be leveraged in a different implementation, for this project however I removed all features who’s target value was ‘Unknown’.

One of the benefits of working with this dataset is most categorical features are already encoded numerically. In the table below, we can see an example of how text data is represented in numerical format.

|  |  |
| --- | --- |
| attacktype1 | attacktype1\_txt |
| 1 | Assassination |
| 6 | Hostage Taking (Kidnapping) |
| 1 | Assassination |

Table – Dataset Encoding

As the machine learning models I am using to establish a baseline require numerical data, the next step of the pre-processing stage was to remove all text data, a total of 60 columns.

I then looked at dealing with missing values, again the models to be used are not able to handle missing values. Instead of removing rows with missing values I replaced all missing values with the Numpy ‘NaN’ value, this allowed me to replace the missing value with the mean across the column values using the SimpleImputer class in Scikit Learn. In my implementation this was a blanket approach covering all columns containing missing values. This is not ideal for columns that contain mostly missing values, or very few actual values as we’re creating values from very little data. I took this approach because columns where this is the case will most likely not carry much weight in helping predict the final target value and will be dropped in the feature selection stage, discussed below.

The next step was to standardise the data, using StandardScaler class in Scikit Learn. This method moves the mean to 0 and distributes the data over a Gaussian distribution with unit variance. This helps minimize the impact of some features purely because they are larger numbers. As a number of models are based on distance in feature space, consider one column ranges from 100,000 – 200,000 while another ranges from 0 – 1, the larger feature will have more of an impact on the overall prediction regardless of whether or not there is a genuine correlation.

The final step of the pre-processing stage (excluding dealing with imbalance) was dealing with outliners. Below is a box plot of the 10 most influential features before outlier detection and removal.

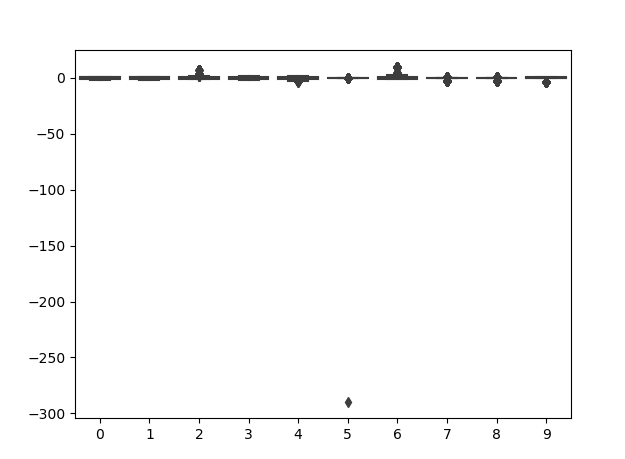


Figure – Box Plot before Outlier Detection & Removal

As you can see, for feature 5 there is an extreme outlier which makes it difficult to take much more meaningful information from the box plot. Using an isolation forest, I was able to identify outliers and remove them, setting the contamination parameter to 0.01 I predicted that 1% of features are outliers. Below is the resulting box plot.

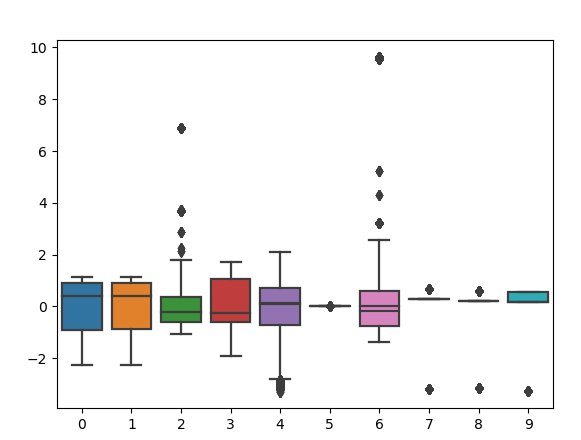


Figure – Box plot after Outlier Detection and Removal

As you can see there are still several outliers, I discuss tuning the contamination parameter further in the experimentation stage of the project. Isolation forests identify outliers by building many isolation trees which repeatedly, arbitrarily split the data in feature space until reaching an isolated instance. From this the model can identify which feature vectors are most isolated in feature space by how many random splits of the data it requires to isolate the instance. The less splits, the more isolated the instance is.

#### Models

I compared the following classification models when establishing a baseline

* K Nearest Neighbours
* Decision Tree
* Random Forest
* Support Vector
* Stochastic Gradient Descent
* Logistic Regression
* Naïve Bayes

I ran each of these models and calculated the F1 score for each, taking the best 3 performing models onto the next stage of hyper-parameter tuning. The F1 score is calculated by the following equation:

Recall is the true positives divided by the sum of the true positives and true negatives for a given class, this helps identify if a model is performing badly in one class (useful when dealing with an imbalanced dataset). Precision is a measure of how many instances were correctly predicted and is calculated by finding the true positives divided by the sum of true and false positives. F1 is a more accurate description of how well a model is performing than accuracy alone. It accounts for imbalanced in the dataset.

Generating a confusion matrix would also give us a good indication of model performance, particularly for minority classes as we can visualise true positives, false positives, true negatives and false negatives for each class. For this implementation of a few hundred classes, a confusion matrix is however not as attractive to use and so we rely on the F1 score for model performance.

### Experimentation

In this section we look at experiments carried out to improve the top performing models.

#### Feature Selection

When selecting which features to use, I used a tree-based feature selection methodology which enables us to rank the features by how much of an impact they have on a large number of decision trees. It does this by calculating the average reduction in gini impurity across all trees for each feature. The feature which leads to the smallest average reduction in gini impurity will rank lowest and is then removed.

#### Outlier Detection

Through experimentation I found that my implementation of outlier detection and removal (isolation forest) was in fact removing several valuable minority class instances. For this reason, I lowered the contamination parameter to .15%.

#### Hyper Parameter Tuning

Parameter optimization can be a computationally costly process. In the accompanied code I used GridSearchCV to carry out parameter optimization. Setting n\_jobs to -1 enables several jobs to run in parallel, taking advantage of multiple cores in the CPU. This is far quicker than running jobs in series.

GridSearchCV fits every possible implementation of the provided param\_grid dictionary and records the results. This allows us to see which parameters best fit our model.

#### Validation

Splitting the test and training data just once does not give us a good enough description of how well the model performs. This is because the model performance is dependant on how the data is split. To mitigate the effects of this we run cross fold validation. This is the process of splitting the data ‘k’ number of times using ‘k-1’ splits as training data and 1 split as test data. We repeat this ‘k’ number of times or until each split has been used as test data. This provides us with a more accurate description of how well a model fit.

For a more accurate description of a model’s accuracy, nested cross fold validation is used. This method involves an outer k-fold loop which is divided into training and test data as described above. The training data is then split again in an inner loop, this loop is then used for parameter optimisation with grid search as described above.

The obvious drawback of nested cross fold validation with grid search is how computationally expensive it is.

### Research

The dataset rebalancing methods discussed in the Research section above were mostly implemented using the Imbalanced Learn python package [9].

When implementing any rebalance techniques it’s important to note that the training data is the only data rebalanced. The test data was left alone, this is because introducing artificial samples into the test data can lead to over fitting. So, during cross fold validation the rebalance process must be called after the data is split on each iteration

#### Random Resampling

For a multi-classification problem like this, when implementing random resampling to a specific number of features we must provide a dictionary of the format:

In the accompanied code I iterate through the classes counts to find and populate the resampling dictionaries as needed. For example, I have one dictionary which has all classes that have less than 6 entries and the desired number of features is equal to 6. When fed into the random oversampling method, this will only randomly oversample these classes.

#### SMOTE

Recalling from the description of SMOTE above in the research section, SMOTE relies on identifying K nearest neighbours of a point in feature space. By default, the imbalanced learn implementation of SMOTE requires 6 nearest neighbours. Some of the minority classes had less than 6 instances and so I randomly over sampled these classes to a minimum of 6 features before piping into SMOTE. The option is also there to change the default K parameter to a lower number.

#### Cost Sensitive Learning

Certain scikit learn models have a class\_weight parameter which allow developers to implement a cost sensitive learning algorithm. In the accompanied code I compare the accuracy of a decision tree classifier implemented using the default settings and the cost sensitive learning approach. Scikit Learn gives us the option to specify the weights of each class individually or specify class\_weight = ‘balanced’ in which case the weights are automatically adjusted proportional to the frequency each class occurs. The less frequently a class occurs the more weight it carries.

## Evaluation

This section will explore the results generated from establishing a baseline, basic experimentation and the impact of various techniques for rebalancing the dataset.

### Baseline

Running the models listed below with their default parameters generated the following F1 scores.

|  |  |
| --- | --- |
| Model | F1 Score |
| KNeighborsClassifier | 0.913453 |
| DecisionTreeClassifier | 0.926972 |
| RandomForestClassifier | 0.938767 |
| SVC | 0.84387 |
| SGDClassifier | 0.648376 |
| LogisticRegression | 0.804904 |
| GaussianNB | 0.787674 |

Table – Baseline Results

As we can see the top 3 performing models on the dataset were:

* Random Forrest
* Decision Tree
* K-Nearest Neighbours

These 3 models were used for hyper parameter tuning and experiments to view the effect of rebalancing the dataset.

### Experimentation

Firstly, as part of my experimentation I investigated the number of target classification values to use. Using too few (1-10) terrorist groups led to the models performing extremely well which made it difficult to see the benefits of fine-tuning models and pre-processing steps. For this reason, I used 200 target values, this had the drawback of increasing imbalance in the dataset particularly between the most and least prolific terrorist groups. Dealing with imbalance in the dataset is described in detail in the research section of the report.

#### Outlier Detection

As mentioned previously, I found that performing outlier detection on this dataset removed several valuable minority class entries. Setting the contamination parameter of the isolation forest to 1% caused the removal of most instances in some minority classes, for example, one class went from 55 entries to 1. For this reason, box plots alone are not enough to go by when assessing a dataset for outliers.

#### Hyper Parameter Tuning

For hyper parameter tuning, I focused on tuning a Decision Tree Classifier using cost sensitive learning. Taking advantage of Grid Search CV I attempted to tune the ‘max\_features’ and ‘max\_depth’ parameters. I found that tuning the ‘max’\_depth’ parameter had little to no impact on the model, however tuning the ‘max\_features’ parameter to between 0.7 and 0.9 led to the best performing model. Best Performing F1 score was 0. 9016510672981728.

Below is the results and parameters used for the 2 iterations of nested CV using grid search.

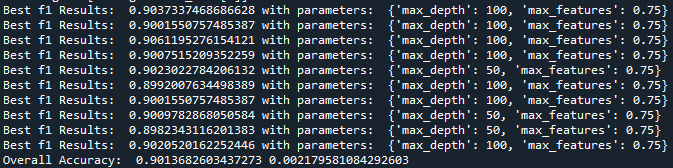


Figure – param grid = {'max\_depth':[10, 50, 100, 500], 'max\_features':[0.25, 0.5, 0.75, 1]}

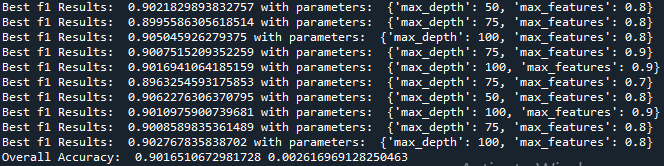


Figure - {'max\_depth':[50, 75, 100], 'max\_features':[0.5, 0.6, 0.7, 0.8, 0.9]}.

As we can see in the first Figure above, max\_features is 0.75 in each best case. Therefore for the second iteration of grid search I fine tuned the max\_features parameter to find the optimal value. The optimal value is the range of 0.7 – 0.9.

#### Feature Selection

I investigated the impact of removing the highest-ranking features (features ranked during pre-processing stage, isolation forest). The goal here was to identify if there was any 1 feature which carried the most weight when predicting our class value.

Figure – Impact of Top Feature Removal

As we can see from the graph above, removing the top 3 features has little impact on the overall performance of any of the models. Removing the top 5 features does have an impact on all models except the Decision Tree Classifier and Random Forrest Classifier.

### Research

As mentioned previously, outlier detection can lead to more imbalance issues. On closer inspection of the data, one of the classes most impacted by outlier detection was ‘Student Radicals’ which encompasses any act of terrorism committed by a student radical. As these are not tied to an actual terrorist group and there is very little correlation between entries, it makes sense that these instances are identified as outliers in the dataset.

From experiments carried out with different dataset rebalancing methods I found that they had little to no effect on the overall model performances. The graph below depicts the impact on various rebalancing methods tested on the top 3 performing models. The models perform relatively well before rebalancing.

Figure – Rebalance Methods Comparison

There is a dip in performance when using random under sampling to rebalance the dataset. This is because I used the default under sampling parameters which under samples each majority class to a point where it has the same number of samples as the minority class. The minority class had 2 entries in this example leading to a total of 400 entries in total.

The chart below shows the number of samples after each resampling method.

Figure – Rebalance Methods Number of Samples Comparison

Most over-sampling methods implemented add entries to minority classes, so they have the same number of samples as the majority class. Considering we have 200 potential classes, each which are artificially swelled to the same number of samples as the majority class, this leads to a massive increase in the total amount of samples. As we can see from the graph above, comparing ‘No Rebalance’ to all over sampling techniques shows that we create many times more samples than we originally had.

#### Cost Sensitive Learning

The table below shows a comparison of implementing a cost sensitive decision tree classifier vs the default decision tree classifier. Using the same training and validation data.

|  |  |
| --- | --- |
|  | F1 Score |
| Default DTC | 0.894648204 |
| Cost Sensitive DTC | 0.899045659 |

Table – Cost Sensitive Decision Tree Classifier Comparison

As we can see the cost sensitive DTC performs slightly better. This is because the cost sensitive approach penalises misclassification of minority class heavily when building the learning model.

## Conclusion

For this dataset, the preferred technique for tackling imbalance is under-sampling by removing Tomek Links. Because there is a large amount of imbalance and a lot of potential classes, all over-sampling techniques investigated lead to most instances in training data feature space being created artificially and the performance of our models does not improve. Adding artificial samples in this case just leads to an increase in computational expense, with no pay off. Removing Tomek Links reduces the computational resources needed by reducing the number of training features with no impact on the overall accuracy. Removing Tomek Links also helps rebalance the dataset by removing overlapping entries in the majority classes.

We also saw a small improvement when implementing cost sensitive learning Decision Tree Classifier instead of the default config.

Outlier Detection led to the removal of many useful minority classes, for this reason the contamination parameter in isolation forest method was lowered to .15%.

Tuning the ‘max\_features’ parameter to 0.7 – 0.9 in this implementation of a Decision Tree Classifier, alongside using cost sensitive learning leads to the best overall result.

### Future Work

As mentioned earlier in the report, we are removing 82782 features straight away because the group name is unknown. This accounts for approximately 45% of the dataset. A potential future project could be to investigate the use of an unsupervised learning technique such as clustering in combination with the techniques described above and evaluate if harnessing this data can lead to building a better performing predictive model.

# References

|  |  |
| --- | --- |
| [1] | “Global Terrorism Databse,” START, [Online]. Available: https://www.start.umd.edu/gtd/. |
| [2] | “Kaggle,” [Online]. Available: https://www.kaggle.com/START-UMD/gtd. |
| [3] | P. Branco, L. Torgo and R. P. Ribeiro, “A survey of Predictive modelling under Imbalanced Distributions,” Universidade do Porto, 2015. |
| [4] | C. Elkan, “The Foundations of Cost-Sensitive Learning,” *Proceedings of the Seventeenth International Joint Conference on Artificial Intelligence,* 2001. |
| [5] | A. Fernandex, S. Garcia, N. Krawczyk and F. Herrera, Learning from Imbalanced Data Sets, Springer, 2018. |
| [6] | N. Chawla, K. Bowyer, L. Hall and P. Kegelmeyer, “SMOTE: Synthetic Minority Over-sampling Technique,” *Journal of Artificial Intelligence Researc 16,* 2002. |
| [7] | D. Wilson, “Asymptotic Properties of Nearest Neighbor Rules Using Edited Data,” *IEE Transactions on Systems and Cybernetics,* 1972. |
| [8] | H. He, Y. Bai, E. A. Garcia and S. Li, “ADASYN: Adaptive Synthetic Sampling Approach for Imbalanced”. |
| [9] | “Imbalanced Learn Documentation,” [Online]. Available: https://imbalanced-learn.readthedocs.io/en/stable/index.html. |