**CS5011 – Artificial Intelligence Practice**

**Assignment 3**

**Topic 3**

**Learning**

**Report**

Logo, company name

Description automatically generated

School of Computer Science

200015215

Word Count: 1, 588

1. **Introduction**

Date of Submission 7th April 2021

The aim of this assignment is to design, build and train a neural network that is capable of predicting whether a tap is functional or needs repair.

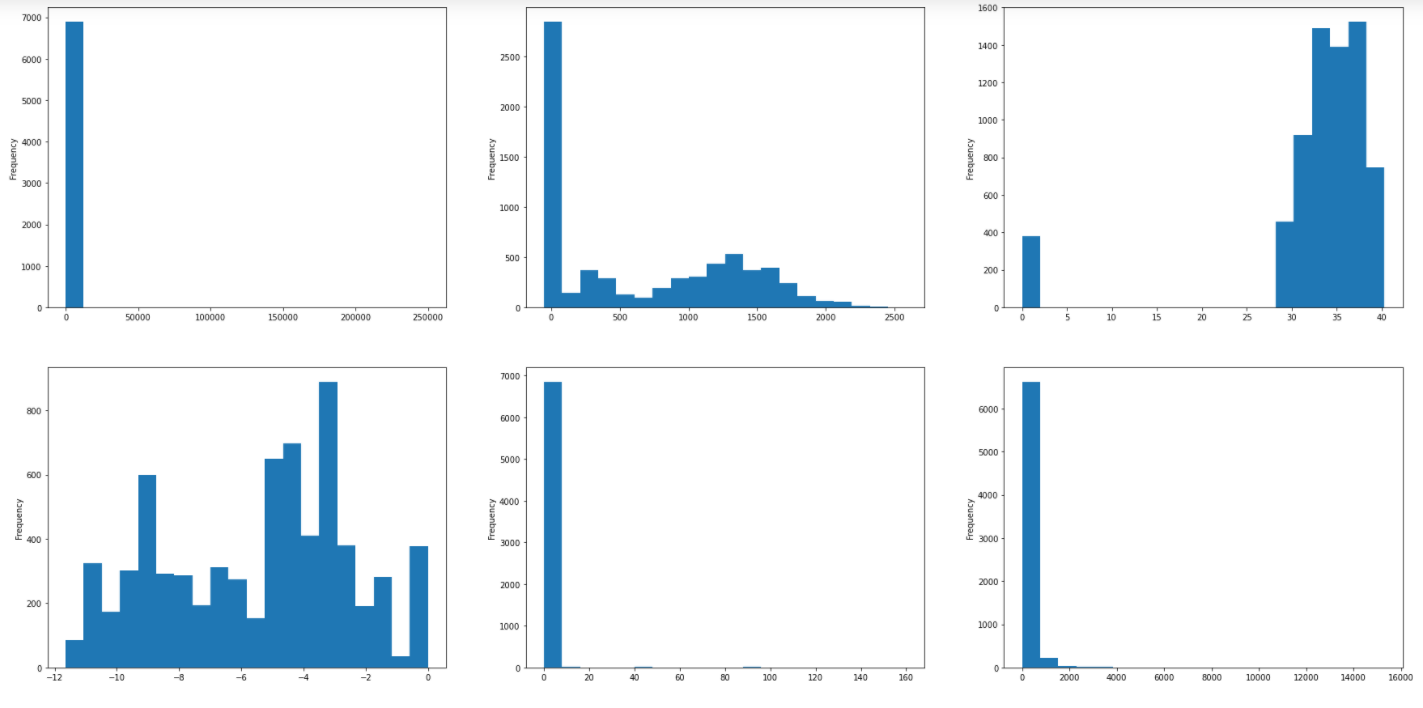
This assignment was implemented in parts according to the data that was given. The reports explains each step taken to complete the assignment

1. Part 1 – Fully working
2. Part 2 – Fully working
3. Part 3 – Fully working
4. Part 4 – Fully working
5. Part 5 – Fully working
6. Part 7 – Attempted

This assignment was written using python3. To run it, enter into to terminal and type command line arguments python A3main.py task\_number type\_of\_operation for example to train a neural network for task1 enter A3main.py task1 train task1\_train.csv task1\_NN

1. **Design, Implementation and Evaluation**
   1. **Part 1**

The task for part one was to build a neural network using only numeric features as provided by the dataset. To implement the neural network for future prediction, the first step was to properly load the dataset and prepare it for training. First, data pre-processing was done. I created a pipeline that takes in a standard scaler to scale the numerical feature so that the distribution can be normal, as some of the features did not have a normal distribution as shown in figure below.



To avoid data leakage, overfitting and other types of machine learning issues, I made sure to split the data into 70% training and 30% testing before fitting the data into the standard scaler for data scaling. Features such as num\_private and id were removed, as they are not affecting the target variable’s outcome. I then built a multi-layer preceptor classifier MLPClassifier and passed in the following parameters; number of hidden layers used is two, mini batch size of 100 and used the rectified activation function ‘relu’ to make our model non-linear. The solver used for the cost function is stochastic gradient descent ‘SGD’ to minimize the loss with a number of iteration of 1000. The learning rate used was 0.001 and early stopping was set to true to avoid overfitting. The accuracy for the training data is a low 58.6% and when the model is run on test data, the accuracy score is 58.7%.Obviously this model wouldn’t be ideal to use in our situation because the results produced is as good as guessing. Clearly, the numerical features are not well correlated with the target variable to give accurate results so that the engineers have the taps that need fixing accurately predicted. To achieve this, more features are added to the data, as task 2 will further explain. To run task 1, in python command line type **python A3main.py task1 train task1\_train.csv task1\_NN** to train the model, **python A3main.py task1 test task1\_test.csv task1\_NN** to test and evaluate the model and finally **python A3main.py task1 predict task1\_test\_nolabels.csv task1\_NN** to run predictions on a dataset without target labels.

* 1. **Part 2**

For this part, more features are added to the data and all the features are categorical. These new features introduced have typical machine learning data issues i.e. missing values, some of the features are highly correlated and can be dropped and our model does not work with categorical features. To deal with this new problem, I added a function called missing\_val that takes in a data parameter and checks which columns have missing values and replace those rows with the mode or mean. This function is added to the column transformer and passed into a pipeline like in task 1. Some columns are dropped from the dataset. This was done because they are highly correlated with another dependent variable and I instead decided to use only one. The total number of dependent features used is 28. Again, to make sure that overfitting and data leakage is avoided, pre-processing is done after the data is split into train and test of 70% and 30% respectively.

The model is implemented using the same parameters as task one just for evaluation purposes. The training accuracy improves to 69.3% and the accuracy for the test data is 75.4%, which is a lot better than the previous accuracy score. Adding more feature has clearly helped our model to perform better and give a better accuracy score as compared to the one we got for task 1. This can longer be attributed as guesswork, as 75% of the time the engineers are given a list of taps that need to be fixed, which is good. To run this task, the instructions are the same as in tasks 1 only with different task number and dataset.

* 1. **Part 3**

Part 3 of the assignment introduces data that is highly imbalanced towards one class. The target variable has 3431 rows classified as ‘functional need repair’ and 44085 rows classified as ‘others’ or in other words tap working fine or does not work at all. The model in such cases is highly skewed towards the class with more data. To implement this task, the model is trained using the previous program with the exact same steps of data pre-processing.

Looking at the accuracy score using this model, the training gives accuracy of 92.22% and the test data a whopping 94.3% accuracy. Almost a perfect accuracy score, however our data is highly imbalanced toward the others class and if one checks further, the predicted results will usually be ‘others’. This is bad for our core tasks of predicting taps that need fixing. To further analyse our results, we need to use a different type of evaluation metric called a classification report. This report gives the scores for Recall and Precision where recall = TP / (TP + FN) and precision = TP/ (TP + FP) where TP is true positive, FN is false negative and FP is false positive. These scores need to be closer to 1 for our prediction to be classified as accurate. For this trained neural network, the recall for ‘functional needs repair’ is a low 20% and for the class ‘others’ it’s over 89%. Our goal is to have the recall for the class ‘functional needs repair’ to be very high, that is when we can say our model is working very well. This issue is solved in part 4 of this assignment.

* 1. **Part 4**

For this part of the assignment, I implemented two solutions to the imbalanced problem. The first approach I used is random under sampling and the second approach is random over sampling. To implement random under sampling, the goal is to have our target features to have the same number of data as minority class ‘functional needs repair’. This gives a dataset with the 3431 for both ‘functional needs repair’ and ‘others’. Training the model using this approach gives recall for functional need repair class of 81% and precision of 72%. This clearly is an improvement on our previous model as both classes are well represented in the training data. To implement random over sampling, the same approach as random under sampling is used, but only now the target feature have to have the same number as the majority class, which in our case is 44085 for both features. Training this neural network takes some time depending on the resources available on your computer because the total number of rows is increased. The training accuracy score after over sampling the data is 88.8% but the recall for ‘functional needs repair’ is a poor 36%. Under sampling technique produces the best result for our model. To run this task, other parameters have been included to show different techniques for handling imbalanced data. For under sampling technique, **python A3main.py task4 train undersample task3\_train.csv** **task4\_NN\_Under** is the command used to run the training and over sampling technique, we just change the undersample to oversample and give a different model name.

* 1. **Part 5**

For part 5, I implemented the methods used in the previous part just to see how well they would work on a multi-class problem. In comparison to task 4, the recall and precision scores reduce when under sampling technique is used. The model parameter solver is changed to softmax as it works better with multi-class problems.

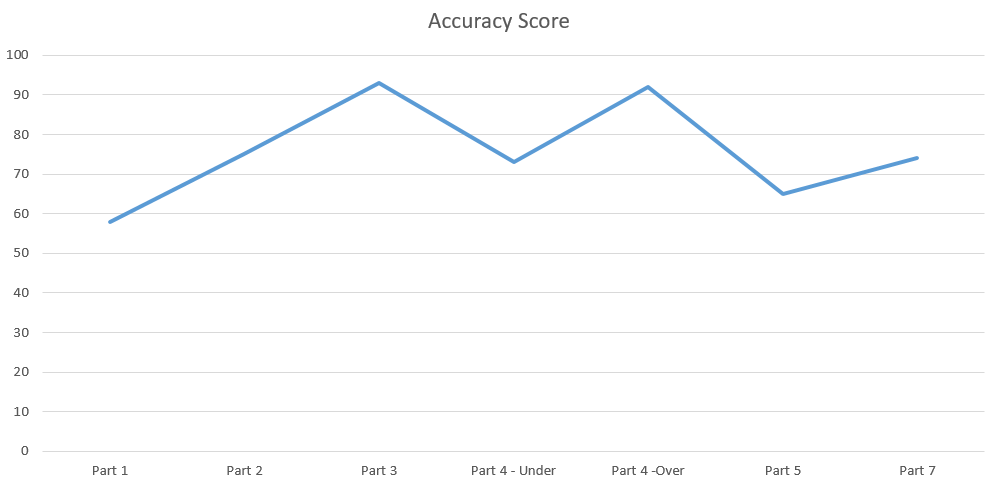
Another technique that we can use to improve the recall for the class ‘functional needs repair’ is a method called synthetic minority oversampling technique SMOTE. Python provides this package and works well in reducing model overfitting. It generates samples in our rare class ‘functional needs repair’ and constructs new synthetic samples which are created by either skew or rotation on the feature space. SMOTE uses nearest neighbours’ technique to solve our problem of class imbalance. For this task SMOTE was highly considered and an implementation was done.

To increase the score for ‘functional needs repair’, cost sensitive learning is applied where the class weight for ‘functional needs repair’ is given more weight compared to the other classes.

* 1. **Part 7**

In part 7 of the assignment, I followed the same principles as the ones set in task 5 but on a decision tree classifier. My approach to making the classifier penalize errors made for ‘functional need repair’ was to have the parameter class\_weight = ‘balanced’ so that all the classes are equally represented in the data training. Another strategy that one can use u=is grid search. Using this methodology, we assign different weights to all the classes and run a search to see which result gives the best false negative result. The goal of penalizing the error for wrong classification is to have a low false negative rate.

**Evaluation.**

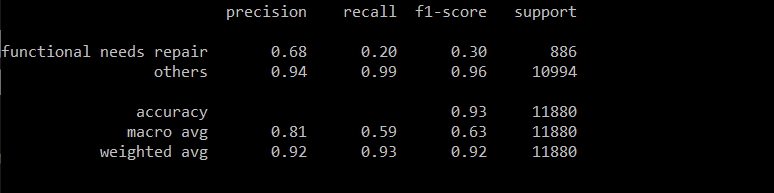


**Testing.**

Part 2

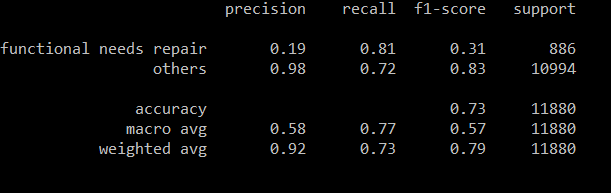


Part 3

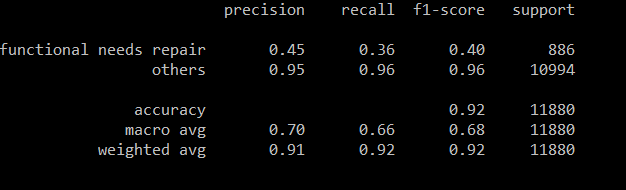


Part 4

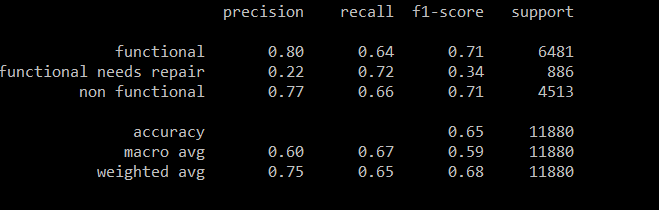
Under sampling technique



Over sampling technique



Part 5 Neural Network with cost-sensitivity



Part 7 Decision tree with cost-sensitivity

