Imperial College London

COURSEWORK 2

IMPERIAL COLLEGE LONDON

DEPARTMENT OF COMPUTING

Introduction to Machine Learning

Authors:
Faidon Mitzalis (00646578)
James Dorricott (01009425)
Harry Coppock (01800452)
Maleakhi Agung Wijaya (01806572)

Date: November 23, 2020

Contents

1 Part 1

We implemented our mini-library which passes all the tests on LabTS. Please see code.

2 Part 2

2.1 Addressing the Imbalanced dataset

Before applying machine learning techniques it is important to investigate the data on which you are training. A key stage is to determine the balance in this data, i.e., how many instances of each class there are. As shown in Figure 1 the provided claim dataset is clearly imbalanced. Data points which lead to a claim are 10 times less likely to appear in the dataset than datapoints which didn't lead to a claim. This creates two main problems when training a classifier. The first is that conventional evaluation metrics can be misleading, for example **Classification Rate** simply represents the performance of the majority class and provides little information on how well the model predicts the minority class. The second is that classifiers trained on this dataset simply learn to predict the majority label for all instances, essentially ignoring all the input data characteristics, as this minimises the loss function. This is clearly not desirable; this model trivially advises the insurance company that no customer will make a claim, offering no differentiation between prospective customers. These two problems related to the imbalanced dataset were addressed via the methods described in the next sections.

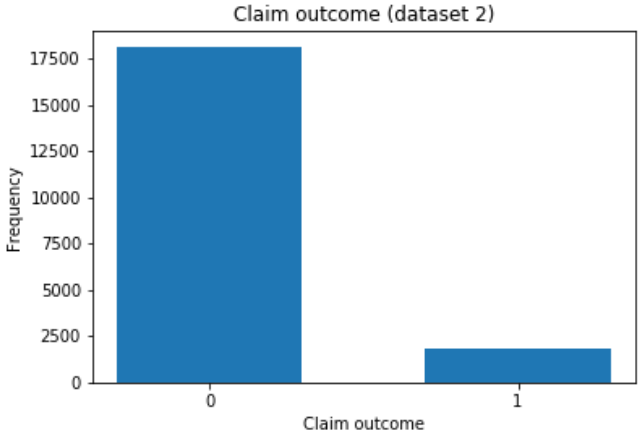


Figure 1: Highlighting the imbalanced nature of the vehicle insurance dataset

2.1.1 Evaluation Metrics

The success of the model in the market is reliant on how well it distinguishes between classes. Therefore, the metric we chose to optimise was the **Receiver Operating Characteristic Area Under the Curve** or **ROC-AUC**. This is because it is a direct

measure of how capable the model is at correctly distinguishing instances between classes. The closer the value is to 1, the better the model is at accurately classifying both classes. A value of 0.5 results when the model has no ability to distinguish between 0s and 1s.

Apart from using ROC-AUC, we also compute the **normalised confusion matrix** rather than the original confusion matrix by dividing each cell on the confusion matrix by the total number of examples in each class. This normalised confusion matrix provides a more informative result on model performance for class imbalances problem.

2.1.2 Up-Sampling

In order to prevent the classifier simply predicting the majority class for all inputs, it is important to train on a balanced dataset. The approach adopted was to increase the number of minority class data points, known as up-sampling. Rather than simply repeating the minority samples, which leads to a high chance of overfitting, data augmentation was applied to the minority samples, creating new synthetic samples which follow the same trend as the minority samples. Simply applying Gaussian noise, however, proved to be troublesome, as deciding on the level of variance of the noise was a difficult parameter to set. Too low, and the synthetic points will cluster too close to the original data points and the model will overfit again. Too high, and the new synthetic points may stray from the original data points, morphing the true underlying trend of the data.

Therefore, an alternative technique was used, **Synthetic Minority Oversampling**, or **SMOTE**. This technique was implemented by first choosing a random minority datapoint then calculating its k-nearest neighbours and generating new, synthetic points at random points along the vectors that join the selected datapoint with each neighbour. The process is repeated until the number of minority datapoints equals the number of majority data points. The pseudo code below represents our method in implementing this technique.

Applying SMOTE to the training data provided significant benefits to the classifier's performance. Figure 2 compares the ROC curve and AUC of our pre-optimised classifier before and after applying SMOTE to the training data. Figure 2a shows that training on unbalanced data results in a model that has no ability to distinguish between classes, with an AUC score of 0.5. Figure 2b shows that by simply apply SMOTE to the training data the model can now learn to distinguish between classes, getting an AUC score of 0.59. The improved performance is also evident in the confusion matrices in Tables 1 and 3. The model trained on unbalanced data predicts every test datapoint as 'will not make a claim', completely miss-classifying every 'made claim' data point. The model trained on the pre-processed data, however, more successfully identifies the made claim class, increasing the recall for this class from 0 for the pre-processed data to 10 for the post-processed data, as seen in Tables

Algorithm 1 Up-sampling: SMOTE

```
procedure SMOTE(DATASET)
    minority_dataset ← dataset where class == 1
    majority_dataset ← dataset where class == 0

    while len(minority_dataset) < len(majority_dataset) do:
        random_point ← random_choice(minority_dataset)
        neighbours ← k_nearest_neighbours(minority_dataset)
        joining_vectors ← random_point - neighbours
        random_value ← random(0,1)
        new_synthetic_datapoints = neighbours + random_value*joining_vectors
        dataset add new_synthetic_datapoints
    return dataset

```

2 and 4.

The dataset was also normalised, which ensured that the model was not biased towards features with higher magnitude. This squeezed all values to between 0 and 1, 0 representing the lowest value and 1 representing the largest value of the feature.

2.2 Implementation of Claim Classifier

The initial architecture of our neural network (pre-tuning) consisted of the following: 2 hidden layers, 20 neurons per layer, ReLU activation functions in the hidden layers, a sigmoid activation for the final layer, and a binary cross entropy loss function (as this was a binary classification problem). This architecture was chosen as it provided a good balance between providing enough flexibility to fit the data, not risking overfitting and unnecessarily high wall clock times. We chose ReLU activation functions for their efficiency and superior convergence properties, noting that ReLU helps with the **vanishing gradient** problem. For the final layer, we chose the sigmoid activation function as, given an input, we seek the probability a claim is made. Our hyperparameter search consisted of adjusting the number of epochs, hidden layers, and neurons, the results of which we discuss in section 2.4.

2.3 Architecture Evaluation

To evaluate and optimise our neural-network architecture a number of evaluation metrics were focused on:

- The **normalised confusion matrices** provided a quick summary of how well the model classified each class and was invariant to the balance of data.
- In the Classification report the **Recall** for both classes was the most important metric as it represents the proportion of correctly classified examples. A high

recall means the class is mostly correctly classified. We strive to achieve high recall for both classes.

- **ROC** curves and **AUC** as they give a direct measure on how well the model can distinguish between classes. The value of the AUC indicates the proportion of time the model can correctly distinguish between the two classes. In addition, the ROC curve can be used to determine the optimal threshold value and compare between models.

Confusion Matrix: No Pre-processing & No Upsampling		
	'No Claim' Predicted	'Made Claim' Predicted
'No Claim' Actual	1.0	0.0
'Made Claim' Actual	1.0	0.0

Table 1: Confusion matrix with no pre-processing and no up-sampling

Performance: No Pre-Processing			
Classification rate: 92%			
Class	Recall (%)	Precision (%)	F1 (%)
'No Claim'	92	100	96
'Made Claim'	0	0	0
Unweighted Averages:	46	50	48

Table 2: Key performance statistics, training data without pre-processing and up-sampling

Confusion Matrix: With Pre-Processing & Up-Sampling		
	'No Claim' Predicted	'Made Claim' Predicted
'No Claim' Actual	0.653	0.347
'Made Claim' Actual	0.594	0.406

Table 3: Confusion matrix with pre-processing and up-sampling

Performance: With Pre-Processing			
Classification rate: 63%			
Class	Recall (%)	Precision (%)	F1 (%)
'No Claim'	92	65	76
'Made Claim'	10	41	16
Unweighted Averages:	51	53	46

Table 4: Key performance statistics, training data with pre-processing

2.4 Architecture Fine-Tuning

We used the grid-search methodology for our hyper-parameter search. This took into account only the number of epochs, number of hidden layers and neurons per layer (uniform across all layers). The optimisation was aimed to maximise the ROC-AUC metric. We experimented and found that a search over activation functions and non-uniform number of neurons per layer did not significantly add to the performance. Furthermore, an exhaustive search including these hyper-parameters proved intractable with our computational resources. We did not apply dropout or a penalty/regularizer in the cost function. This was because as network is relatively simple, we found dropout reduces, rather than adds to the performance, since it reduces the network's capacity to learn, and causes it to under-fit. As the number of epochs is low and the network is relatively shallow, the danger of overfitting is low. These techniques are more appropriate for larger networks/a high number of epochs. Within the grid-search process, we applied three-fold, stratified cross validation to find the best performing model. The reason for stratified validation was that it ensures the proportion of each class in all the folds are approximately equal, ensuring that ROC-AUC value can be computed.

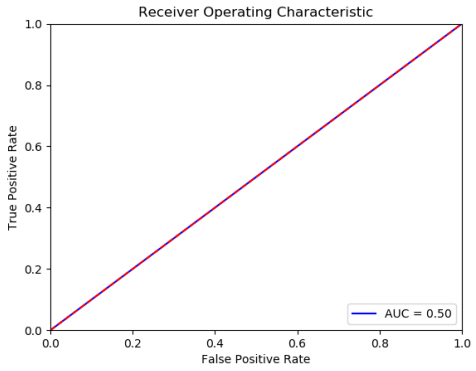
The optimal architecture after the grid search consisted of 5 hidden layers with 100 neurons per layer.

Confusion Matrix: With Pre-processing & Hyper-Parameter Optimisation		
	'No Claim' Predicted	'Made Claim' Predicted
'No Claim' Actual	0.573	0.427
'Made Claim' Actual	0.330	0.670

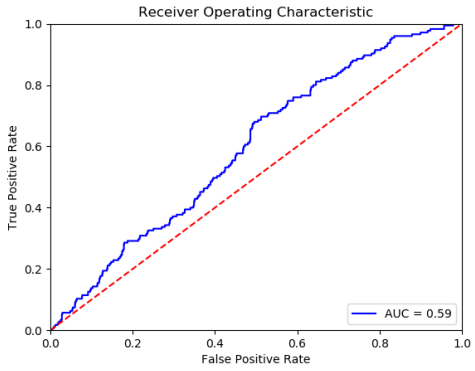
Table 5: Confusion matrix with pre-processing, up-sampling, and Hyper-parameter optimisation

Performance: With Pre-Processing after Hyper-Parameter Optimisation			
Classification rate: 58%			
Class	Recall (%)	Precision (%)	F1 (%)
'No Claim'	94	57	71
'Made Claim'	15	67	25
Unweighted Averages:	55	62	48

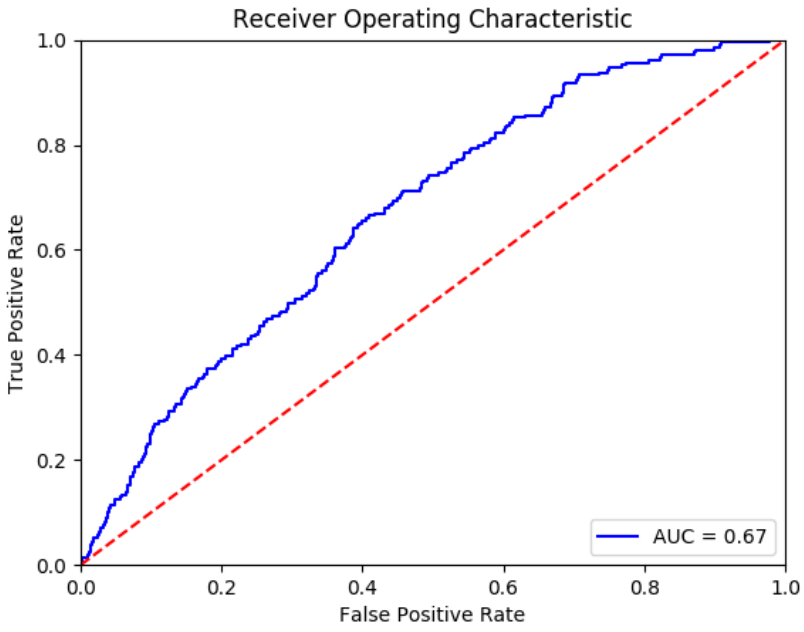
Table 6: Key performance statistics, training data with pre-processing and optimised hyper-parameters



((a)) Unbalanced dataset



((b)) Balanced dataset



((c)) Balanced dataset, tuned hyper-parameters

Figure 2: Comparing the effect of up-sampling the minority samples on the ROC metric (before and after hyper-parameter optimisation)

3 Part 3

3.1 Dataset Inspection

Using the Pandas library on the dataset of 80,000 vehicle insurance records (dataset 3), we identified the imbalance in the dataset in terms of the target variable. There were just 7,264 examples with a claim, and 72,156 examples without a claim. To get around this issue, we applied similar upsampling techniques discussed in the earlier sections. Pre-processing also involved replacing any null values with zeros.

A larger number of columns/features (35 in total) was available in this dataset. In particular, some of these features were categorical. While a selection of these were encoded using one-hot encoding (to transform the feature into numerical values), others were dropped due to the high number of classes and the computational expense of encoding them (e.g. vehicle make and vehicle type). The resulting input features after **one-hot encoding** was 43 (the choice of columns was set during training and kept the same for all test data sets). Even though most of the categorical features were weakly correlated with the target parameter, these were still fed to the Neural Network to avoid unnecessary feature engineering which would potentially miss non-linear correlations that the neural network can capture. For reference, we have made a cross-correlation plot of the target variable and these 43 features, which highlights these weak correlations (see leftmost column of Figure 3).

3.2 Pricing Strategy

Figure 4 shows a histogram of claim amount versus frequency. From this we can immediately see the positive skew in terms of claim amount, with claims exponentially tailing off in the lows 1,000s (EUR). Because of the presence of extreme positive outliers, the plot in Figure 4 excludes these outliers (the five largest claims are listed in Table 7). The median was selected for the severity constant as a better summary statistic for this distribution.

Our pricing strategy was relatively aggressive, and consisted of applying a 1% premium to the median claim amount (from those that claimed). The level of premium is key in the pricing strategy. A high level of premium increases the expected profit margin for a given contract and reduces the level of risk for the insurance company. However, a high premium implies less competitive prices which means that a smaller fraction of the market will be captured, reducing revenue. In reality we are expecting that in a competitive environment the level of premium set by the insurer would respond to market prices (avoiding negative premium which - ignoring the value of 'float' - leads to expected losses for the insurer).

3.3 Market Simulation

When testing against the AI market, we ranked 2nd, with a AUC of 0.69.

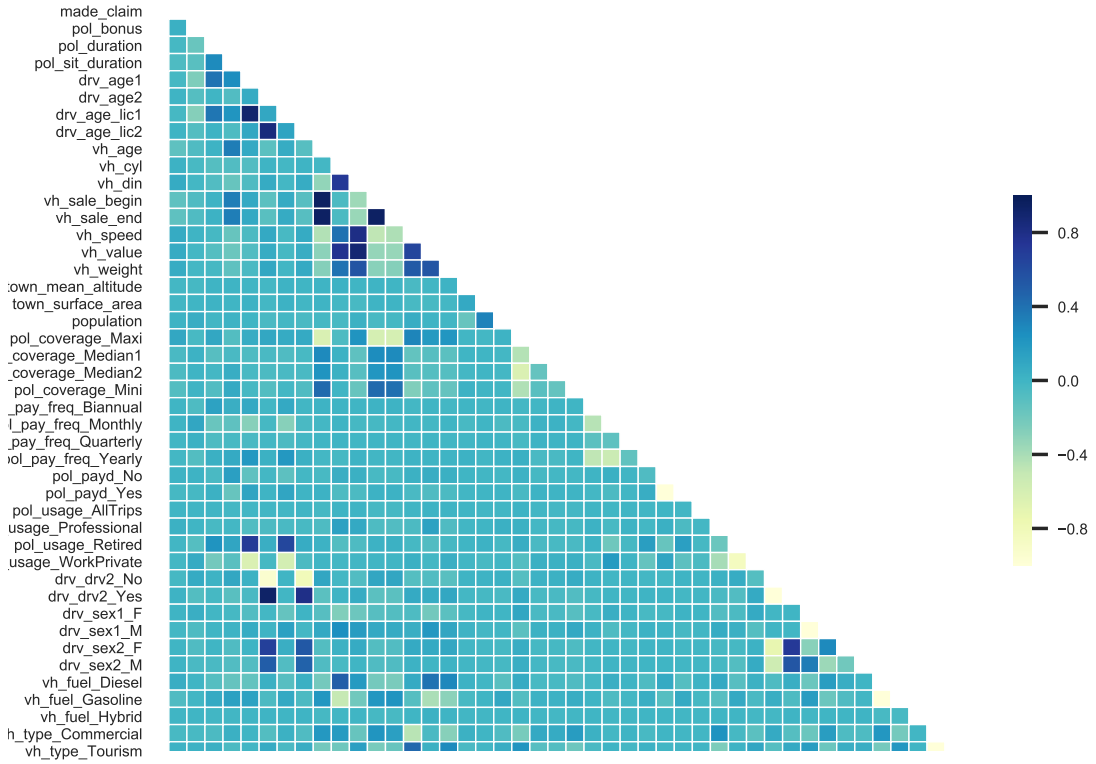


Figure 3: Correlation matrix of features and target varaible

3.4 Logistic Regression

A Logistic Regression model was trained on the same dataset in order to compare with the performance of the Neural Network. The Logistic Regression was implemented using the Scikit-Learn library (default parameters). The ROC-AUC achieved was 0.63, inferior to the equivalent Neural Network. In addition, it only manages to achieve fourth position on the market simulation. This highlights the particular benefit of our neural network in being able to predict very low probabilities for specific samples, allowing us to offer competitive prices to 'non-risky' prospective customers.

4 LabTS Notes

We'd like to note that we spent quite a significant proportion of our time debugging certain things. These are mentioned below. Please take this into consideration when marking.

There were a number of problems with the LabTS tests that required significant at-

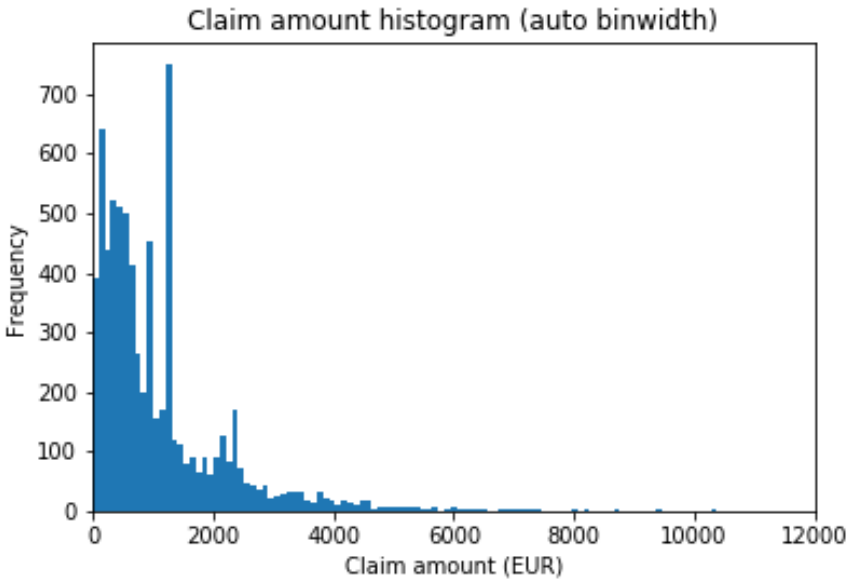


Figure 4: Frequency distribution of claim amount

Rank	Claim amount (EUR)
1	226,841.89
2	97,270.79
3	48,099.50
4	43,543.35
5	29,565.90

Table 7: Top five claims

tention. First, the spec said that `loss_fun` can take `"mse"` and `"bce"`. However, what the test script expected is `"MSE"` and it gave us 2/3 on the `Trainer` class tests. After we changed `"mse"` to `"MSE"`, we passed all tests, but this required a long time to debug.

In addition to this, there were problems with the instructions in the ”predict” function in class Claim Classifier (Part 2). The instructions in the comments mentioned that a probability of a positive class was expected. Therefore, we wrote most of the code assuming a probability to be returned (including the evaluate function). When we ran the test for Part 2, however, the test was failing because actually what was required was hard classification (0 or 1), not a probability. We therefore fixed this and also rewrote numerous functions in the file.

Lastly, there is a mistake in the test when importing PricingModelLinear. It took some time to debug. Finally we realised that lines 20 and line 21 needed swapping in `tests_public.py`.