Get Pumped: Water Points in Tanzania

Ethan Buck, Leyli Garryyeva, Gergana Gospodinova, and Xin Zhang

April 27, 2019

1 Problem Description

This project stems from finding a data set from a competition on DrivenData. The competition is called "Pump it Up: Data Mining the Water Table", and provides a data set with different water points. There are six different types of water points: cattle trough, communal standpipe, dam, hand pump, improved spring, and other.

Tanzania is one of the African countries where over 20% of the population lacks access to drinking water that is protected from contamination and 10% of the population spends more than 5% of its expenditure on purchasing drinking water. Therefore, determining which water points are nonfunctional could help reduce such costs and improve access of the population to safe drinking water.

The goal is to accurately predict the operating condition of these water points located in Tanzania. There are three different classes for this response variable, which are

1. FUNCTIONAL
2. FUNCTIONAL NEEDS REPAIR
3. NON FUNCTIONAL

Thus, this is a classification problem. There are 39 different predictors for this data set, nine of which are continuous covariates. Early data exploration reveals that none of these continuous covariates appear to resemble a normal distribution, and thus LDA and QDA most likely are not suitable choices. However, an adapted logistic regression may prove to be fruitful in this analysis, as well as a potential random forest model. KNN classification may be suitable as well, since there are a large number of training observations (59,400).

Section 2 will discuss the data cleaning process. Exploratory plots are shown in Section 3. The KNN analysis follows in Section 4. Section 5 goes through the logistic regression results, and Section 6 explains the results from random forests. In Section 7, we are going to compare performance between models.

2 Data Cleaning

The data cleaning process was mostly straightforward, though there were some key discoveries that helped immensely for the model fitting.

We first started by getting rid of factors that had too many levels, as we feared adding hundreds of parameters was unnecessary and would lead to over-fitting by the model. After inspecting the data to see how many levels there were for all of the factors, we set the boundary to be at most 21 levels for each factor. The variables that were removed from this process were:

1. date recorded: the date the row was entered. 356 levels
2. funder: who funded the well. 1,898 levels
3. installer: organization that installed the well. 2,146 levels
4. wpt name: name of the water point if there is one. 37,400 levels
5. subvillage: geographic location (also represented by other variables). 19,288 levels
6. lga: geographic location (also represented by other variables). 125 levels
7. ward: geographic location (also represented by other variables). 2092 levels
8. scheme name: who operates the water point. 2697 levels

These variables also make sense to remove, as they appear to be more miscellaneous information that would not provide useful information for predictions (except for geographic location, which is represented by other variables remaining in the data set). Additionally, the "recorded by" variable was a factor with only one level, which was "GeoData Consultants Ltd", and thus this was removed as well.

There are many categorical predictors, and after looking again at the structure of the dataset we realize that many of them are duplicates of others. By this we mean that some columns are exactly the same, or that some predictors just group some of the labels together, but represent the same measurement. We examined each variable using the table() R command, to see all of the unique label names and counts and compare them with their similar counterparts. In most cases, we chose to keep the predictor with less labels (or the one that grouped together more categories) so that there will be less parameter estimates, saving important degrees of freedom. However, if we thought that the labels should not have been grouped together, and that their responses would vary significantly, we chose to keep the predictor with more variables. The choices are summarized in Table 1.

There were minor relabeling of factor labels and grouping of certain labels together that did not have many counts into an "other" category, but these were the main steps taken in the data cleaning process.

Table 1: Repeat factors that were removed due to containing the same type of information as others.

Removed predictors	Represented By
extraction type, extraction type group	extraction type class
management	management group
payment type	payment
water quality	quality group
quantity group	quantity
source type, source class	source
waterpoint type	waterpoint type group

3 Exploratory Analysis

We wanted to first see if LDA and QDA were potential models by checking for normal covariates. We saw that there were a large number of factors in the data, and so we tried only considering the numerical predictors, and used histograms to see if we could rule out normal distributions. These are shown in Figure 1. It is clear that all of these are far from being able to be assumed to follow normal distributions, and thus we presume that LDA and QDA models are not appropriate and would not yield predictive models.

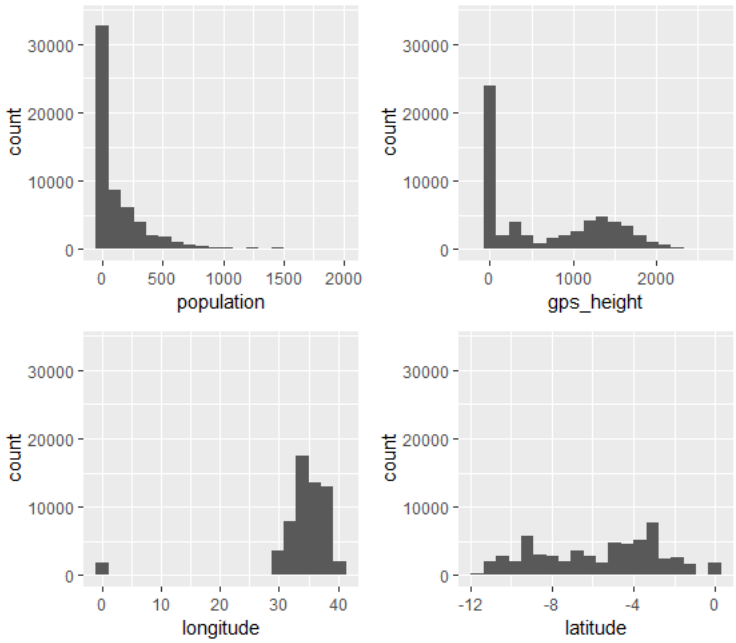

Figure 1: Histograms of some of the numerical predictors.

More exploratory plots are shown in Appendix A.

4 KNN Classification

In this section, we are focusing on using k–nearest neighbors (also known as KNN) algorithm to fit our data. To prepare for our KNN model, we apply one-hot encoding to transform all the categorical features with n possible values into n binary features, in order to make sure our KNN model can measure the “distance" between categorical features well.

Our first attempt with KNN is trying to fit a model on all the features. When we finished our data cleaning process, we had 23 features left; however, after we perform one-hot encoding to all categorical features, we now have 101 columns. As KNN is known for computationally expensive, with a large amount of feature, it took 19 hours to test our KNN model with 10 fold cross-validation using 50 different value of k's. Despite the long processing time, we only achieve a accuracy of 70.44% on the test data.

Next thing we tried is reducing feature columns in order to reduce running time and to increase model accuracy. By examining the data, we started with deleting columns containing more than 9 levels. Then, we applied `evimp` function in `earth` package to help us determine which columns are of more importance to the model. We ended up choosing 22 “important" columns among the original 101 ones. For a variable, the more model subsets can be generated by `earth`'s backward pass, the more important the variable is. By doing so, not only we reduce the run time for the model from 19 hours to 30 minutes, we also increase the accuracy to 76.64%. The confusion matrix is shown below in Figure 3, and the model accuracy associated with our choice of k's in shown in Figure 2. For our data set, a small $k = 5$ or 7 seems work the best.

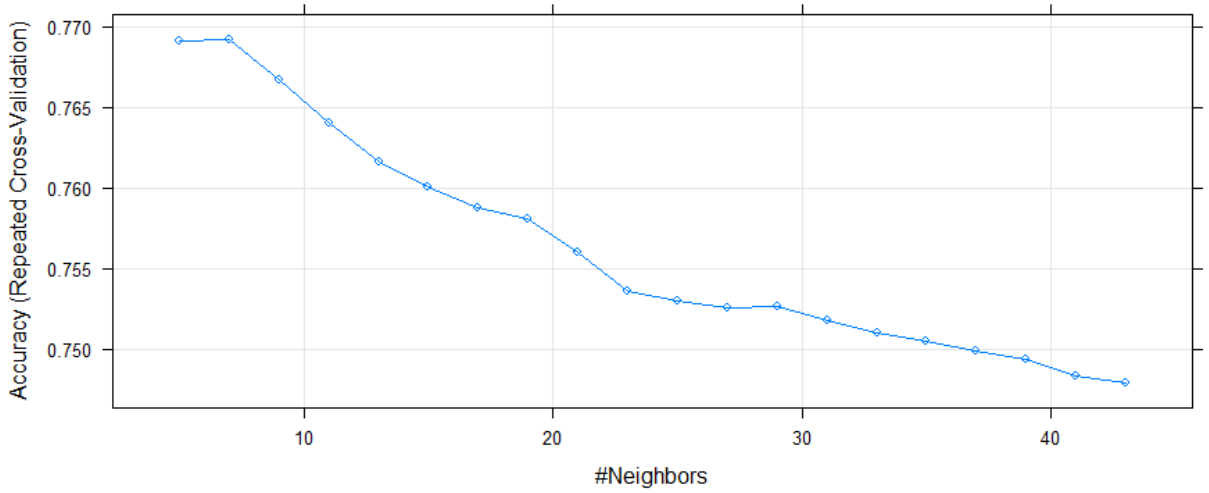

Figure 2: KNN model with different K's

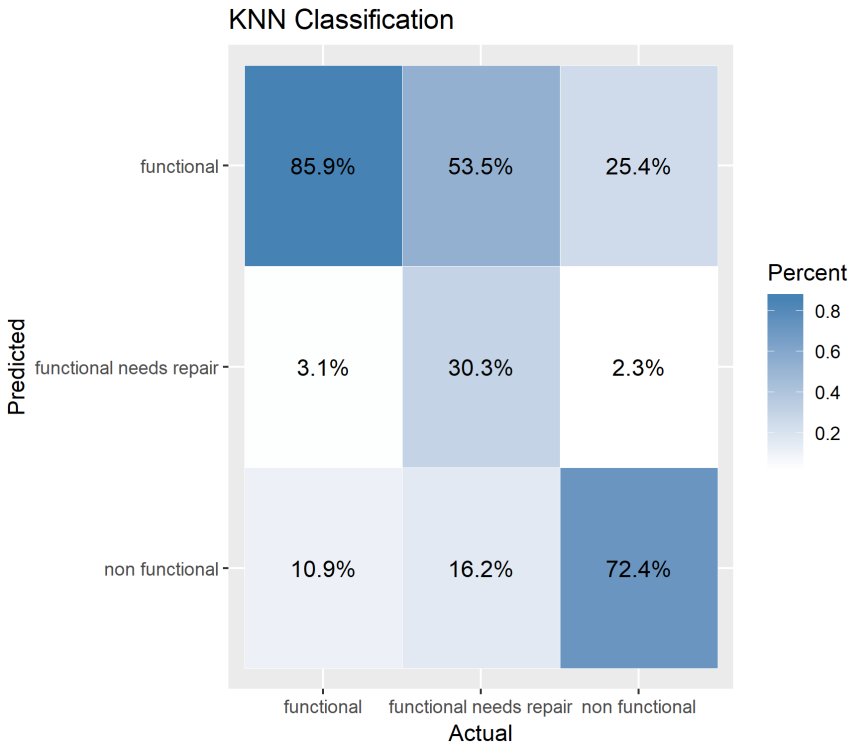

Figure 3: Visualization of Confusion Matrix for KNN Model

As we observed from above confusion matrix and other models we tried, while the model predicts the FUNCTIONAL and NON FUNCTIONAL class relatively well, our accuracy on FUNCTIONAL NEEDS REPAIR class are far from satisfying. Taking a look back at our data set, there is only 7% data are in the FUNCTIONAL NEEDS REPAIR class, and this imbalanced issue leads to low accuracy of this class. We adapt Synthetic Minority Over-sampling Technique(SMOTE) to help us "create" more minority class samples. The overall accuracy of KNN model after we re-sampling the data using SMOTE decreases to 71.03%, however, the average accuracy between three classes increases from 62.87% to 66.82%, and the accuracy on FUNCTIONAL NEEDS REPAIR class increased from 30.3% to 68.5%. Eventually, it's a trade-off between overall and average accuracy.

5 Logistic Regression

This section describes attempt to predict the condition status of the water pump by means of multinomial logistic regression (MLR). The model was fit using `multinom` function from the `nnet` package in R to perform MLR. The package uses feed-forward neural networks with a single hidden-layer and `multinom` function fits multinomial log-linear models. MLR is an extension of a simple logistic regression that allows for predicting variables with three or more categories.

The target categorical variables are FUNCTIONAL, FUNCTIONAL NEEDS REPAIR, and NON FUNCTIONAL which correspond to the status of the water pump's condition. Prior to fitting the model, the data was split in the ratio of $1/3$ and $2/3$ to create training and test datasets, respectively.

The initial model included all the features and the accuracy rate for training data and testing data were 73.41% and 72.68%, respectively. When applied to the whole training, the accuracy rate was 73.22%. After submitting the predictions to the competition website we got the score of 71.43%. Figure 4 shows the confusion matrix associated with this model:

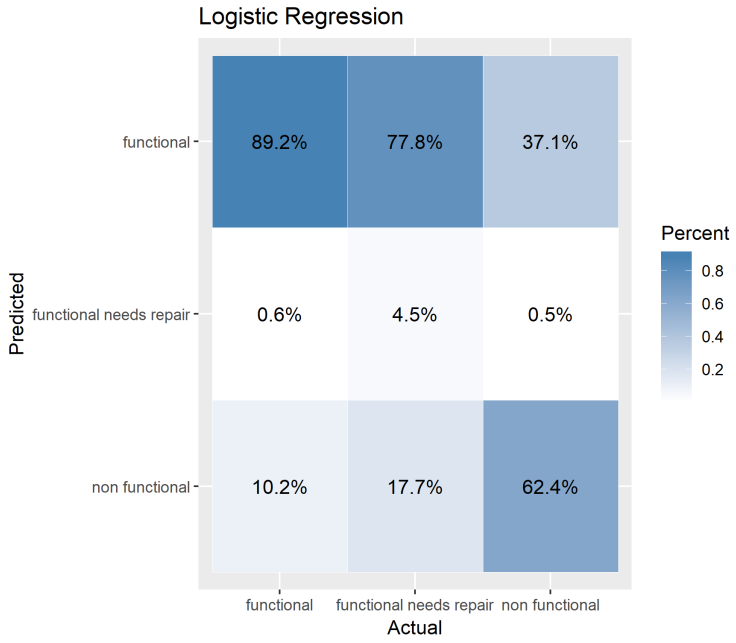

Figure 4: Confusion matrix for the logistic regression model.

The confusion matrix indicates that most of the inaccuracy is associated with miss-classification of the FUNCTIONAL NEEDS REPAIR category with 95.5% overall error rate where 77.8% of this class is miss-classified as FUNCTIONAL and 17.7% as NON FUNCTIONAL. The NON FUNCTIONAL category is the second most inaccurately classified category with 37.6% error rate where 37.1% of this class is miss-classified as FUNCTIONAL. Lastly, only 10.8% of functional pumps were miss-classified.

Further, the prediction accuracy was compared among several models with varying features and interactions between certain features, which seemed to perform better on the training data. The best rate recorded for the training data was equal to 73.68%. However, after comparing the accuracy rates for the test data, it was found that the prediction accuracy did not differ between these models. The highest submission score achieved after all these attempts indicated 72.11% accuracy.

6 Random Forest Model

The third and last model we considered was Random Forest. It proved to be the most effective and easy to use algorithm out of all three. Not only did it not require any further preprocessing of the data such as transforming categorical labels or scaling, but it also achieved the highest accuracy on the test set ($\approx 81\%$) without much tuning of the hyperparameters.
One big advantage of Random Forest is that you can measure and plot the importance of each feature. This way we can make sure we are only using features that contribute to the prediction process and drop all the features that don't really affect the accuracy, as using too many (unnecessary) features might lead to overfitting.
If we take a look at Figure 5, we can see the importance plot for our problem. Nothing really stands out, except for num_private(as we can see it doesn't really affect the accuracy as much). Turns out that after dropping num_private(the feature with the lowest value for both the mean decrease accuracy and mean decrease gini) we improved the accuracy by 1%. Even though not much tuning

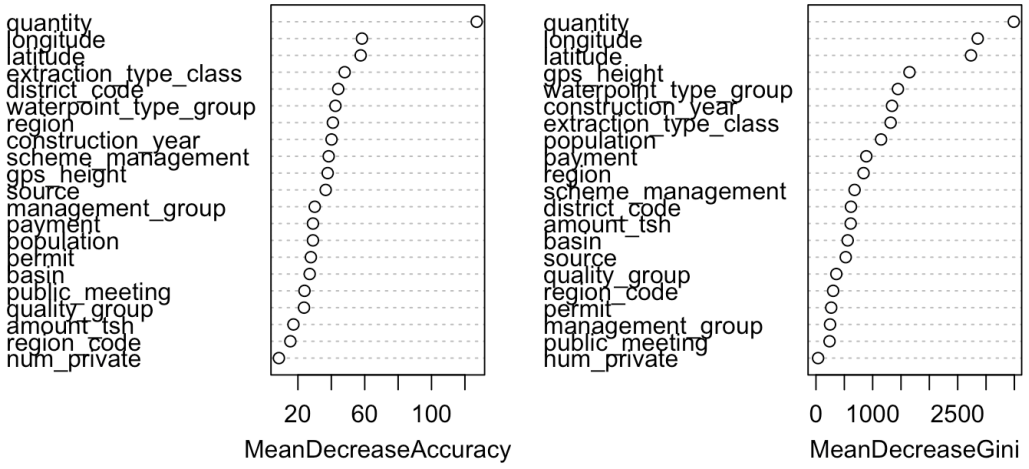

Figure 5: The importance of the features for the Random Forest Model

of the hyperparameters was needed to achieve good accuracy, some values for these parameters proved to lead to a slightly better solution than others.

1. ntree - or number of trees is the parameter that tells us how many trees the algorithm builds before taking averages of predictions. As we can see in Figure 6 the greater the number of trees, the better the accuracy. The only downside to using a large number of trees is that the computation time increases. If we don't see significant/any improvement in accuracy we shouldn't increase the number of trees as it'll only make the training process slower. From Figure 6 we can see how the accuracy changes as we increase the number of trees. There isn't a big jump in accuracy from 50 trees to 100, so this why we have decided to pick 100 for ntrees and not increase the number any further.

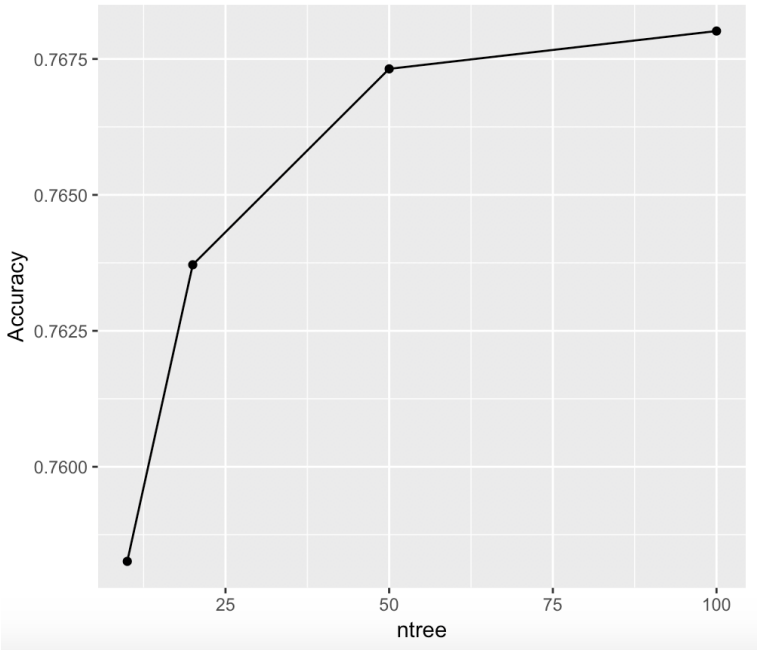

Figure 6: The effect of the number of trees on the accuracy of the Random Forest Model

2. mtry - or the number of variables randomly sampled as candidates at each split. While some researchers claim that varying the values of mtry doesn't really affect the classification rates, others claim that it has a strong influence on the importance estimates. A good recommended value for classification problems is the square root of features. In our case we have 20 predictor variables, if we take the square root of 20 and round it up we get 5 and this is the value we picked for mtry.

3. nodesize - or the minimum size of terminal nodes. We ended up using the default value for classification(1). Picking a large nodesize results in less computation time and small loss of accuracy. Since in our case the computation time isn't really an issue, we would not benefit in any way from picking a larger value for this hyperparameter.

So far we have only focused on talking about the accuracy(as it is the most commonly used to measure the performance), but there are other measures that can prove to be equally if not more important depending on the specific problem.
For the problem we are trying to solve, the most important thing is to predict the following two classes: NON FUNCTIONAL and FUNCTIONAL NEEDS REPAIR. As we can see from the confusion matrix below, the model does a good job at predicting the NON FUNCTIONAL, but it does poorly on FUNCTIONAL NEEDS REPAIR (50% of the FUNCTIONAL NEEDS REPAIR are classified as FUNCTIONAL). The results are not surprising, since we have very few instances of the two classes in the dataset compared to the FUNCTIONAL class.

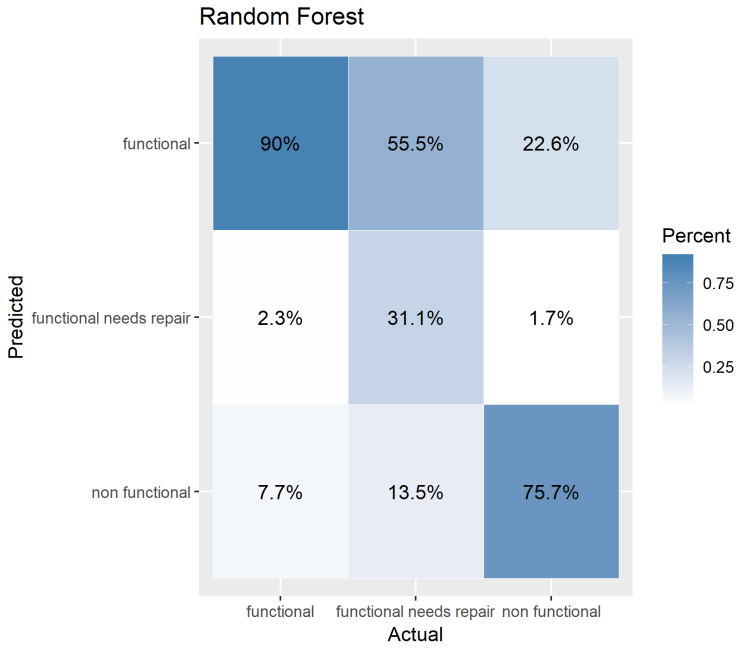

Figure 7: Visualization of Confusion Matrix for Random Forest Model

In the table below we can see some statistics by class. The specifity or "true negative rate" is pretty high for all classes except for the FUNCTIONAL. We already discussed the sensitivity or the "true positive rate". Last but not least, let's take a look at the balanced accuracy or the average of the proportion corrects of each class individually. If the test set is not balanced(which is most likely the case), it's better to use balanced accuracy instead of accuracy. While we can see from the confusion matrix that FUNCTIONAL has the highest accuracy, it turns out that NON FUNCTIONAL has the highest balanced accuracy.

	Class: functional	Class: functional needs repair	Class: non functional
Sensitivity	0.8853	0.32042	0.7838
Specificity	0.7470	0.97923	0.9035
Pos Pred Value	0.8048	0.54382	0.8371
Neg Pred Value	0.8468	0.94911	0.8686
Prevalence	0.5409	0.07172	0.3874
Detection Rate	0.4789	0.02298	0.3036
Detection Prevalence	0.5950	0.04226	0.3627
Balanced Accuracy	0.8161	0.64983	0.8437

Figure 8: Other statistics by class from the confusion matrix for the Random Forest Model

6.1 Oversampling the FUNCTIONAL NEEDS REPAIR class

In all of the models, our visualization of the confusion matrices showed that this particular class was hard to predict accurately. This is due to the fact that there are far fewer occurrences of this class relative to the others in the training set (approximately 7% of the data). Thus, the random forest model was utilized again, but this time with oversampling this class. We used 1.5 times the number of occurrences of this class (sampled with replacement), and 70% of the occurrences of the other classes in our new training data. We also made sure that the test data included all points not in the training data.

The visualization of the confusion matrix is shown in Figure 9. Note that we now have a more balanced confusion matrix, and are close to predicting the FUNCTIONAL NEEDS REPAIR class correctly most of the time.

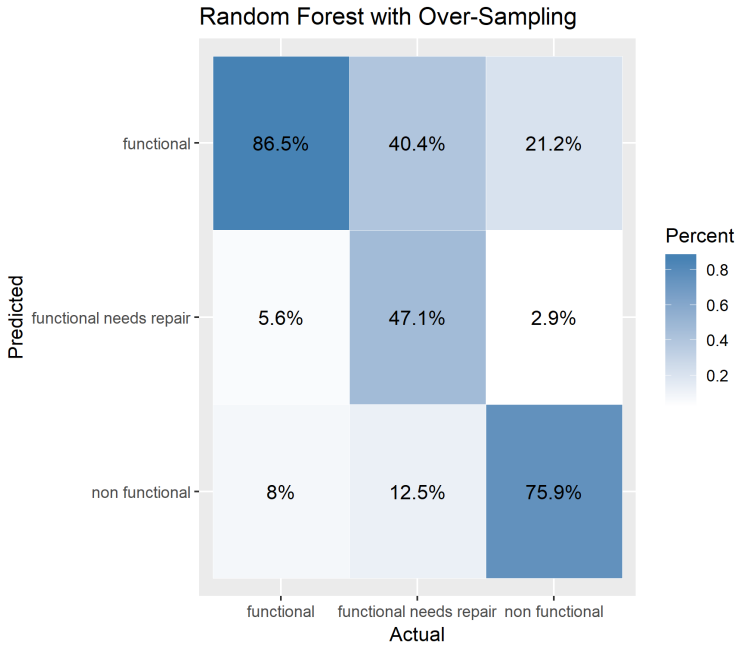

Figure 9: Visualization of Confusion Matrix for Random Forest with Oversampling

7 Comparing the Models

Figure 10 shows a bar graph comparing the performances of the different models. The models seem to perform very similarly on the FUNCTIONAL water points, as there are so many observations of this type. The models appeared to suffer much more when trying to predict the FUNCTIONAL NEEDS REPAIR class, as there are not many observations in this category (comparatively). However, the oversampling method with the random forest model did help to combat this. This method also comes close to doing the best on the NON FUNCTIONAL class, and these two classes are the most important to predict correctly, as the purpose is to identify broken water points.

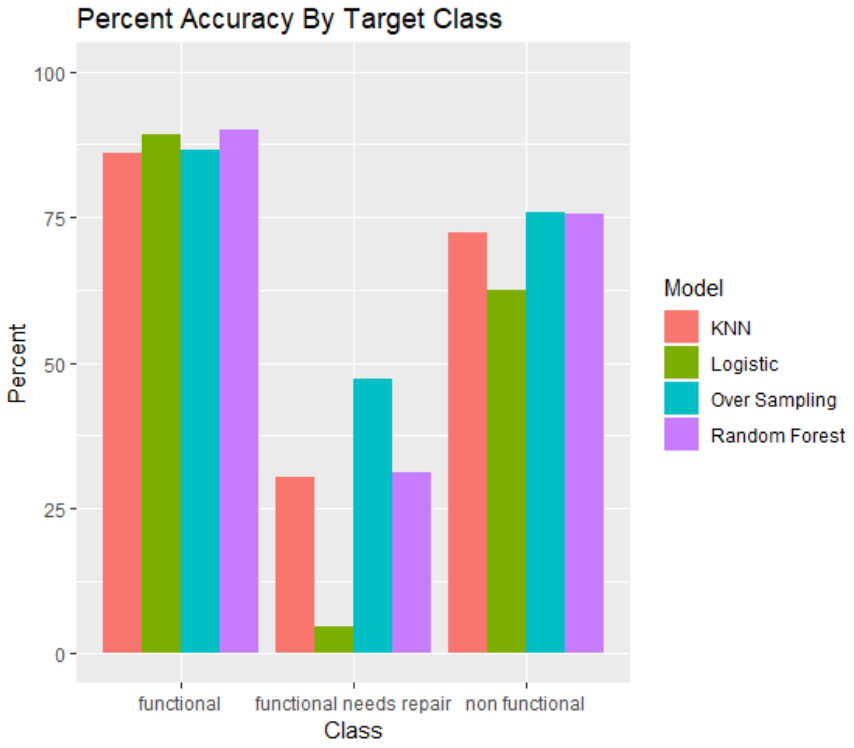

Figure 10: Comparing the models' accuracy of predicting each target class

Overall, the random forest model outperformed the other fitting methods, and this was without requiring excessive tuning of the model. The pre-processing of the data was enough to be able to fit a relatively accurate model, and with the addition of oversampling, we were able to achieve more balanced class accuracy rates.

A More Exploratory Plots

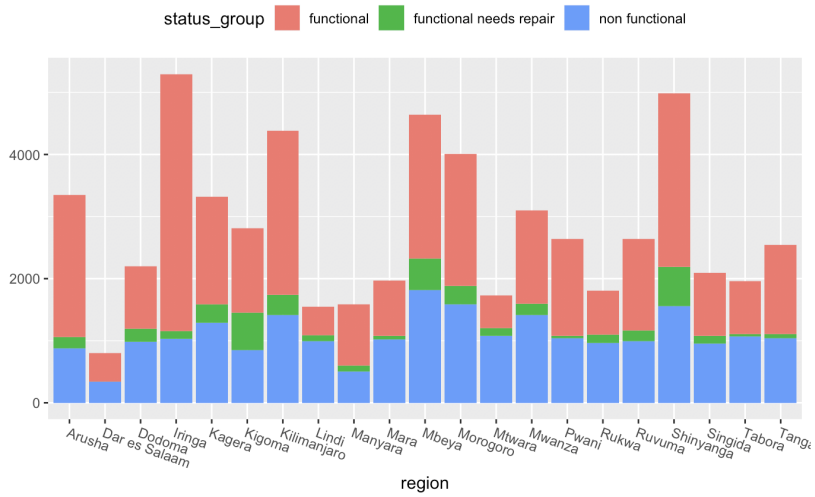

Figure 11: Looking at the region predictor variable and its effect on the response

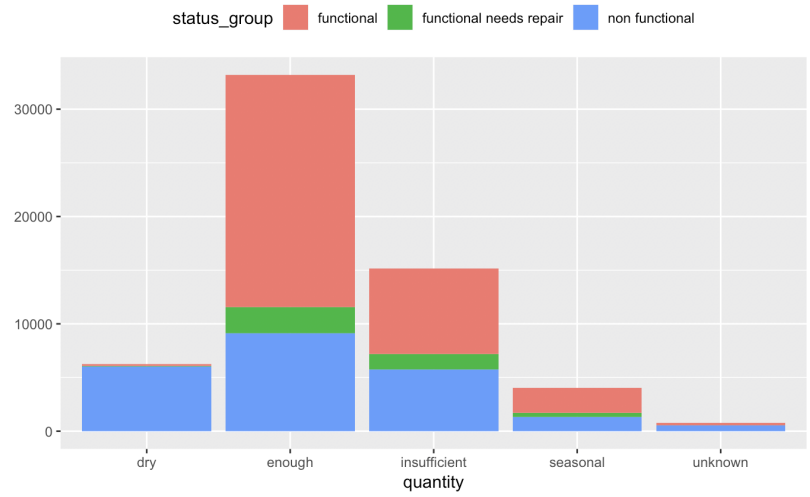

Figure 12: Looking at the quantity predictor variable and its effect on the response

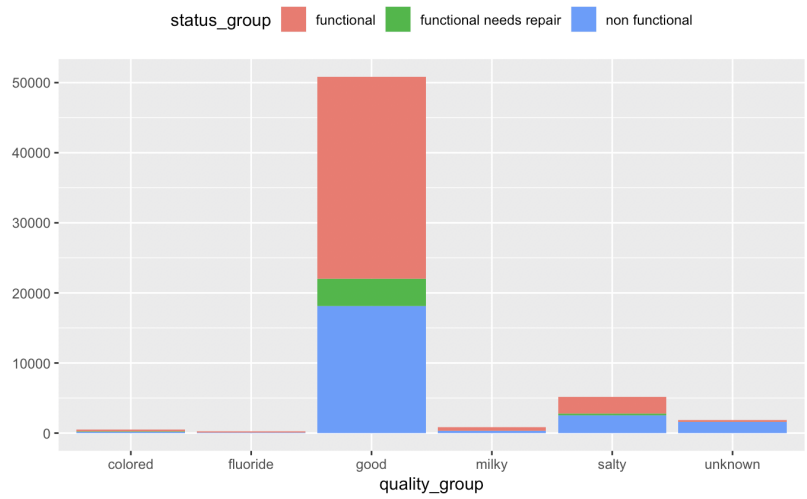

Figure 13: Looking at the quality_group predictor variable and its effect on the response

B Data Cleaning Code

```
set.seed(42)
water <- read.csv("../Data/waterTraining.csv")
trainLabels <- read.csv("../Data/waterTrainingLabels.csv")

dim(water)
colnames(water)
head(water)

unique(trainLabels$status_group)
length(unique(water$installer))  # checking number of unique installers - too many to add all covariates
                                 # could maybe group them together though

#str(water)
which_numeric = sapply(1:ncol(water),
                       function(x) is.numeric(water[,x]) &&
                         all(is.finite(water[,x])))

str(water[, which_numeric][, -1])  # getting rid of id and seeing just numeric columns

any(apply(water, 2, function(x) any(is.na(x) | is.infinite(x))))  # No N/A or Inf!

NZero <- data.frame(ColName = c("0", "0"), numZeros = c(0,0))
for (col in colnames(water)) {
  nzero <- sum(water[, col] == 0)
  NZero <- rbind(NZero, data.frame(ColName = c(col), numZeros = c(nzero)))
}
NZero <- NZero[3:nrow(NZero),]

dim(water)
head(water[, "amount_tsh"])

boundary <- 21
count <- 0
indexes <- c()
for (colIndex in 1:(ncol(water))) {
  if (length(levels(water[, colIndex])) > boundary) {
```

```
    count <- count + 1
    indexes <- c(indexes, colIndex)
  }
}
count
colnames(water[, indexes])

length(levels(water[, "installer"]))  # Thought could be useful but too many
redWater <- water[, -indexes]
redWater <- subset(redWater, select = -c(recorded_by))   # Factor with only 1 level

# source_type and source_class is shown in source column, but
#    we need to combine "unknown" category into "other"
redWater <- subset(redWater, select = -c(extraction_type,
                                         extraction_type_group,
                                         management,
                                         payment_type,
                                         water_quality,
                                         quantity_group,
                                         source_type,
                                         source_class,
                                         waterpoint_type))
dim(redWater)

redWater$source[redWater$source == "unknown"] <- "other"
levels(redWater$source) <- droplevels(redWater$source)
dim(redWater)

# 3 columns that have blank entries
sum(redWater$public_meeting == "")      # 3,334 blanks
sum(redWater$scheme_management == "")   # 3,877 blanks
sum(redWater$permit == "")              # 3,056 blanks

# how many rows have any blanks?
sum(redWater$public_meeting == "" | redWater$scheme_management == "" | redWater$permit == "")
     # 9,559 -> too many to just remove

#Merge many levels into "Other" for scheme management
# make another label "Unknown" for blanks in public_meet and permit
#
table(redWater$public_meeting)      # Not obvious -> name "Unknown"
table(redWater$scheme_management)   # None, SWC, Trust, Blanks -> Other
table(redWater$permit)              # Not obvious -> name "Unknown

library(plyr)
levels(redWater$public_meeting)[levels(redWater$public_meeting)==""] <- "Unknown"
table(redWater$public_meeting)
levels(redWater$permit)[levels(redWater$permit)==""] <- "Unknown"
table(redWater$permit)

levels(redWater$scheme_management) <- c("Other", "Company", "Other", "Other",
                                "Parastatal", "Private operator", "Other",
                                "Other", "VWC", "Water authority", "Water Board",
                                "WUA", "WUG")
table(redWater$scheme_management)

# Check out construction_year
hist(redWater$construction_year, labels = TRUE)

# huge gap in possible values -> really just bunch of 0's which are unknowns.
sum(redWater$construction_year == 0)
class(redWater$construction_year)
# makes more sense to make this a factor with ranges, so able to have "unknown"
hist(redWater$construction_year[redWater$construction_year != 0],
     breaks = seq(1960, 2020, 10), labels = TRUE, main = "construction year histogram",
```

```
     ylim = c(0, 20000))

# change to factor by decade
redWater$construction_year <- cut(redWater$construction_year,
          breaks = c(-1, 1960, 1970, 1980, 1990, 2000, 2010, 2020),
          labels = c('Unknown', "1960s", '1970s', '1980s', '1990s', '2000s', '2010s'),
          right = FALSE)

table(redWater$construction_year)

# write reduced data to file
write.csv(redWater, "../Data/ReducedWaterTraining.csv")
```

C KNN Code

```
library(onehot)
library(caret)
library(earth)

X = read.csv("Data/ReducedWaterTraining.csv")
y = read.csv("Data/waterTrainingLabels.csv")

X = as.data.frame(predict(onehot(X, max_levels = 21), X))

data_set = merge(X, y, by="id")
data_set = subset(data_set, select=-c(id, X))

set.seed(996)

random = data_set[sample(1:nrow(data_set)),]
test_data = random[1:floor(0.25 * nrow(data_set)),]
train_data = random[-(1:floor(0.25 * nrow(data_set))),]

### Basic KNN ###

start.time = Sys.time()

ctrl = trainControl(method="repeatedcv", number = 10,repeats = 3)
knnFit = train(status_group ~ ., data = train_data, method = "knn",
                               metric = "Accuracy", trControl = ctrl, tuneLength = 50)
plot(knnFit)
knnPredict = predict(knnFit,newdata = test_data)
confusionMatrix(knnPredict, test_data$status_group)

end.time = Sys.time()
time.taken = end.time - start.time
time.taken

### KNN with Feature Selection ###

X = read.csv("Data/ReducedWaterTraining.csv")
y = read.csv("Data/waterTrainingLabels.csv")

X_reduced = subset(X, select=-c(region, scheme_management, amount_tsh,
                                population, gps_height, num_private))
X_reduced_onehot = as.data.frame(predict(onehot(X_reduced, max_levels = 9),
                                X_reduced))
data_set_reduced_onehot = merge(X_reduced_onehot, y, by="id")
data_set_reduced_onehot = subset(data_set_reduced_onehot, select=-c(id, X))

marsModel_onehot <- earth(status_group ~ ., data=data_set_reduced_onehot)
```

```
ev_onehot <- evimp(marsModel_onehot)

var_imp_col = case.names(ev_onehot)
var_imp_col = gsub("`", "", var_imp_col)
colnames(data_set_reduced_onehot) = gsub(" ", "", colnames(data_set_reduced_onehot))
data_set_reduced_onehot_imp = data_set_reduced_onehot[ ,
                                which(colnames(data_set_reduced_onehot) %in% var_imp_col)]
data_set_reduced_onehot_imp$status_group = data_set_reduced$status_group

set.seed(996)

random_reduced_onehot_imp = data_set_reduced_onehot_imp[sample(1:nrow(data_set_reduced_onehot_imp)),]
test_data_reduced = random_reduced_onehot_imp[1:floor(0.25 * nrow(data_set_reduced_onehot_imp)),]
train_data_reduced = random_reduced_onehot_imp[-(1:floor(0.25 * nrow(data_set_reduced_onehot_imp))),]

start.time = Sys.time()
ctrl = trainControl(method="repeatedcv", number = 10, repeats = 3)
knnFit_reduced = train(status_group ~ ., data = train_data_reduced, method = "knn",
               metric = "Accuracy", trControl = ctrl,
               preProcess = c("center", "scale"), tuneLength = 20)
plot(knnFit_reduced)
knnPredict_reduced = predict(knnFit_reduced, newdata = test_data_reduced)
confusionMatrix(knnPredict_reduced, test_data_reduced$status_group)
end.time = Sys.time()
time.taken = end.time - start.time
time.taken

### KNN with SMOTE ###
library(DMwR)

print(prop.table(table(train_data_reduced$status_group)))

smoted_train_data <- SMOTE(status_group ~., train_data_reduced, perc.over = 200,
                                                        k = 7, perc.under = 200)

print(prop.table(table(smoted_train_data$status_group)))

start.time = Sys.time()
ctrl = trainControl(method="repeatedcv", number = 10, repeats = 3)
knnFit_smote = train(status_group ~ ., data = smoted_train_data, method = "knn",
               metric = "Accuracy", trControl = ctrl,
               preProcess = c("center", "scale"), tuneLength = 20)
plot(knnFit_smote)
knnPredict_smote = predict(knnFit_smote, newdata = test_data_reduced)
confusionMatrix(knnPredict_smote, test_data_reduced$status_group)
end.time = Sys.time()
time.taken = end.time - start.time
time.taken
```

D Logistic Code

```
# Load the packages:
library(ggplot2)
library(dplyr)
require(nnet)
#install.packages('caret')
#install.packages('e1071')
library(caret)
library(e1071)

# Load the data:
```

```
#setwd("~/1_wiliiam_and_mary/classes/DataMining/DataMining-Assignments/TeamProject")
clean_data <- read.csv("../Data/ReducedWaterTraining.csv")
clean_data <- clean_data[-1]
trainLabels <- read.csv("../Data/waterTrainingLabels.csv")
head(clean_data, 2)

#Merge the dataset with the labels set:

clean_data_1 <- merge(clean_data, trainLabels)
colnames(clean_data_1)

#Start splitting the data into training and test sets:

set.seed(4321)
randomized = sample(1:nrow(clean_data))
train_set = randomized[1:floor(length(randomized)/3)]
test_set = setdiff(randomized,train_set)
train_d = clean_data_1[train_set,]
test_d = clean_data_1[test_set,]
head(train_d, 2)
head(test_d, 2)

#Model with all variables included (excluding the id)

train_model <- multinom(status_group ~ ., data = train_d[, -1])

#Get training accuracy information:

train_pred = predict(train_model)
#summary(train_pred)
# use caret package to see the confusion matrix
train_conf = confusionMatrix(data=train_pred,reference=train_d$status_group)
train_conf
# Old: 72.36% accuracy
# New: 73.41%


# Get test set predictions and accuracy
test_pred = predict(train_model,newdata=test_d)
test_conf = confusionMatrix(data=test_pred,reference=test_d$status_group)
test_conf
# Old: 71.59% accuracy
# New: 72.68%

# Calculate training and test error rates
train_err = mean(train_pred!=train_d$status_group)
test_err = mean(test_pred!=test_d$status_group)
train_err
test_err

# Now try on the full training set:

log_model <- multinom(status_group ~ ., data = clean_data_1[, -1])

#Get accuracy information:

log_pred = predict(log_model)
#summary(train_pred)
# use caret package to see the confusion matrix
log_conf = confusionMatrix(data=log_pred, reference=clean_data_1$status_group)
log_conf
# 73.22% accuracy

# Load the test data for the competition submission
```

```
clean_test <- read.csv("../Data/ReducedWaterTest.csv")
head(clean_test,2)

#Try to get predictions for the complete Test data:
test_log_pred = predict(log_model, newdata=clean_test)
summary(test_log_pred)

#Add the predicted labels to the test data set

clean_test$status_group <- test_log_pred
head(clean_test, 2)

# write new label data to file
write.csv(clean_test, "../Data/Label_Log_1.csv")

# Got score of 0.7143 for the competition submission.
```

E Random Forest Code

```
library(randomForest)
library(caret)
train_values <- read.csv("../Data/ReducedWaterTraining.csv")
labels <- read.csv("../Data/waterTrainingLabels.csv")
train_values <- train_values[-1]
data_set <- merge(labels, train_values)
train_size <- floor(0.8* nrow(data_set))
train_index <- sample(seq_len(nrow(data_set)), size = train_size)
#Splitting the data set into training and test set
train_set <- data_set[train_index,]
test_set <- data_set[-train_index,]
set.seed(42)
model_forest <- randomForest(as.factor(status_group) ~ ., data = train_set[-1],
                             importance = TRUE, ntree = 100, nodesize = 1, mtry =5)
pred_forest_train <- predict(model_forest, train_set[-1])
pred_forest_test <- predict(model_forest, newdata = test_set[-1])
confusionMatrix(pred_forest_test, test_set$status_group)$overall['Accuracy']
varImpPlot(model_forest)

test_values <- read.csv("../Data/ReducedWaterTest.csv")
test_values <- test_values[-1]
View(test_values)
colnames(test_values)
colnames(data_set)
pred_forest_test_r <- predict(model_forest, newdata = test_values[-1])

#Testing different values for ntree

control <- trainControl(method = 'repeatedcv',
                        number = 10,
                        repeats = 3,
                        search = 'grid',
                        p = 0.8)
#create tunegrid
tunegrid <- expand.grid(.mtry = 5)
modellist <- list()
#train with different ntree parameters
for (ntree in c(10,20,50,100)){
  set.seed(123)
  fit <- train(as.factor(status_group)~.,
               data = data_set,
               method = 'rf',
```

```
              metric = 'Accuracy',
              tuneGrid = tunegrid,
              trControl = control,
              ntree = ntree)
  key <- toString(ntree)
  print(fit)
  modellist[[key]] <- fit
}
#Compare results
results <- resamples(modellist)
summary(results)
```

F Oversampling Code

```
library(randomForest)
library(caret)
set.seed(42)

train_values <- read.csv("../Data/ReducedWaterTraining.csv")
labels <- read.csv("../Data/waterTrainingLabels.csv")
train_values <- train_values[-1]
data_set <- merge(labels, train_values)

needRepInds <- which(data_set$status_group == 'functional needs repair')
otherInds <- which(data_set$status_group != 'functional needs repair')

train_size_needRep <- floor(1.5 * length(needRepInds))
train_size_others <- floor(0.7 * nrow(data_set[-needRepInds,]))

train_index_needRep <- sample(needRepInds, size = train_size_needRep, replace = TRUE)
train_index_others <- sample(otherInds, size = train_size_others)

train_index = c(train_index_needRep, train_index_others)

#Splitting the data set into training and test set
train_set <- data_set[train_index,]
test_set <- data_set[-train_index,]

# checking to make sure no train set in test set
test_check <- row.names(test_set)
train_check <- row.names(train_set)
for (testRowName in test_check) {
  if (testRowName %in% train_check) {
    print("NOOOO")
    print(testRowName)
  }
}

model_forest <- randomForest(as.factor(status_group) ~ ., data = train_set[-1],
                             importance = TRUE, ntree = 100, nodesize = 1, mtry =5)
pred_forest_train <- predict(model_forest, train_set[-1])
pred_forest_test <- predict(model_forest, newdata = test_set[-1])
confusionMatrix(pred_forest_test, test_set$status_group)$overall['Accuracy']

varImpPlot(model_forest)

confusionMatrix(pred_forest_test, test_set$status_group)

test_values <- read.csv("../Data/ReducedWaterTest.csv")
test_values <- test_values[-1]
View(test_values)
```

```
colnames(test_values)
colnames(data_set)
pred_forest_test_r <- predict(model_forest, newdata = test_values[-1])

control <- trainControl(method = 'repeatedcv',
                        number = 10,
                        repeats = 3,
                        search = 'grid',
                        p = 0.8)
#create tunegrid
tunegrid <- expand.grid(.mtry = 5)
modellist <- list()

#train with different ntree parameters
for (ntree in c(10,20,50,100)){
  set.seed(123)
  fit <- train(as.factor(status_group)~.,
               data = data_set,
               method = 'rf',
               metric = 'Accuracy',
               tuneGrid = tunegrid,
               trControl = control,
               ntree = ntree)
  key <- toString(ntree)
  print(fit)
  modellist[[key]] <- fit
}

#Compare results
results <- resamples(modellist)
summary(results)
```

G Comparison Plots Code

```
# plot comparison plots for all models
library(reshape2)
# list with confusion matrices for models.
# predicted are rows, actual are cols
#  -> Var1: predicted, Var2: actual
classNames <- c('functional', 'functional needs repair', 'non functional')

confMats <- list(KNN = data.frame(melt(matrix(c(6904, 580, 1455, 251, 328, 129, 878, 176, 4149),
                              nrow = 3, byrow = TRUE,
                              dimnames = list(classNames, classNames)))),
                 Log = data.frame(melt(matrix(c(19158, 2263, 5646, 126, 131, 83, 2183, 516, 9494),
                              nrow = 3, byrow = TRUE,
                              dimnames = list(classNames, classNames)))),
                 RandForest = data.frame(melt(matrix(c(5809, 473, 1034, 150, 265, 78, 495, 115, 3461),
                                     nrow = 3, byrow = TRUE,
                                     dimnames = list(classNames, classNames)))),
                 OverSamp = data.frame(melt(matrix(c(8370, 383, 1453, 539, 446, 196, 770, 118, 5197),
                                                   nrow = 3, byrow = TRUE,
                                                   dimnames = list(classNames, classNames)))))

confMats$KNN$rescale <- confMats$KNN$value /
  c(rep(6904 + 251 + 878, 3), rep(580 + 328 + 176, 3), rep(1455 + 129 + 4149, 3))
confMats$Log$rescale <- confMats$Log$value /
  c(rep(19158 + 126 + 2183, 3), rep(2263 + 131 + 516, 3), rep(5646 + 83 + 9494, 3))
confMats$RandForest$rescale <- confMats$RandForest$value /
  c(rep(5809 + 150 + 495, 3), rep(473 + 265 + 115, 3), rep(1034 + 78 + 3461, 3))
confMats$OverSamp$rescale <- confMats$OverSamp$value /
```

```
  c(rep(8370 + 539 + 770, 3), rep(383 + 446 + 118, 3), rep(1453 + 196 + 5197, 3))

colnames(confMats$KNN) <- c('Predicted', 'Actual', 'value', 'Percent')
colnames(confMats$Log) <- c('Predicted', 'Actual', 'value', 'Percent')
colnames(confMats$RandForest) <- c('Predicted', 'Actual', 'value', 'Percent')
colnames(confMats$OverSamp) <- c('Predicted', 'Actual', 'value', 'Percent')

# array of accuracy scores for models.
accs <- c(0.7664, 0.7268, 0.8026, 0.802)   # KNN, Log, then RandForest


library(ggplot2)
plotConf <- function(data, title) {
  print(ggplot(data, aes(Actual, Predicted)) +
    geom_tile(aes(fill = Percent), colour = 'white') +
    geom_text(aes(label = paste(round(Percent * 100, 1), '%', sep = ""))) +
    ylim(rev(levels(data$Predicted))) +
    scale_fill_gradient(low = "white", high = "steelblue") + ggtitle(title))
}
#setwd('../Paper/Figures/')
plotConf(confMats$KNN, title = "KNN Classification")
#ggsave('KNNConfMatrix.png')
plotConf(confMats$Log, title = "Logistic Regression")
#ggsave('LogConfMatrix.png')
plotConf(confMats$RandForest, title = "Random Forest")
#ggsave('RandForestConfMatrix.png')
plotConf(confMats$OverSamp, title = "Random Forest with Over-Sampling")
#ggsave('OverSampleConfMatrix.png')


plotBars <- function(data, title) {
  print(ggplot(data, aes(Predicted, Percent * 100)) +
          geom_bar(aes(fill = Model), position = 'dodge', stat = 'identity') +
          ylab('Percent') + ggtitle(title))
}

library(data.table)
allData <- data.frame(rbindlist(confMats))

allData$Model <- c(rep('KNN', 9),
                   rep('Logistic', 9),
                   rep('Random Forest', 9),
                   rep('Over Sampling', 9))

functional <- allData[allData$Actual == 'functional',
                      c('Model', 'Predicted', 'Percent')]
broken <- allData[allData$Actual == 'functional needs repair',
                  c('Model', 'Predicted', 'Percent')]
replace <- allData[allData$Actual == 'non functional',
                   c('Model', 'Predicted', 'Percent')]

plotBars(functional, 'Actual Class: Functional')
#ggsave('FunctionalBarComps.png')
plotBars(broken, 'Actual Class: Functional Needs Repair')
#ggsave('BrokenBarComps.png')
plotBars(replace, 'Actual Class: Non Functional')
#ggsave('ReplaceBarComps.png')


plotBars2 <- function(data, title) {
  print(ggplot(data, aes(Model, Percent * 100)) +
          geom_bar(aes(fill = Predicted), position = 'dodge', stat = 'identity') +
          ylab('Percent') + ggtitle(title) + xlab("Class") + ylim(c(0,100)))
}
```

```
truePreds <- allData[allData$Predicted == allData$Actual,]
plotBars2(truePreds, 'Percent Accuracy By Model')


# water <- read.csv('../../Data/ReducedWaterTraining.csv')
# library(gridExtra)
# p1 <- ggplot(water[water$population < 2000,],
#        aes(x = population)) + geom_histogram(bins = 20) + ylim(c(0, 34000))
# #ggsave('PopulationHist.png')
# p2 <- ggplot(water, aes(x = gps_height)) +
#   geom_histogram(bins = 20) + ylim(c(0, 34000))
# #ggsave('gps_heightHist.png')
# p3 <- ggplot(water,
#        aes(x = longitude)) + geom_histogram(bins = 20) + ylim(c(0, 34000))
# #ggsave('longitudeHist.png')
# p4 <- ggplot(water,
#        aes(x = latitude)) + geom_histogram(bins = 20) + ylim(c(0, 34000))
# #ggsave('latitudeHist.png')
# grid.arrange(p1, p2, p3, p4, ncol = 2, nrow = 2)
# ggsave('HistsNotNormal.png')
```