Pump it Up: Data Mining the Water Table

(ML competition held in Driven Data)

Project Report

By

Rohith Reddy (rxk152130)

Keertan Dakarapu (kxd160830)

Nikhil Kumar Gondrala (nxg163130)

Abhishek Thangudu (axt153530)

Mahesh Paramati (mxp150830)

I. Introduction

"Pump It Up: Data Mining the Water Table" is one of the current active competitions on Driven Data. Using data from Taarifa and the Tanzania Ministry of Water, participants must build models to predict the status of a waterpoint: functioning, in need of repair, or non-functioning. In short, the goal was classification. The data provided included geographic information, structural information, and administrative information.

The challenge is quite eagerly participated by machine learners around the globe. It is a reputed platform for showcasing our machine learning talent. An understanding of how the water points and pumps work according to existing data gives a better insight into their maintenance and help the community.

II. Problem Description and Analysis

Problem Statement

The Data collected over the past years of operation on different water pumps with many environmental attributes such as Geo Location, Water basin, Region and such are given along with the status of the pump. The task is to analyse the data and find a classifier model that could predict the working status of the pumps based on given attribute data.

Set of Attributes provided

The raw dataset provided contains about 59400 instances with 40 attributes each. The data is huge to work considering all 40 attributes. We have analysed the given attributes and decided on the validity of data attribute in terms of output classification. The output classification label is from: {'functional', 'non-functional', 'functional needs repair'}

We are given another csv with 59401 instances connecting to their respective output classification (Training set)

We are given a submission csv file with 14851 instance id's, and the predicted class should be given under the 'status_group'.

a) Training dataset size: 54900
b) Test dataset size: 14850
c) Number of instances: 54900
d) Number of Attributes: 39

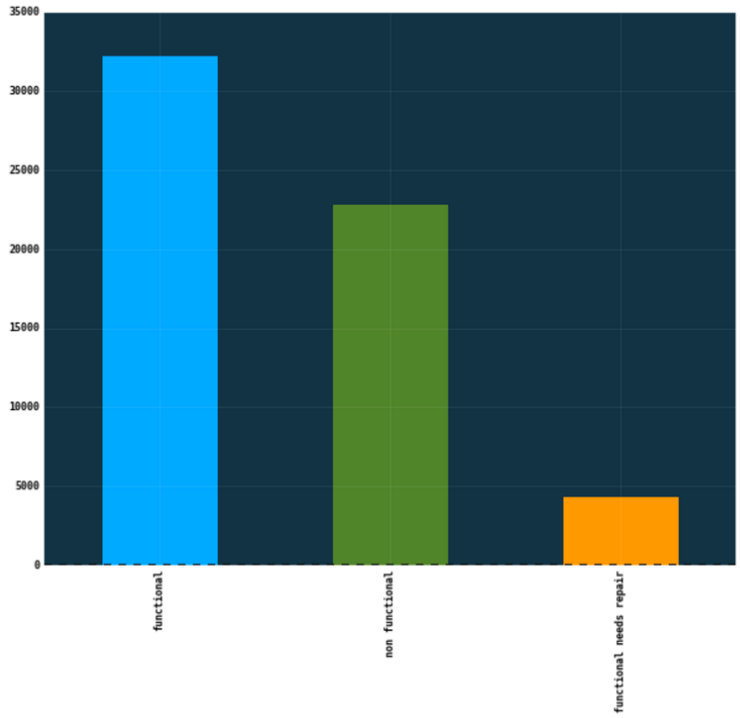
Attribute List and their Interpretation:

Attribute Name	Interpretation
1. **amount_tsh**	Total static head (amount water available to waterpoint)
2. **date_recorded**	The date the row was entered
3. **funder**	Who funded the well
4. **gps_height**	Altitude of the well
5. **installer**	Organization that installed the well
6. **longitude**	GPS coordinate
7. **latitude**	GPS coordinate
8. **wpt_name**	Name of the waterpoint if there is one
9. **num_private**	No data on this.
10. **basin**	Geographic water basin
11. **subvillage**	Geographic location
12. **region**	Geographic location
13. **region_code**	Geographic location (coded)
14. **district_code**	Geographic location (coded)
15. **lga**	Geographic location
16. **ward**	Geographic location
17. **population**	Population around the well
18. **public_meeting**	True/False
19. **recorded_by**	Group entering this row of data
20. **scheme_management**	Who operates the waterpoint

21. scheme_name	Who operates the waterpoint
22. permit	If the waterpoint is permitted
23. construction_year	Year the waterpoint was constructed
24. extraction_type	The kind of extraction the waterpoint uses
25. extraction_type_group	The kind of extraction the waterpoint uses
26. extraction_type_class	The kind of extraction the waterpoint uses
27. management	How the waterpoint is managed
28. management_group	How the waterpoint is managed
29. payment	What the water costs
30. payment_type	What the water costs
31. water_quality	The quality of the water
32. quality_group	The quality of the water
33. quantity	The quantity of water
34. quantity_group	The quantity of water
35. source	The source of the water
36. source_type	The source of the water
37. source_class	The source of the water
38. waterpoint_type	The kind of waterpoint
39. waterpoint_type_group	The kind of waterpoint

Table 1.1

Label Distribution



The labels are not completely evenly distributed, though there are ample number of instances in each label. The following bar graph shows a histogram of label distribution.

Attribute Analysis

The Attributes given and the no of instances given are humongous when compared to normal learner training we have done so far. Hence it requires special view into it. The major problem with high amount of training data being available is overfitting. We have to make sure that our model should not over fit. One way to do this is by discarding some of the non-useful attributes. We have started analysis of the attributes and have found a few rouge attributes where there are more than 100 levels and a low correlation. Such attributes are bad for the model creation. Some of the attributes we found have no logical correlation to the data classification neither in real time, nor in the correlation plots we made. Num_Private is one such attribute. Hence we discard attributes like that.

Over fitting avoidance should not interfere with our path to greater accuracy. Hence, we don't discard the attributes only based on no of missing values and levels. Amount_tsh is such an attribute. We have to consider it as it makes complete logical sense as to keep it. We have many NA values in Amount_tsh but it evidently plays a major role in classification.

Several attributes such as region, region code, basin, etc., have very high correlation amongst themselves, and are also important for the model. But, considering all the attributes when correlation amongst them is so high (almost 0.95) results in model giving them too much preference, and ignoring other attributes. We have hence, disregarded a few attributes who have a high correlation amongst themselves.

One of the attributes that we have created for better model performance is the 'Year in operation' attribute(YIP). We have converged two attributes into another attribute through a logical operation.

```
#Creating New attribute called 'YIP' for testdata

tmp1 <- format(as.Date(testdata$date_recorded, '%Y-%m-%d'),'%Y')

tmp1 <- as.numeric(tmp1)
```

This feature engineering of which attributes to keep and which to remove plays a pivotal role on model performance. Below is the table for feature engineering and reasons for their dismissal.

Feature Engineering (Attribute removal/ Inclusion)

Attribute	Removed	Reason
1. date_recorded	Removed	This field is being replaced by another field 'Years in operation'
2. wpt_name	Removed	The attribute has no logical relevance and too low correlaiton
3. num_private	Removed	This attribute has no logical relevance and no explanation, and most of them have 0 value
4. subvillage	Removed	Sub village has too many levels and low correlation
5. region_code	Removed	Too many NA Values
6. district_code	Removed	Redundant data Attribute
7. lga	Removed	Too many levels
8. ward	Removed	
9. recorded_by	Removed	Attribute having no logical significance and low correlation too.
10. scheme_name	Removed	Attribute having no logical significance and low correlation too.
11. permit	Removed	Too low correlation.
12. extraction_type	Removed	Attribute with redundant data.

13. extraction_type_class	Removed	Attribute with redundant data.
14. management_group	Removed	Attribute with redundant data.
15. quality_group	Removed	Attribute with redundant data.
16. quantity_group	Removed	Attribute with redundant data.
17. source_type	Removed	Attribute with redundant data.
18. waterpoint_type_group	Removed	Attribute with redundant data.
19. Years in Operation	Included	Extra attribute added instead of year of opening and date recorded.

Table 1.2

Pseudo Code

The data needs a strong classifier algorithm to obtain maximum accuracy. Attaining such an accuracy with an independent classifier is highly improbable. Using a single independent model would have huge overfitting issues and would not work well with the test data. Hence we use a combination approach such as ensemble methods. We would use bagging and boosting for creating a stronger classifier.

Steps:

1. **Data Cleaning**
   Finding the attributes which actually affect the output classification. We plan to do this by eliminating redundant attributes such as, considering only 'Region_id' and discarding 'Region'. Clean the data of any 'NA' values. For this, we first generate the data set without irrelevant attributes. Now, of those instances, we consider instances with more than

   Data-Attribute <- NULL

2. **N-Cross Validation (For applying on training set only)**
   Divide the data set appropriately for cross-validation, for say if n-fold cross validation, then the whole data set would be divided into n sub sets where in each set would be selected as a testing set and the others act as training data for the learner.

3. **Use the required strong learner (Ensemble methods)**
   Use Random Forest technique to create subsets of training data with random instances [Bootstrapping] but, with random attributes too.
   Using Gradient Boosting technique, for averaged classification using multiple weak classifiers.

4. **Increasing Efficiency and Accuracy**
   Depending on output accuracy, change these factors:
   a. Classifier Technique that is used.
   b. Attributes to be included for model creation.

III. Actual Method

After feature engineering the next step is to develop a learner using the training data. As the training data is very complex we need to use complex learners to achieve high accuracy. We chose to use ensemble learning methods. The methods we chose to implement are *'Random Forest'* and *'Gradient Boosting'.*

***Random Forest**:*

Random Forest is a combination of Bootstrapping and Random attribute selection. This methods basically creates decision trees using random attributes at different levels of the learning process. And now because of bootstrapping we now have more data and in machine learning more data is always good.

The reason for selecting this method is to avoid over fitting which was the problem with decision trees. To avoid over fitting instead of taking all the available attributes into consideration at any given point, random forest randomly selects a set of attributes. The decrease in accuracy of training data is very minute compared to decision trees but the increase in accuracy of testing data will be much higher compared to decision trees.

We now implement this method using R. We use a package called 'randomForest' to implement our methodology. Implementing this is pretty straight forward. We have a function called `randomForest()` to implement random forest. We also implement cross validation to get MSE over the training data.

```
cvresults <- rfcv(traindata, traindata$status_group, cv.fold = 10)

rf_model<-randomForest(traindata$status_group~.,data = traindata, ntree
= 100)
```

We then use the predict function to predict the classes for testdata

```
predtemp <- predict(rf_model, testdata)
```

For the initial testing we fixed ntree to '100' and used feature engineering to get the best attribute set. We now report accuracies for both training and testing data with varied ntree values. The training data set is divided into two parts: 90% for training the data and 10% for the validation data set.

ntree =	Validation Data Accuracy	Test Data Accuracy
100	81.00	81.16
200	81.15	81.03
300	81.32	81.16
400	81.38	81.19
500	81.27	81.18

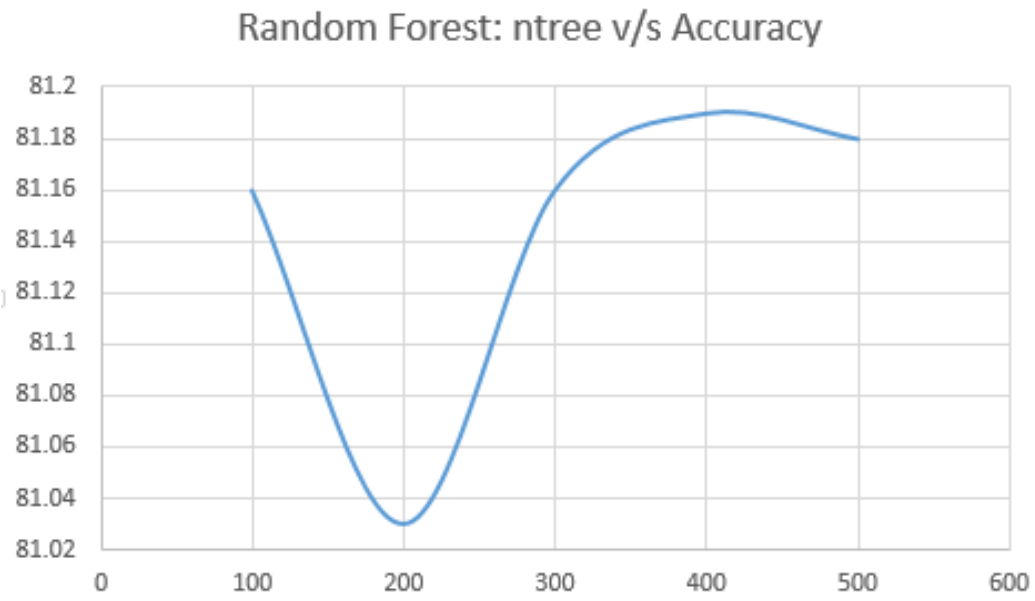


Fig. : Random Forest Accuracy Curve

***Gradient Boosting**:*

Random Forest method gave a best accuracy of 81.19% but generating the model took a lot of time(~20 minutes) and we also wanted to improve our accuracy. So, we choose a new technique called Gradient Boosting which is an extension to the traditional boosting technique by combining it with gradient descent algorithm.

The general concept in boosting is to introduce a weak learner at every stage of learning. In the normal adaboost we use the weights of the data weights to introduce a weak learner and compensate for weak learner. The method in gradient boosting is different where we use the negative gradients of each class to overcome the shortcomings.

Similar to Random Forest we divide the dataset into training set and validation set with 90:10 ratio. We present the accuracies of our model on validation set. To use the training data to train the model, gradient

boosting package in R requires that the data is in `xgb.Dmatrix` format. The following commands converts the data set to `xgb.Dmatrix` format.

```
newtrain <- xgb.DMatrix(train.xgb, label = TL)
```

The above line only works if `train.xgb` is in either dense or sparse matrix format.

Gradient Boosting takes a lot of parameters to model the training dataset.

```
param.xgb <- list(objective = "multi:softmax",eval_metric = "merror",num_class = 3, booster = "gbtree",eta = 0.2,subsample = 0.7,colsample_bytree = 0.4, max_depth = 14)
```

Cross validation is performed using the following line

```
results <- xgb.cv(params = param.xgb, newtrain, nrounds = 200, nfold = 10,  early.stop.round = 20,maximize = FALSE, print.every.n = 10)
```

The main parameters that affect the accuracy are 'eta' and 'maxdepth'. After numerous experiments the best value for eta is 0.2 and the best value of max_depth = 14. Now we train the data using the following line.

```
model.xgb <- xgb.train(data = newtrain, param.xgb, nrounds = 200, watchlist = list(valid = newvalid, train = newtrain), nfold = 10, early.stop.round = 20, print.every.n = 10,maximize = FALSE,save_name = "model.xgb")
```

The above line performs the gradient boosting for '200' rounds. The drawback for performing these many rounds is that the model may over fit. So, we need to stop training once the accuracy on validation set starts increasing. Thanks to the very customizable xgb.train() function which outputs the best round we can easily avoid overfitting. One interesting observation we found is that setting the seed before training affected the model. We now present the results for different seed values for their best rounds.

set.seed(i); i =	**No. of rounds (Best Iteration)**	**Validation Set Accuracy**	**Testing Set Accuracy**
1	46	81.86	81.47
2	42	81.83	81.41
3	51	81.71	81.56

4	31	81.73	81.52
5	**56**	**82.52**	**81.41**
6	35	81.81	81.39
7	28	81.85	80.96
8	28	81.75	81.39
9	**38**	**81.61**	**81.76**
10	38	81.71	81.16
11	77	82.08	81.34
12	71	82.00	81.43

To find the no. of rounds (best iteration) we first run the model for all seed values with nrounds = 200 in xgb.train() function. Then xgb.train() function outputs the best iteration based on the error rate for validation set. This best iteration value is used as the no. of rounds to model the data for testing data set.

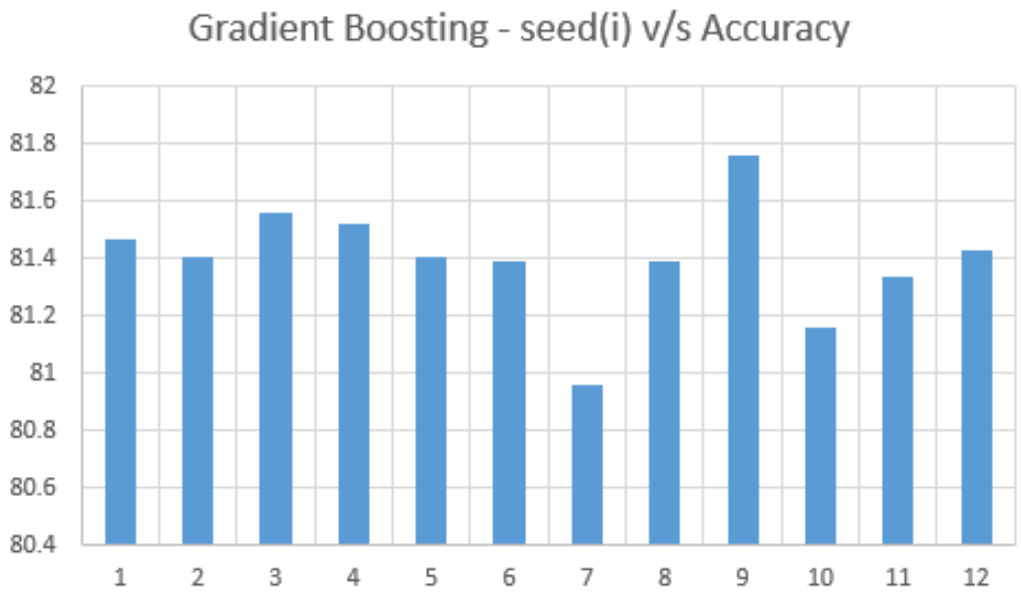
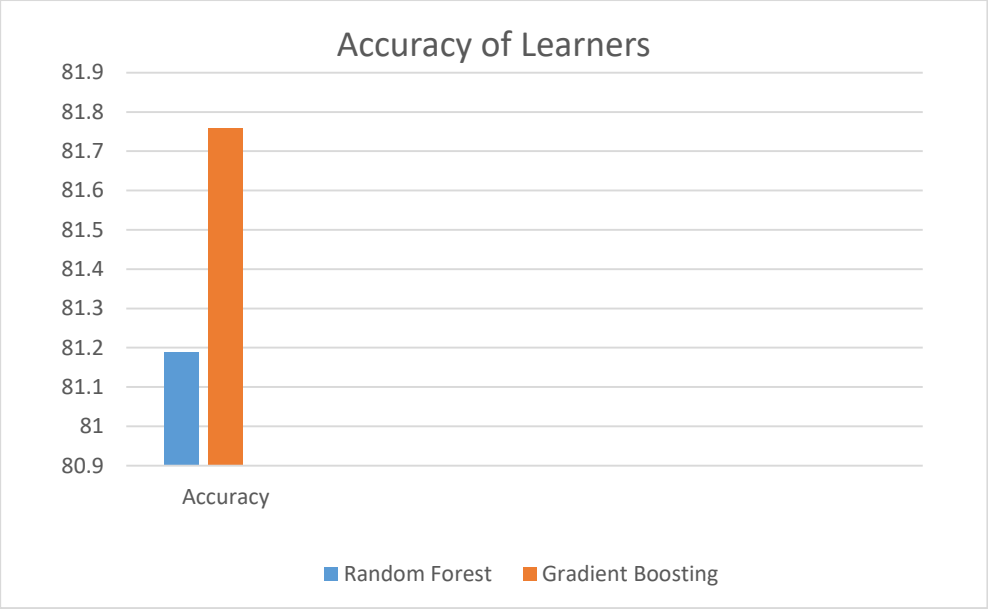


Fig. : Gradient Boosting Accuracy Curve

IV. Results

Conclusion

The obvious contenders for the final algorithm are Gradient Boosting and Random Forest. Random Forest is a widely accepted as a strong learner and has a good theoretical base for prediction. Boosting also has many followers who bet on its supremacy as a strong learner. But, a good learner should also be efficient whilst giving out good accuracy.



Considering our case in this scenario, Random Forest and Boosting have given similar accuracy, but, considering the running time of both the algorithms, Random Forest is a heavy learner program which takes about 20 folds more time than Gradient Boosting. Hence, here Gradient Boosting is a winner hands down.