Abstract

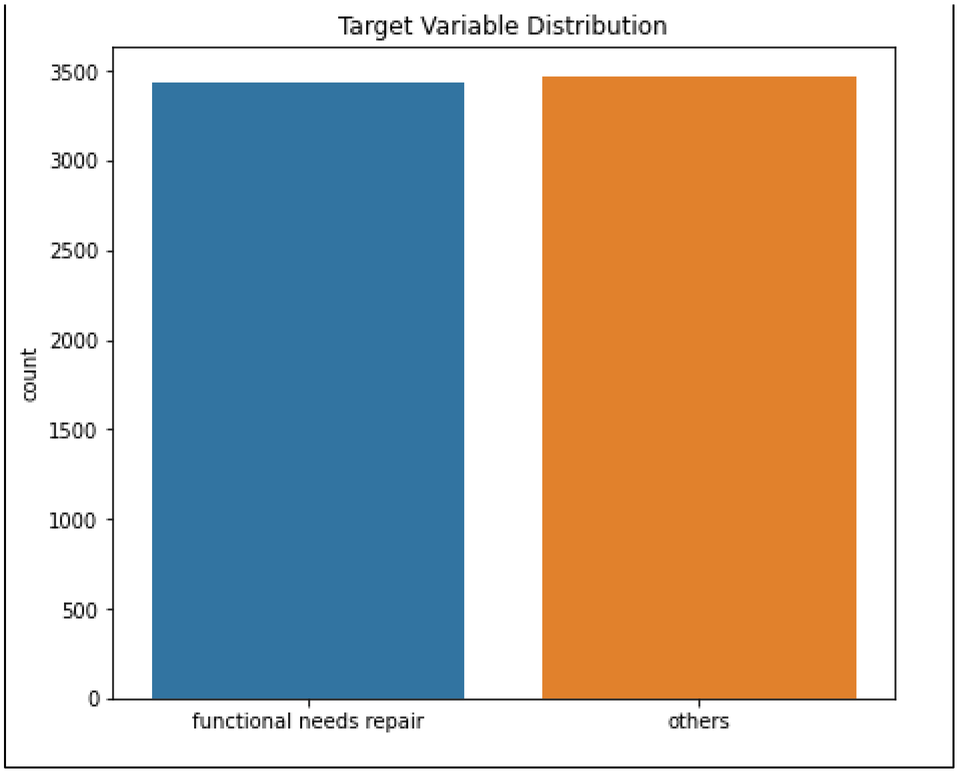
Leveraging over 8,000 pump datasets gathered from, this paper demonstrates how deep learning is employed to predict which pumps are not functional and therefore need repair. This paper follows the machine learning development life cycle to achieve the goal. The steps taken include identification of data, exploration data analysis, feature engineering, model design and parameter selection, model training and model testing.

Task1: Building a Neural network prediction model using numerical data

This task entailed developing a neural network model that leverage only numerical data. In this context numerical data refers to features that are measurable. Hence features used for this task include ['id', 'amount_tsh', 'gps_height', 'longitude', 'latitude', 'num_private', 'population']. Except the 'id' which is used to index our data points, the rest of the features were used to model the data.

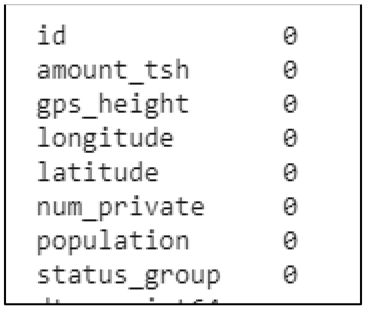
Target

The target feature was status_group which include two set of classes as shown in the caption below



Task 1 Data

We are provided with three sets of data namely: training, test and test without label datasets. The train dataset contains data used for training our dataset. The train data is labeled. This data has a shape of (6906, 8) which implies 6906 data points with 8 features. The data contains no missing values as indicated below.



```
id              0
amount_tsh      0
gps_height      0
longitude       0
latitude        0
num_private     0
population      0
status_group    0
```

The test data include a section of the data to be used to validate the model. The data comes in a shape of (1726, 7), implying 1726 data points and 7 features. The data is not label and does not have missing values as well.

Data Preprocessing

The two-dataset train and test data are concatenated in order to be preprocessed uniformly. After concatenation we obtain a combined data of shape (8632, 8) implying about 8700 data points. In the combined data both id and status_group fields are dropped in order to remain with main features for the model building. The header is striped off in order to remain with the numerical data. The data obtained is converted to float type in order to have uniform format. Using LabelEncoder( ) library from Sklearn library.

Modelling

The model follows Sequential architecture with Dense of input_dim = 60 and Relu as the activation function. Furthermore, another 1 layer is used with sigmoid as the activation function. Binary_crossentropy is used as the loss function with Adam as the optimizer and accuracy as the performance metrics.

The estimator which is a KerasClassifier obtained from wrapper.sklearn library uses 100 epochs divided in 5 batches for modelling. In order to validate the model StratifiedKFold is used with N_splits of 10. This means that the train data is divided into bits of 10 so as to validate the model. The model's final outcome is the percentage of the accuracy.

Task 2 Model

In the second attempt of modelling, both numerical and categorical data are used. With the inclusion of the categorical data the preprocessing becomes more complex. With the increase the model learning is guaranteed to improve but to a particular point where an increase in features does not count any more. A quick look into the data shape we find our new data in the shape of (6907, 41). This implies that the data has 6906 data points with 41 features. This features include 'id', 'amount_tsh', 'date_recorded', 'funder', 'gps_height', 'installer', 'longitude', 'latitude', 'wpt_name', 'num_private', 'basin', 'subvillage', 'region', 'region_code', 'district_code', 'lga', 'ward', 'population', 'public_meeting', 'recorded_by', 'scheme_management', 'scheme_name', 'permit', 'construction_year', 'extraction_type', 'extraction_type_group', 'extraction_type_class', 'management', 'management_group', 'payment', 'payment_type', 'water_quality', 'quality_group', 'quantity', 'quantity_group’, 'source', 'source_type', 'source_class', 'waterpoint_type', 'waterpoint_type_group', 'status_group'. Additionally, the new dataset presents itself with a couple of missing values as follows.

```
id                    0
amount_tsh            0
date_recorded         0
funder              567
gps_height            0
installer           570
longitude             0
latitude              0
wpt_name              0
num_private           0
basin                 0
subvillage           27
region                0
```

```
region_code            0
district_code          0
lga                    0
ward                   0
population             0
public_meeting       307
recorded_by            0
scheme_management    404
scheme_name         3200
permit               405
construction_year      0
extraction_type        0
extraction_type_group  0
extraction_type_class  0
management             0
management_group       0
payment                0
payment_type           0
water_quality          0
quality_group          0
quantity               0
quantity_group         0
source                 0
```

```
source_type              0
source_class             0
waterpoint_type          0
waterpoint_type_group    0
status_group             0
```

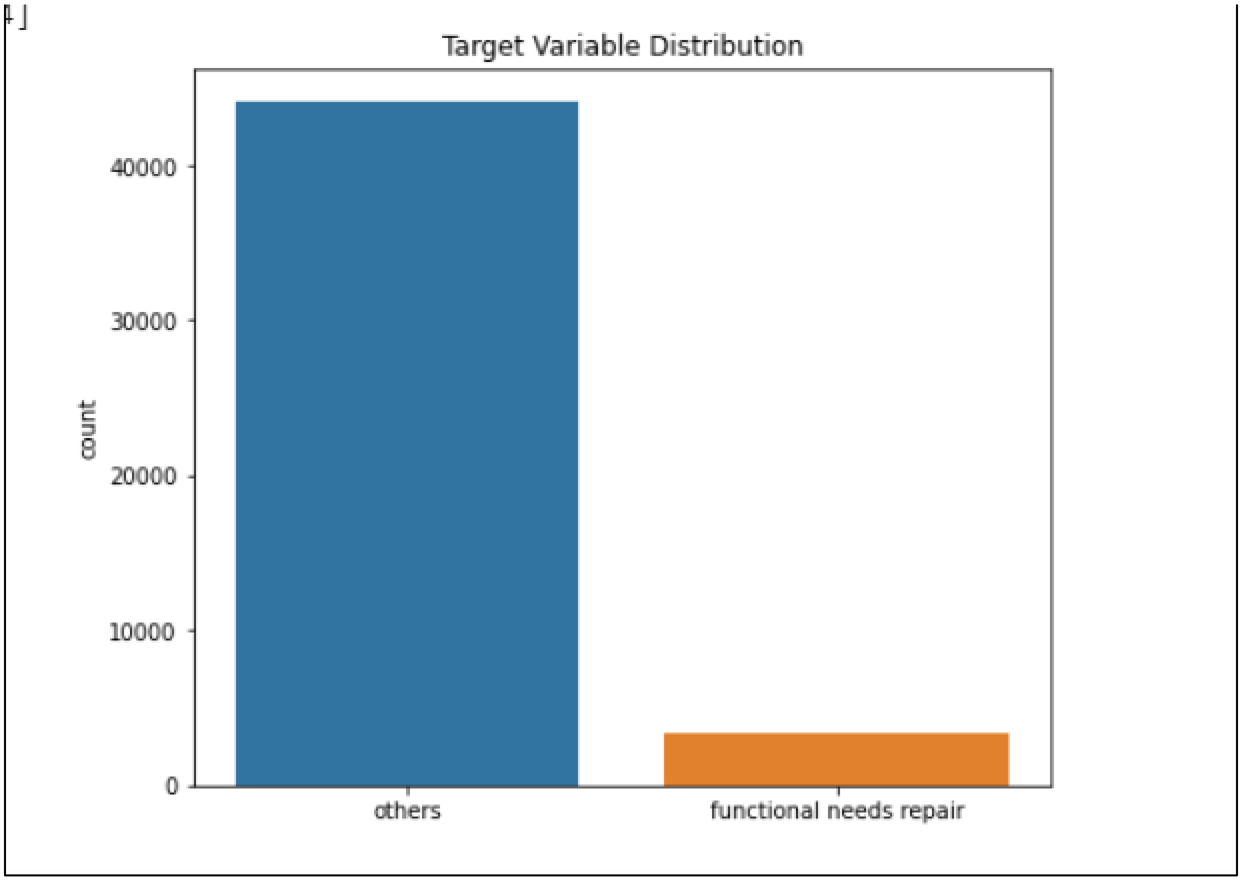
Data Preprocessing

Similar to the previous task, both train and test data are concatenated in order to come up with combined data for uniform formatting. Columns like scheme_name have a higher number of missing values hence it is better to drop it since imputing values for this column will lead to altering the data too much. Also it was important to identify columns that were not of great value to the final outcome and drop them. Columns dropped included the target column as well as the id. These columns are "id", "test", "train", "status_group", "num_private", "scheme_name", 'waterpoint_type_group', 'quality_group', 'payment_type', 'extraction_type_group', 'extraction_type_class', 'management_group', 'source_type', 'source_class'.

The data is now ready. It is subjected to normalization where StandardScaler( ) is used to normalize the data. The normalization is only applicable to numerical data. For categorical data, the data is encoded using the LabelEncoder library. Furthermore, we combine highly related features such as installer, funder, subvillage and scheme_name. These features were dropped also due to the fact that they have a large number of unique values. The remaining categorical data was labelled using LabelEncoder( ) library. The library assigns different numerical values to unique features. After the data was prepared, we tested again with the baseline model to compare the result with the original model. The accuracy improved as much as the baseline hyperparameters were not maximally tuned.

Task 3: Dealing with imbalance data

This task is represented with the main objective being how to deal with imbalanced data. Imbalanced data means a dataset that has one type of class of outcome relatively more than the other. Dealing with the dataset of task 3 we are faced with the challenge of the huge difference between "others" type of pumps for our target (over 40, 000) compared to "functional need repair" which is below 10,000 as captured in bar graph below.

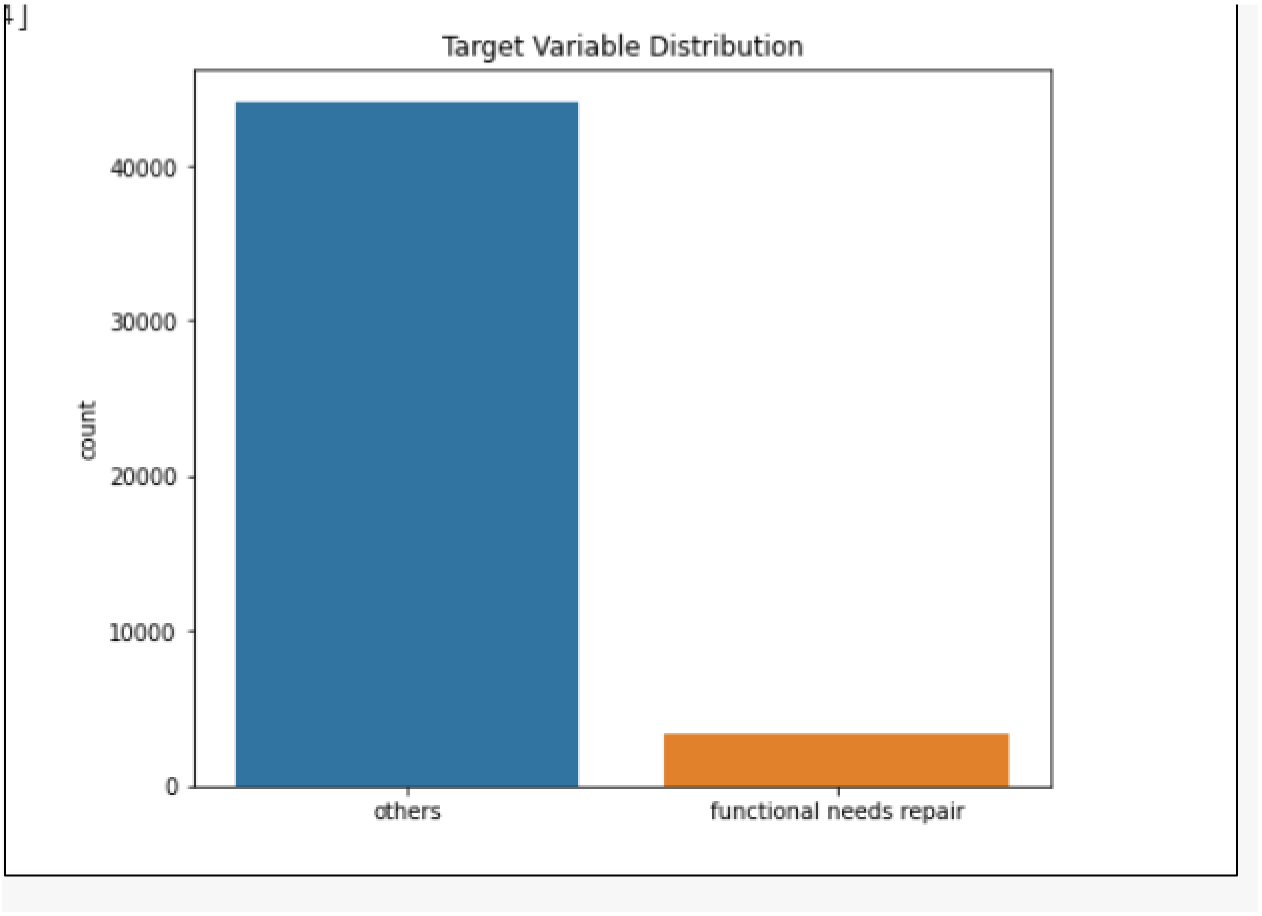


This indicated functional needs repair was under represented (around 7.8%). Therefore, an under sampling was done with varying ratios of 1:2 up to 1:10 with the later representing "others". This approach never yielded a better result than the original approach.

Task 4: Dealing with imbalanced Binary Classification

To eliminate the problem of imbalanced binary classification a quick check on features that are closely related was the next resolution. The features included extraction_type, extraction_type_class and extraction_type_group in which only the finer grain was retained. Additionally, features which tend to have high arity were weighed on a scale of 10 while categorizing the rest as others.

From this approach a data with 20 features was obtained and yielded a better accuracy. Digging deeper into the details of classification, it was realized that for the "functional needs repair" recorded less True Positive rate.

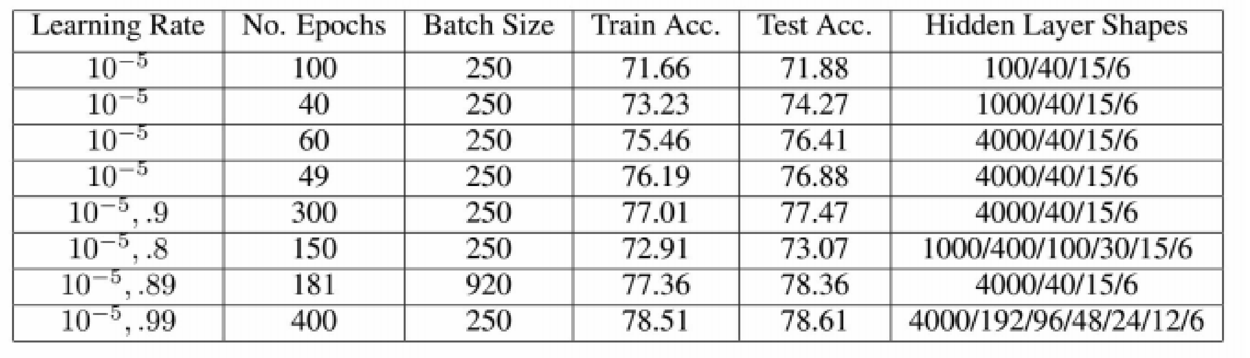


Task 6: Improving performance using hyperparameter tuning

Trying out various architecture this project adopted a fully connected network shaped (432, 4000, 192, 96, 48, 24, 12, 6, 3) with a softmax output layer (3 neurons).

After training to 550 epochs accompanied by a decaying learning rate approximated at 10^{-5} the decay was 0.99 using step size of 2000. The decaying rate was fundamental to proceeding iterations in comparison to the baseline model of batch size 100 which caused a lot of noise while converging with a constant learning rate.

As much as other learning rate returned an analogous improvement to the learning, this project finds 250 as the most suitable batch size which increased the training speed.



Learning Rate	No. Epochs	Batch Size	Train Acc.	Test Acc.	Hidden Layer Shapes
10^{-5}	100	250	71.66	71.88	100/40/15/6
10^{-5}	40	250	73.23	74.27	1000/40/15/6
10^{-5}	60	250	75.46	76.41	4000/40/15/6
10^{-5}	49	250	76.19	76.88	4000/40/15/6
10^{-5}, .9	300	250	77.01	77.47	4000/40/15/6
10^{-5}, .8	150	250	72.91	73.07	1000/400/100/30/15/6
10^{-5}, .89	181	920	77.36	78.36	4000/40/15/6
10^{-5}, .99	400	250	78.51	78.61	4000/192/96/48/24/12/6

From the result we could conclude that the model did not overfit and performed excellently on the test set.