ECOLE
POLYTECHNIQUE
DE BRUXELLES

INFO-F422 | Statistical foundations of machine learning

DrivenData Challenge Report
Pump it Up: Data Mining the Water Table

Darius Couchard
Stefano Donne
Paul Parent

11th May 2021

1 Introduction

The following report will describe our methodology and results for the DrivenData Challenge "Pump it Up: Data Mining the Water Table". The objective for the participating teams is to predict the functionality of water pumps in Tanzania using machine learning techniques based on a set of 40 features. In this report, we will first show how we processed the data (missing value imputation, normalization, feature selection and engineering). Then, we will show the results of three algorithms we chose from the assignment list (random forest, SVM and neural network) and a model not studied in the lessons (Gradient Boosting Tree). Based on these results, we implemented a learning procedure. Finally, we submitted the predictions to the DrivenData challenge to compare our results with other teams.
A video summarizing our work is available here on **youtube**.

2 Data analysis & Data pre-processing

The data provided by DrivenData is divided in three files: the "training set values" which is composed of 40 columns (features) and 59800 rows (pumps), the corresponding "training set labels" which is composed of 2 columns (ID of the pump and functionality) and the "test set values" (dataset to test our machine learning pipeline on). The following table (cf. Table 1) shows details about each feature of the training set.

Table 1: Table describing each feature according to the missing values (NA or "0"), selection of the feature (i.e., drop or keep) and engineering (e.g., standardization, one-hot encoding).

Feature	**Missing values**	**Selection**	**Engineering**
id: Unique ID of the pump	None	Drop (one different for each pump)	None
amount_tsh: Amount of water available	41639 "0"	Drop (too much NA)	None
date_recorded: Date the row was entered	None	Drop (not relevant for functionality)	None

funder: Who funded the pump	3635 NA and 777 "0" categorized as "others"	Keep	Two categories of funders: "big" (10% of biggest funders) and "small" (the rest) and one-hot encoded
installer: Who installed the pump	3635 NA and 777 "0" categorized as "others"	Drop (correlated with funder feature)	None
gps_height: Altitude of the pump	21960 "0"	Drop (too much NA)	None
wpt_name: Name of the waterpoint	None	Drop (not relevant for functionality)	None
num_private: No information on this feature	58643 "0"	Drop (too much NA)	None
longitude: GPS coordinate	None	Drop	None
latitude: GPS coordinate	None	Drop	None
subvillage: Geographical location	None	Drop	None
region: Geographical location	None	Drop	None
distric_code: Geographical location	None	Drop	None
lga: Geographical location	None	Drop	None
ward: Geographical location	None	Drop	None
region_code: Geographical location	None	Keep (only one geographical feature)	One-hot encoding
basin: Geographic water basin	None	Keep	One-hot encoding

population: Population around the pump	22900 "0"	Keep	Fill "0" with the mean of the region's population and standardized
public_meeting: If public meeting happened	None	Drop (not relevant for functionality)	None
recorded_by: Who entered data in the row	None	Drop (same value for each pump)	None
scheme_management: Operator of the pump	3877 NA	Drop (uncorrelated)	None
scheme_name: Operator of the pump	28166 NA and 2697 levels	Drop	None
permit: If the pump is permitted	3056 NA (replaced by mean of feature)	Keep	Standardization
construction_year: Year of construction	20709 "0" replaced by mean of the pump's age	Keep	Year changed to age of the pump and standardized
extraction_type: Kind of extraction the waterpoint uses	None	Drop	None
extraction_type_group: Kind of extraction the waterpoint uses	None	Drop	None
extraction_type_class: Kind of extraction the waterpoint uses	None	Keep (only one extraction feature)	One-hot encoding
management: How the waterpoint is managed	None	Drop	None
management_group: How the waterpoint is managed	None	Keep (only one management feature)	One-hot encoding
payment: What the water costs	None	Drop	None

payment_type: What the water costs	None	Keep (only one payment feature)	One-hot encoding
water_quality: The quality of the water	None	Drop	None
quality_group: The quality of the water	1876 NA replaced by mean of the column	Keep (only one quality feature)	6 categories (1 to 6) and standardization
quantity_group: The quantity of the water	None	Drop	None
quantity: The quantity of the water	789 NA replaced by mean of the column	Keep (only one quantity feature)	6 categories (1 to 6) and standardization
source: Source of the water	None	Drop	None
source_class: Source of the water	None	Drop	None
source_type: Source of the water	None	Keep (only one source feature)	One-hot encoding
waterpoint_type: Kind of waterpoint	None	Drop	None
waterpoint_type_group: Kind of waterpoint	None	Keep (only one waterpoint type feature)	One-hot encoding

The "Missing value" column shows that some features contain "NA" (Not Available or missing values) and also "0" which can be considered as missing values as well. First example, in "amount_tsh", 41639 out of 59400 pumps have "0". Still, some of these pumps are functional: how can pumps be considered functional (during the record) if no water is available? Indeed, we considered that we could not differentiate and that there were too much missing values: we decided to drop the feature. Second example, the funder feature has 3635 NA and 777 "0", both are considered as missing value and are categorized as "others". There are other tools to replace missing values, such as taking the mean of the column (e.g. construction_year).

The "Selection" column shows our choice of keeping or dropping each variable. Features are dropped for different reasons:

- Numerous missing values (e.g. amount_tsh, gps_height)
- Correlation with other features (e.g. installer)
- Uncorrelated with functionality of the pump (e.g. wpt_name, public_meeting, scheme_management)
- Features contain the same kind of information (e.g. geographical features, source feature).

The "Engineering" column shows the necessary modifications that we brought to the feature before processing them with the classification algorithms. There are three types of modifications:
1) A feature consists in continuous numerical values, we applied standardization in order to obtain a new column with a mean value of 0 and a standard deviation of 1 (e.g. construction_year).
2) A feature consists of ordinal categorical values (hierarchy between categories), we mapped each string to a numerical value (e.g. quality_group). We also apply standardization on those values.
3) A feature consists of nominal categorical values, we applied one-hot encoding which creates a new column (with binary values) for each category (e.g. source_type).

3 Models implementation

3.1 Random Forest

Random Forest is a powerful algorithm that arises from decision tree techniques and is fit for classification tasks. As the name 'Forest' suggest, it is made from an ensemble of N decision trees. Each tree is built upon d randomly selected features and bootstrapped samples from the training set.

For every input given to the model, each tree will output a class prediction. The class that has the majority upon the N trees is taken as the output of the Random Forest model.

The model has been implemented in R with the *randomForest* library. Tweaking on the N and d values allowed to slighty increase its performances. To assess its performances the metrics used are the Out-Of-Bag (OOB) error and the accuracy on validation set. The class *functional needs repair* has shown to be the worst in term of accuracy. So the prediction error value on this class is also used to assess the model tweaking.

The Table below summarize the results obtained for different tweaks, with the funder variable dropped.

Table 2: Tweaking of the Random Forest Model (funder variable dropped)

N and d values	OOB	Accuracy	Pred. Error on F.N.R. class
100 Trees / 8 Features	22.36%	76.8%	90%
250 Trees / 8 Features	22.26%	77%	90%
100 Trees / 10 Features	21.84%	76.9%	87%
150 Trees / 12 Features	21.47%	77.1%	83%
120 Trees / 20 Features	21.21%	77.4%	77%
200 Trees / 25 Features	21.13%	77.8%	75%
500 Trees / 25 Features	21.20%	77.7%	76%
250 Trees / 28 Features	21.12%	77.9%	75%

The last entry of the Table above is the best tweak found. More trees or features doesn't increase the performances. It has been decided to run again some benchmarks but with the funder variable categorized and one-hot-encoded. The results are displayed in the Table below.

Table 3: Tweaking of the Random Forest Model (with funder variable)

N and d values	OOB	Accuracy	Pred. Error on F.N.R. class
600 Trees / 18 Features	20.94%	78.3%	76%
450 Trees / 18 Features	20.88%	**78.4%**	77%
350 Trees / 20 Features	21%	78.3%	76%
350 Trees / 23 Features	21.07%	78.3%	75.2%
350 Trees / 32 Features	21.35%	78.2%	73.5%

As the bagging is a random operation, the OOB can be slightly different from one run to an other even though the same parameters are used. The tweak chosen is **450** trees and **18** randomly selected features per tree. With these tweaks, the model shows the best Accuracy (**78.4%**) on validation set.

3.2 Support Vector Machines

Support Vector Machines (SVM) is a popular machine learning model. It can be used in regression problems, but is mostly used as a binary classifier.

As we can interpret our dataset instances as points into a multidimensional space (around 80 dimensions in our case), the SVM model will place an hyperplane and split the points in two regions. To classify a new instance, we just need to determine on which side of the plane is placed the new instance point.

As most datasets are not linearly separable, a kernel (such as a polynomial kernel) can be used withing the SVM training in order to artificially add new features and increase the number of dimensions. Two figures representing a linear SVM and a linear SVM with polynomial can be found below.

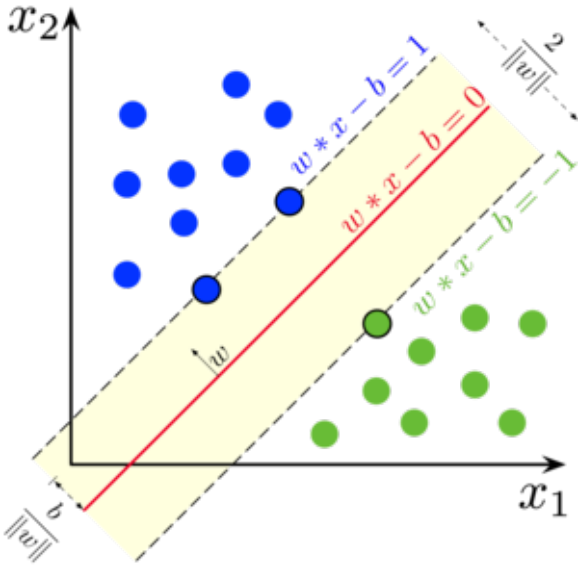


Figure 1: Linear SVM classification.

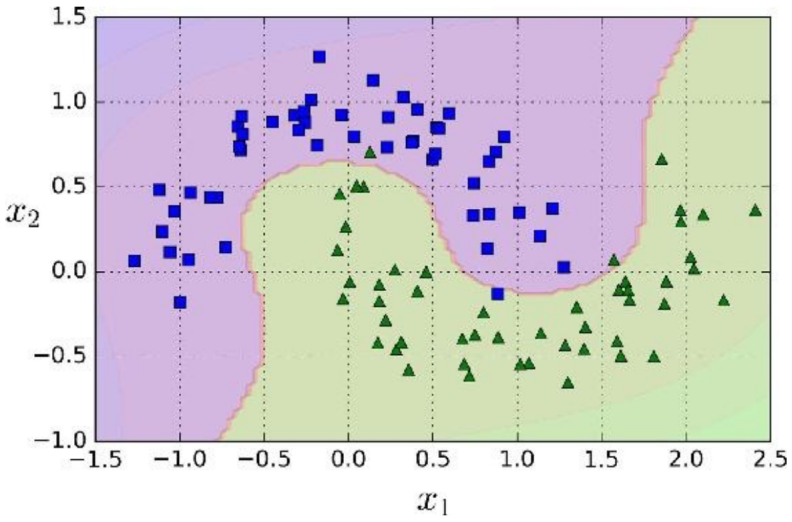


Figure 2: Linear SVM with a polynomial kernel.

As SVM can be used only in binary classification, multiclass classification can be performed by doing one-on-one comparisons and selecting the final class through a voting system.

While SVM is considered to be very performant in binary classification problems with lots of features, many hyperparameters must be tweaked in order to obtain an efficient classifier. Two of those parameters are known to be the most important among the others for polynomial kernels. We chose them during the model fine-tuning.

- As some outliers can exist (see figure 2), there is a border around the plane called *Soft Margin* that contains those outliers. The **cost** coefficient C influences the width of this margin. A highest C value leads to a wider street and more margin violations, but can also make the model generalize better. An inappropriate value of C can lead to overfitting issues.
- For the polynomial kernel, the degree deg of the polynomial must be chosen. A higher deg leads to a more accurate separation, but increases the computation complexity and can also lead to overfitting issues.

By performing a grid search for the parameters $C \in 1, 5, 10$ and $deg \in 3, 5, 7$, we obtained the best accuracy of value 0.62 in the validation dataset for the hyperparameters $C = 1$ and $deg = 3$.

3.3 Neural Network

A Neural Network (NN) is another machine learning algorithm that is inspired by the structure of the human brain. The network takes the data as input, processes the data through a pre-defined number of neurons and trains itself to recognize patterns associated with this data. Once the model is trained with the training data, the NN will be able to predict the output on a new set of similar data.

A NN is composed of three kind of layers: the input layer which receives the data, the hidden layer(s) where happens most of the computation, and the output layer which predicts the output.

We used the R package nnet which processes the data through a one hidden layer NN for which we can change the number of neurons. We had to change some parameters for the NN with more than 11 neurons: we increased the maximum allowable number of weights and the maximum number of iterations, otherwise the NN does not converge.

In order to asses the performance of the different NN, we computed the confusion matrix each time to get the accuracy. The results are shown in the table 4 below.

Table 4: Tweaking of the NN

Number of neurons in hidden layer	Accuracy
5 neurons	73.00%
8 neurons	74.51%
11 neurons	74.62
15 neurons	75.09%
20 neurons	75.09 %

We used 5 hidden layers (with 5, 8, 11, 15, 20 neurons). The least performing NN is the 5 neurons NN with a difficulty to predict the column "functional needs repairs": depending on the run, this NN does not success to assign any pumps to this category. The best performing NN, in terms of accuracy, is the 20 neurons with 75%. From 15 neurons and beyond, the accuracy score does not vary significantly.

3.4 Selection

The three models presented in the previous sections have been trained and implemented on the training set. The validation set is now used to assess and compare their respective prediction quality.

The model predicting with the best *accuracy* will be selected.

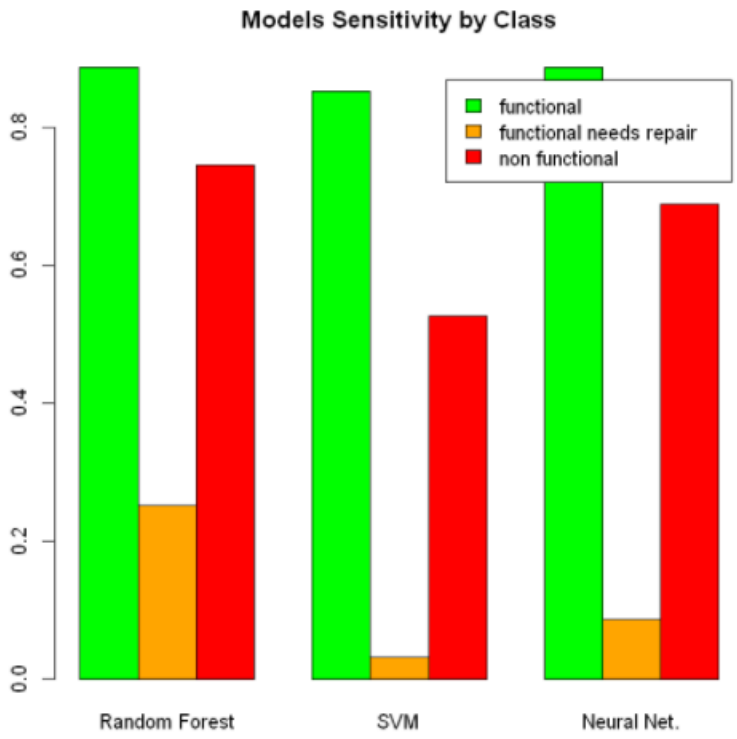
Two other metrics are also displayed in this section. The *Cohen's Kappa* is a coefficient that is useful to determine the level of agreement between two sets : here the expected and predicted labels. It takes into account to probability of agreeing by chance and also the proportion of false negative/positive predictions. The *sensitivity* (also called *recall*) permits to measure the proportion of False negative per class predicted.

The results are presented below.

Table 5: Models performances

Models	**Accuracy**	Kappa
Random Forest	78.34%	0.59
SVM	66.5%	0.34
Neural Network	75%	0.52

The Random Forest model has made the most accurate predictions on the validation set.



However, the comparison of the sensitivity per class for each model shows that each model struggle to predict correctly the element of the class *functional needs repair*. The proportion of false negative in this class is higher than 70% for each model.

4 Gradient Boosting Tree

Gradient Boosting is a technique consisting of sequentially adding classifiers one after another, creating an ensemble. Instead of performing an independant classification with dataset bootstrapping, each new classifier will fit the residual errors made by the previous classifier.

In our case, we used the library *XGBoost*. This library can determine automatically the best hyperparameters in order to obtain the best results. The classifiers used in the ensemble are simple decision trees of depth 3. The learning rate $\eta = 0.4$ scales the contribution of each tree. Decreasing the learning rate requires more classifiers, but will make our model generalize better. A low learning rate with too many classifiers will overfit, so both parameters are important to balance.

XGBoost determines automatically the number of sequential trees by measuring the validation error after each new classifier (see example in figure 3). In our case, 150 decision trees are used in the final model.

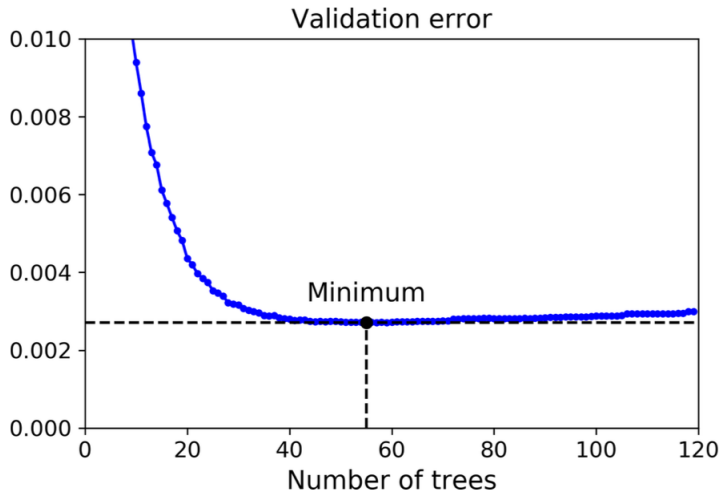


Figure 3: XGBoost - example of number of tree determination

Bootstrapping and subspace sampling are also performed on the different trees. In the best parameters found by XGBoost, only 50% of the instances are randomly selected and used on each

tree, with 80% of the features, also sampled randomly. With those parameters, the model obtained an accuracy of 67% for the validaton set.

5 Test Set - Predictions Submission

The best model implemented yet is chosen to predict the test set labels. The test set has been processed with the same parameters that the training set.
The Random Forest implementation (450 trees and 18 random features) is selected as it performed with **78.34%** accuracy on the validation set.
The predicted output has been submitted to the DrivenData website and returned with **78.12%** accuracy on the competition set.

Figure 4: Competition Ranking

6 Conclusion

Our machine learning pipeline allows us to predict with a 78 to 79% accuracy, the functionality of waterpumps present in Tanzania. We selected the best performing machine learning algorithm (among the four tested) to run the test set: Random forest. The pre-processing step is essential for the algorithm to run properly and output the best results. As a future work, looking at the good results of the NN algorithm, we could run a Deep NN on the dataset and maybe outperform the results of the Random forest.