VRIJE
UNIVERSITEIT
BRUSSEL

STATISTICAL FOUNDATION OF MACHINE LEARNING

Project 2021

Luckas Declerck 0580862
Mengyao Song 0563486

May 18, 2021

Sciences and Bio-Engineering Sciences

1 Introduction

Predicting which pumps are faulty is a practical and challenging project. On one hand, if the performance of pumps can be predicted by the collected data, the manpower and material resources can greatly economize. On the other hand, it is due to the abundance, redundancy, and uncertainty of data, the prediction will be extremely complicated. Once the prediction is implemented, it would be much easier to improve maintenance operations and ensures the quality of water according to the information obtained through regular monitoring. In order to properly mine the valuable information contained in the dataset, this project contains data preprocessing and several procedures used for model assessment and selection.

2 Data Preprocessing

One way to improve the performance of a machine learning is data preprocessing. Processing the datasets properly allows us to remove unnecessary data label, avoid missing value and make the data more suitable for the specific classifiers.

In our project, analyzing the dataset is achieved by using *summary*(train) and *sapply*(train,class). As required by the limitation of the random forest classifier, data cleaning need to list all columns which have more than 53 categories. The filter code is as follows:

```
for i in len(column):{
    if(length(unique(train[,i])) > 53):
    print(colnames(train[i]))}
```

The outcomes are: "id", "amount_tsh", "date_recorded", "funder", "gps_height", "installer", "longitude", "latitude", "wpt_name", "num_private", "subvillage", "lga", "ward", "population", "scheme_name", "construction_year". Through analysis, it was found that one maps to multiple region_code, but one code only maps to one region, e.g. subvillage: Arusha, region_code: 2, 24. region_code:2, subvillage: Arusha. With region_code and district_code, the values of basin, region, lga, ward can be determined. Those proxies for the regions and two labels of *longitude* and *latitude* have a very limited effect on the outcomes. As for *population*, *num_private* and *shceme_name*, the meaning of these attributes is unclear.

But it is not reasonable to remove all of these columns because it would drop too much information. Two of the columns *funder* and *installer* are kept by reducing their factor levels rather than remove them directly for these labels may affect the training results. The process of reduction is shown below. The other columns except *id* are dropped due to the fact that they have too much levels and are not as important as previous two labels. The label *recorded_by* is also removed here for it is a constant that does not make a difference to the training model.

```
## SET THE LOW FREQUENT VALUES INTO "OTHER"
col <- as.character(col)
col="" <-NA
train <- na.omit(train)
colVal <- the most frequent 30 colVal
Val not in colVal <- "Other"
```

Also, in the remaining columns, some of them have missing values which are recorded as*""*, like *scheme_management*, *permit*, *public_meeting*. The others like *source_class* and *management_group* have *"unknown"* values. In this step, all these values are set into *"NA"* and then omitted. Since some of them are factor and it is difficult to deal with a factor directly, the first thing i did here is convert them into character and then back again after the omitting is done.

```
## REMOVE EMPTY AND "UNKNOWN" VALUES
col <- as.character(col)
col = "" or col = "unknown" <-NA
col <- as.factor(col)
train <- na.omit(train)
```

The next cleaning process is to deal with categorical data in *water_quality*, *quality_group*, *quantity*, *quantity_group*, *payment* and *payment_type*. Since there are also *"unknown"* values in these columns and I intend to keep these data by replacing them with mean value rather than removing them from the dataset, the following process is adopted:

```
###
#  Deal with categorical data
#  Replace "unknown" values with the mean value
###
col(level = "unknown") <- 0
levels <- numeric labels
col = "0" <- NA
col = "NA" <- mean(col, na.rm = TRUE)
```

For the normalization, *amount_tsh* contains abundant numeric values. The definition of *amount_tsh* is the total static head and that it means the amount of water available to a waterpoint. However, there are many *amount_tsh* recorded as "0" while the pumps are still functional. I decided to replace the "0" value with the mean value firstly. By plotting the boxplot, I noticed there are several outliers, the function rm.outlier(data1[,"amount_tsh"], fill = TRUE, median = TRUE, opposite = FALSE) is adopted here to remove the outliers so that the training performance can be enhanced.

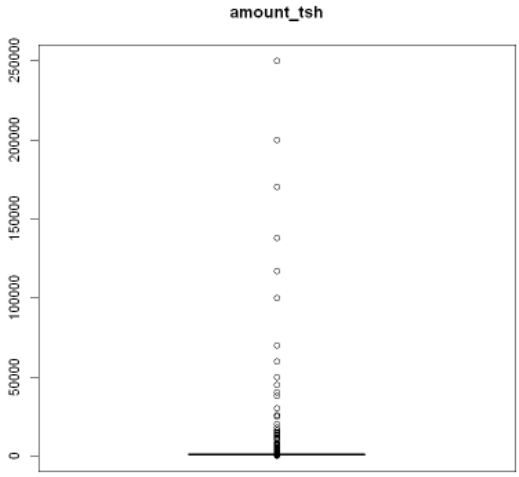

Figure 1: The boxplot of the *amount_tsh* columns.

After data preprocessing, we get a new dataset with 48020 rows and 28 columns which has no missing values nor the NA values. The new dataset ensures the success of the following training process.

3 Model Selection

In this chapter we're going to assess which models are suited for the correct prediction of *status_group* labels. We have already preprocessed the data and made a selection of features which we can use.

The final choice of our model will be decided based on the estimated out-of-sample error. This will in our case computed using **k-fold cross-validation**. Most of the used algorithms have built-in support for cross validation, so this will be explained here as well as in the notebook itself.

3.1 rpart trees

The first model that we're going to try is a using **recursive partitioning**. This algorithm (the ***rpart*** library in R) creates a decision tree which strives to correctly link objects to the correct class. This is done by first creating the fully grown tree and then pruning it to the smallest possible tree with lowest error value. Internally, the algorithm uses k-fold cross validation and fits each subtree on a fold. This allows to visualise the relative cross-validation error after the model has been trained.

```
# Temporary only select numeric features
model2_data = data

# We don't need to define the parameters for the Cross-Validation,
# the rpart algorithm uses 10-fold CV internally.

# Create the tree model. Since rpart is not really efficint, we're going to use
# a subset of features that yielded the best results while testing this.
model2 <- rpart(status_group~ amount_tsh + region_code + district_code + extraction_type +
                extraction_type_class + source_type + quantity + quantity_group +
                 waterpoint_type + payment + payment_type + source_class,
                data = model2_data, method = 'class', control=rpart.control(minsplit=2,
                    minbucket=1, cp=0.005))
```

The rpart algorithm was really struggling to create a good fit in time. Even with only a few features selected, it got a root node error of 0.448. and this was after CV as well (see image)

3.2 nearest neighbours

The next model we're going to try is created using KNN. Each entry will be classified based on the euclidian distances using the selected features and their dimensions. A requirement of course is that the distance needs to be calculated, and needs to make sense. This might be a problem for categorical features, but we already took care of most of them in the preprocessing phase.

To validate our model, we still need to add some things since the knn algorithm does not provide this by itself. There is however a function **knn.cv** in the **class** packages which uses k-fold cross validation, and we will manually create the confusion table afterwards.

We're going to select the features converted to numerical ones and create the model:

```
install.packages("class")
library(class)

# Temporary only select numeric features
knndata = data %>% select_if(sapply(., is.numeric))
knndata = knndata[ , -which(names(data) %in% c("id"))]

# We need of course to evaluate our model using cross-validation.
# The 'class' package gives us a knn.cv method which will perform this
# using leave-one-out cross validation. Variable k defines the number of neighbours considered
# and the amount of folds in the CV process.

# Create model
model3 <- knn.cv(knndata, data$status_group, k = 10, prob=TRUE)

# To verify our results, we will create a confusion matrix to calculate the error:
confusion_matrix <- table(model3, data$status_group)
confusion_matrix

accuracy <- sum(diag(confusion_matrix)) / sum(confusion_matrix)
```

```
21 accuracy
```

This yields the following confusion matrix and gave an accuracy of 0.65. Even after some tuning, we knew that we could do better.

Confusion Matrix

model3	functional	functional needs repair	non functional
functional	4293	489	1564
functional needs repair	103	102	39
non functional	1056	200	2154

3.3 random forest

The last model that we're going to try is a **random forest** classifier. A few reasons why we're selecting this are:

- the dataset contains a lot of categorical data, and RF can handle this (at least with an upper cap)
- random forests are known to provide a high accuracy, even with a large amount of features
- compared to other *decision tree based* algorithms, they tend to limit overfitting without increasing error

There is no need for CV to get an estimate for the dataset error, since it is estimated internally during the runtime of the algorithm. After the run, we receive an OOB estimate that we will use to compare the model with other models.

```
1  install.packages("randomForest")
2  library(randomForest)
3
4  # First make sure that the 'id' column is not in the training dataset
5  # (but we still need it to link it to the test values)
6  rfdata = data[ , -which(names(data) %in% c("id"))]
7
8  # Create the RandomForest model. We're going to use 400 trees since this yielded the best
       results.
9  model1 <- randomForest(status_group ~ ., data = rfdata, ntrees = 400, importance = TRUE)
10
11 # Show OOB estimate of error rate of model
12 print(model1)
13
14 # Show the importance of each feature after first run
15 varImpPlot(model1)
```

The first thing we note is that the accuracy is a lot better than the previous model. We included all our selected features from the preprocessing step and got an accuracy of 0.775. If we take a look at the confusion matrix and the results of the model, we see that our model has trouble correctly classifying *functional needs repair* pumps.

Confusion Matrix

	functional	functional needs repair	non functional
functional	4856	131	465
functional needs repair	458	192	141
non functional	971	84	2702

3.4 XGBoost

Apart from the learning model mentioned in the previous part, we select XGBoost to implement a learning procedure.

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the Gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

It takes several types of input data:

- Dense Matrix: R's dense matrix, i.e. matrix ;
- Sparse Matrix: R's sparse matrix, i.e. Matrix::dgCMatrix ;
- Data File: local data files ;
- xgb.DMatrix: its own class (recommended).

In the project, xgb.Dmatrix is used as input type. First convert all data to numeric data because data stored as factors/ characters won't be able to convert to a matrix. Then the input frame is converted to matrix first using as.matrix() and then pass to xgb.Dmatrix().

```
xgb.tab = xgb.cv(data = train.DMatrix, objective = "multi:softmax", booster = "gbtree",
                 nrounds = 500, nfold = 4, early_stopping_rounds = 10, num_class = 4, maximize
                      = FALSE,
                 evaluation = "merror", eta = .2, max_depth = 12, colsample_bytree = .4)
```

The parameter *multi:softmax* stands for set XGBoost to do multiclass classification using the softmax objective, it also need to set *num_class*(number of classes). Here, the *num_class* is set to the number of the label categories *length(unique(label))*. *nrounds* stands for the max number of iterations. The dataset is randomly partitioned into *nfold* equal size subsamples. *early_stopping_rounds* is set to an integer k, means that training with a validation set will stop if the performance doesn't improve for k rounds. *booster* suggests which booster to use.

XGBoost has several features to help view the learning progress internally. The purpose is to help to set the best parameters, which is the key of your model quality. One of the simplest way to see the training progress is to set the verbose option. Here the parameter *verbose* is set to 1, indicating to print evaluation metric.

```
model4 <- xgboost(data = train.DMatrix, objective = "multi:softmax", booster = "gbtree",
                  eval_metric = "merror", nrounds = 33,
                  num_class = 4,eta = .2, max_depth = 14, colsample_bytree = .4, verbose = 1)
```

It is important to check if there are highly correlated features in the dataset. There are several types of importance in the Xgboost - it can be computed in several different ways. In the projcet the gain type is selected to compute the importance. The gain type shows the average gain across all splits where feature was used. The output of the importance is shown as figure.2:

It could be found that the features date_recorded and quantity_group have a high importance value for our model. However, if analyze it from a practical point of view, these two features should not be a decidable values for a prediction. This leads to a low score in the prediction.

4 Conclusion

We remarked the best results using the Random Forest classifier, since it had the lowest OOB of the error rate after the CV. We could use all features that were selected in the first phase and

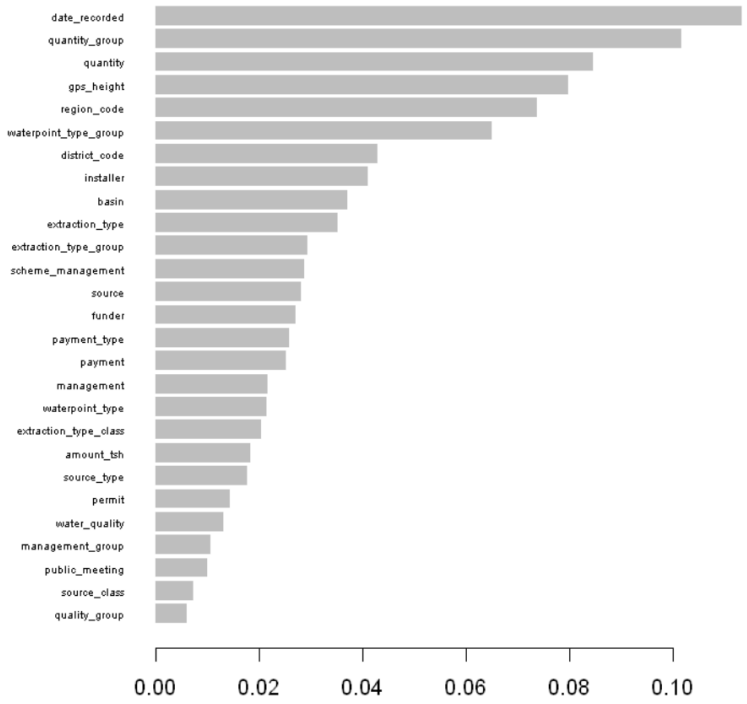

Figure 2: The importance of each feature

eventually scored 0.78 accuracy. After that, we predicted the labels of the Pump It Up test set and uploaded it to the DrivenData portal.

Our DrivenData teamname is **SFML Team** consisting of two members Luckas Declerck (0580862) end Mengyao Song (mengyaosong). We got a high score of 0.7446 and had a rank of 3498 on 17/05/2021. Screenshots are added in the ZIP.

We implemented this project with only two students, since Mark has not worked on this project after registering. We divided the work in one exercise per student, so we did't have much time for exercise three.

The link of the video is:

https://drive.google.com/file/d/1W8HNAJ3kI9dD5DKMgYL_lcjl6e7h605j/view?usp=sharing [1]

[1] https://drive.google.com/file/d/1W8HNAJ3kI9dD5DKMgYL$_l$cjl6e7h605j/view?usp = sharing