Assignment 3: CMTH642

Christopher Graham

November 27, 2015

Overview

This assignment develops a model to predict a subjective quality rating on Portuguese red vinho verde wines. The results we generate are compared with results obtained by the original paper to work with the data.

We instigated three main changes over the original paper's approach:

1. approaching this as a classification, rather than regression problem
2. applying an ensemble learning model to the data
3. using an undersampling/SMOTE oversampling algorithm to deal with class imbalance in the data.

Our ensemble model did not improve on the original paper's SVM approach, probably due to a poor choice of component models. However, one of the constituents of our ensemble model - random forests - showed markedly better performance.

Import and Split Data

The data is provided at https://archive.ics.uci.edu/ml/datasets/Wine+Quality and was originally used for the paper "Modeling wine preferences by data mining from physicochemical properties" available at http://repositorium.sdum.uminho.pt/bitstream/1822/10029/1/wine5.pdf

The data set provides provides 11 chemical measurements of Portuguese red wines, and one quality score assigned by human testers. The data is complete and clean. Pre-processing is described in the original paper.

```
# load all libraries required in analysis
require(caret)
require(corrplot)
require(FNN)

# load functions that are used later in analysis
source('ensemble_eval.R')
source('over_under_wrapper.R')
source('find_minor_class_f_stat.R')

red <- read.csv('winequality-red.csv', sep = ';')
# Treating as classification problem so convert quality to factor variable
red$quality <- as.factor(red$quality)

# Divide off a testing set for final validation
# All work will be done only based on the training set
set.seed(25678)
# Note that createDataPartition attempts to balance classes
train_idx <- createDataPartition(y=red$quality, p=0.8, list = FALSE)
training <- red[train_idx,]
testing <- red[-train_idx,]
```

Check Data Characteristics

Completeness

```
if (sum(complete.cases(testing)) == nrow(testing)) {
    print('All complete cases')
}
```

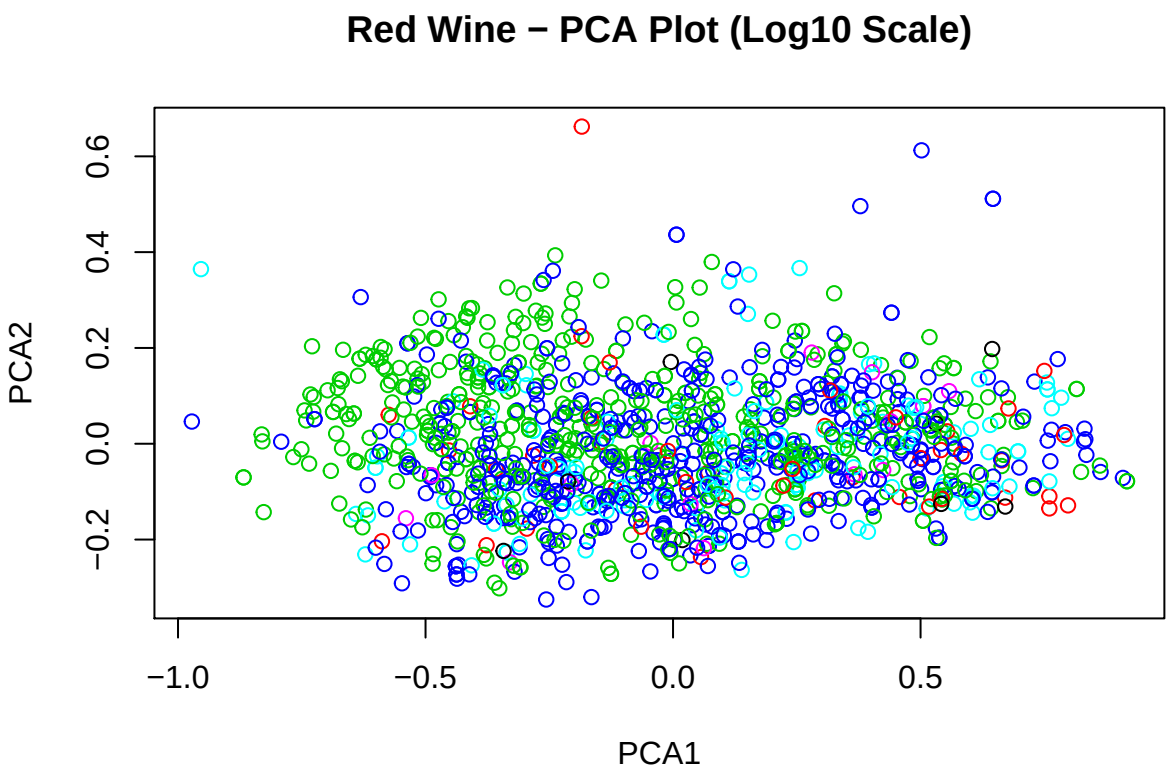
```
## [1] "All complete cases"
```

The data is complete in all variables, and as such there is no need to impute values.

2-D Visualization

Applying Principal Component Analysis to the data set, we can capture over 99% of the variance with 2 Principal Components, giving us a pretty good visualization of the data:

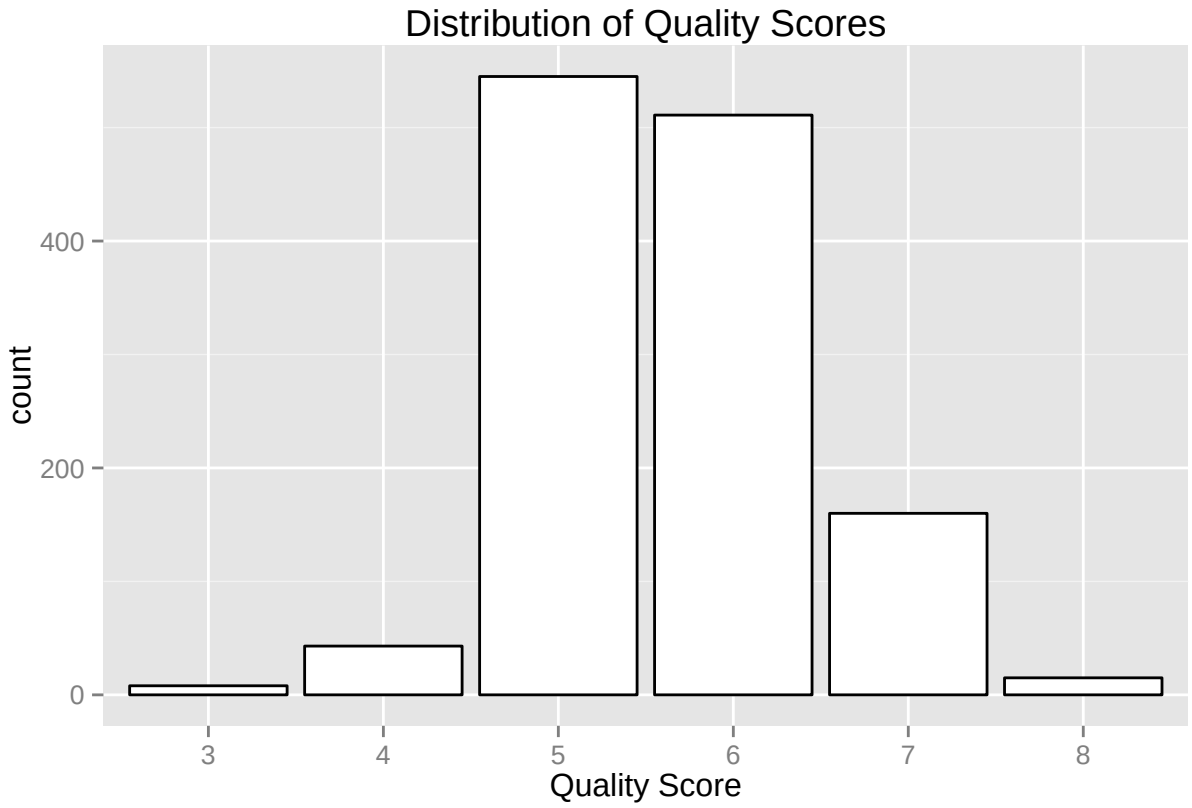
```
pc_comp <- prcomp(log10(training[,-12]+1))
plot(pc_comp$x[,1], pc_comp$x[,2],
     col = training$quality,
     main = 'Red Wine - PCA Plot (Log10 Scale)',
     xlab = 'PCA1',
     ylab = 'PCA2')
```



Essentially, it's a mess, with a great deal of overlap between classes and no readily apparent trends. There seems to be a bit of a division between some of the major classes, but nothing particularly clear. We'll see if we can make sense of it with some machine learning.

Class Balance

There is a significant concern with the distribution of quality scores in the data set. Specifically, over 80% of the wines have average ratings (5 or 6), and very few wines get ratings at the more extreme ends of the rating spectrum. (3 and 8 are the extreme values awarded, even though wines were rated on a 10-point scale). Class imbalance has been shown to be a significant problem in building an effective Machine Learning model. We provide a strategy for dealing with this below, but first will look at some other characteristics of the data.

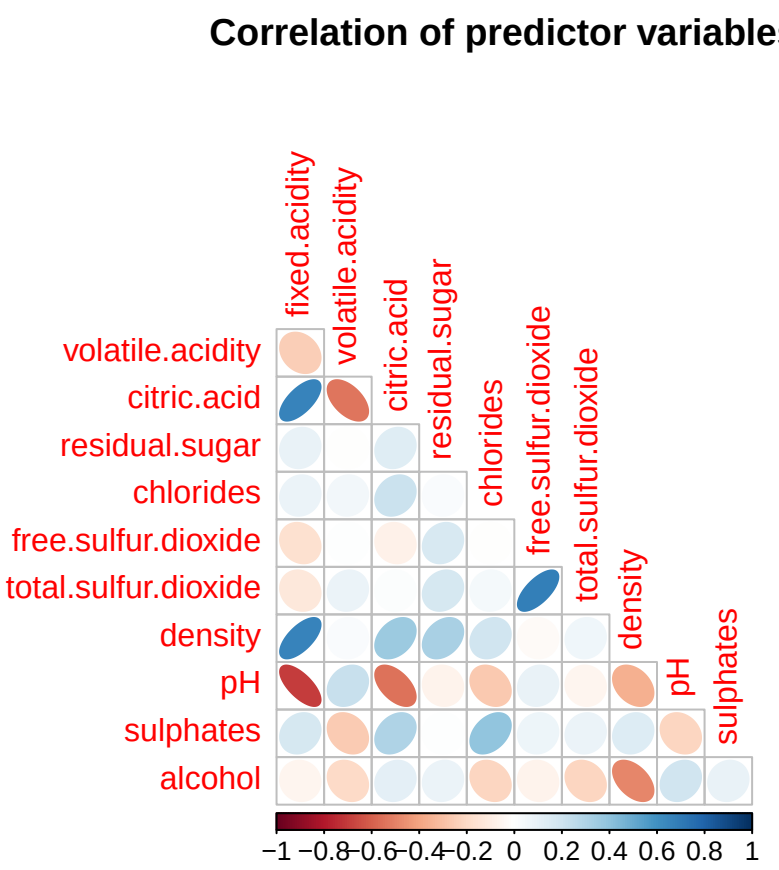


Variable Relationships

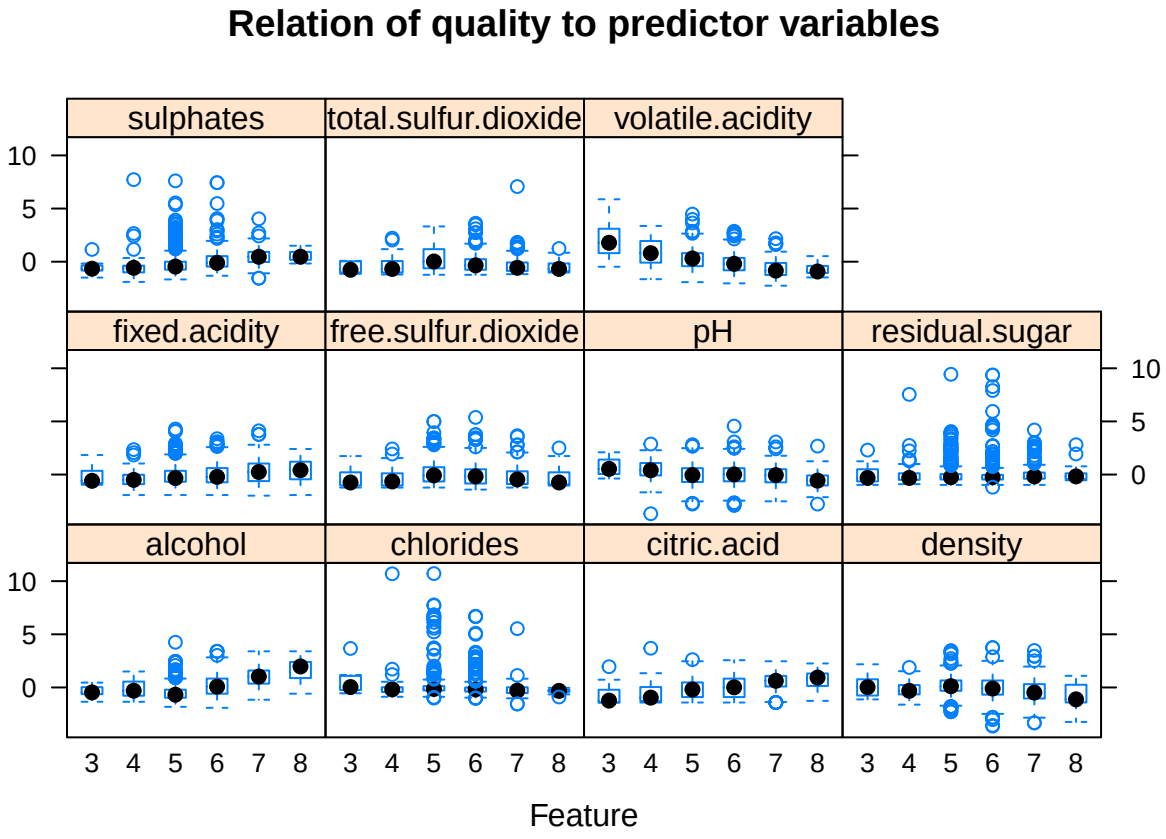
```
# Check for variables with near zero variability
nearZeroVar(training, saveMetrics = TRUE)
```

```
##                      freqRatio percentUnique zeroVar   nzv
## fixed.acidity         1.086957     7.2542902   FALSE FALSE
## volatile.acidity      1.055556    10.6864275   FALSE FALSE
## citric.acid           1.824561     6.2402496   FALSE FALSE
## residual.sugar        1.070796     6.6302652   FALSE FALSE
## chlorides             1.255814    10.9984399   FALSE FALSE
## free.sulfur.dioxide   1.337209     4.6021841   FALSE FALSE
## total.sulfur.dioxide  1.166667    10.9984399   FALSE FALSE
## density               1.033333    30.9672387   FALSE FALSE
## pH                    1.119048     6.7082683   FALSE FALSE
## sulphates             1.000000     7.3322933   FALSE FALSE
## alcohol               1.425000     4.6801872   FALSE FALSE
## quality               1.066536     0.4680187   FALSE FALSE
```

```
# look at correlation between predictor variables
corrplot(cor(training[-12]), method = 'ellipse', type = 'lower',
         title = 'Correlation of predictor variables', diag=FALSE,
         mar=c(0,0,1,0))
```




```
# Compare relationsip of quality to predictor variables
# first normalize the variables so we can compare in one graph
norm_obj <- preProcess(training[,-12], method=c('center', 'scale'))
train_norm <- predict(norm_obj, training)
featurePlot(x=train_norm[,-12], y=train_norm$quality, plot='box',
            main = 'Relation of quality to predictor variables')
```



Summary of Exploratory Data Analysis

What this preliminary exploration tells us is:

- None of the variables show near zero variability, meaning there is no *prima facie* reason to exclude any at the start of our analysis
- For the most part, there does not seem to be a huge amount of collinearity between variables. There are some instances of collinearity, where it might be expected (citric.acid, fixed.acidity and pH), (free.sulfur.dioxide and total.sulfur.dioxide). So there may be some room to reduce dimensionality. However, we will not be doing that in this particular analysis.
- There are some clear relationships between some of the predictor variables and wine quality. The variables volatile.acidity, alcohol and density seem particularly important.

Strategies for dealing with Class Imbalance in Provided Data

Since one of the goals for a prediction system would be to help vintners identify wines that are likely to be either very good or very bad, the lack of wine specimens with extreme scores could be problematic for our model development.

Dealing with class imbalance is a common problem in machine learning. The standard approach in this situation is a combination of under-sampling the majority class and over-sampling the minority class to create a distribution that will help to develop an adequate predictive model. However, there is a persistent concern about the extent to which this under & oversampling should occur, and the specific techniques to implement.

While there are different pre-made solutions to this problem, many of them are designed around a binary response variable. Specifically, the R package 'unbalanced' provides a number of pre-made approaches to dealing with an unbalanced data set. But, unfortunately, it requires a binary response variable.

In this case, we are approaching this as a multiple-level classification, relying on the classes provided in the original data (reasons for this are discussed below).

I have identified two different approaches to effectively determining an over-/ under-sampling approach for multi-class response variables. The first is the wrapper approach proposed by Chawla et al[1], and the resampling ensemble algorithm proposed by Qian et al[2].

Both of these approaches rely on a combination of SMOTE oversampling and random undersampling. However, the Chawla paper offers a more coherent (and KDD-Cup proven!) approach to determining sample levels, and so we will use that approach here.

Note, however that the algorithms used in Chawla et al require us to identify a classifier algorithm that will be used to determine stop points for sampling.

So, we need to turn briefly to model selection.

Model Selection

Regression vs. Classification

As noted on the source page for this data set, the data lends itself to either a regression or classification approach. The original paper to use this data set adopted a regression approach to the data, arguing that regression allows us to better evaluate "near miss" predictions (e.g. if the true value is 3, we can up-score the model if it predicts 4, rather than 7). The paper used both Neural Network (NN) and Support Vector Machine (SVM) models, and obtained an accuracy rate of 64% with a 0.5 error tolerance in quality prediction, and 89% accuracy with a 1.0 error tolerance.[3]

Furthermore, given the somewhat arbitrary way in which quality scores were derived (the median value from 3 or more judges), it's safe to say that quality scores are far from reliable values, but rather near approximations. As such, regression probably makes most sense.

But Assignment 1 was based around regression, and I'd like to make this one a bit more challenging, so I'm going to frame this assignment as a classification problem. This means the fairest comparison for our results is with the 0.5 tolerance level in the original paper – a difference of less than 0.5 would round to the correct value.

Performance Criteria

In order to ensure easy comparability with the original paper, and for simplicity of understanding, our primary judge of performance will be overall classification accuracy.

However, overall accuracy does not indicate how effective our model is at identifying minority class examples.

Note that because we are implementing the Chawla et al algorithm, our over/under sampling function will evaluate performance based on the model's f-statistic (also known as F1 score). The closer this is to 1, the better the performance:

$$f - measure = \frac{2 * precision * recall}{recall + precision}$$

[1] Chawla et al "Automatically countering imbalance..."

[2] Qian et al. "A resampling ensemble algorithm..."

[3] Cortez P, Cerdeira A, Almeida F, Matos T, Reis J. Modeling wine preferences by data mining from physicochemical properties. Decision Support Systems. 2009;47(4):547-53.

The f-statistic is a good overall performance measure, balancing precision and recall. We will therefore look at the f-stat for non-average-rated wines. This will give us a better understanding of how well our model identifies bad or excellent wines **only**. For the purposes of this analysis, we define non-average-rated wines as having a quality score of 3, 4, 7 or 8.

In order to minimize the impact of the relative overweight in class 7, we will be using the mean of the f-stats for these four classes, rather than calculating an overall f-stat for all non-average-rated classes combined.

The code for the function we will have written to calculate this is:

```
## function (conf_mtx)
## {
##     f_stat <- NULL
##     for (i in c(1, 2, 5, 6)) {
##         newf <- (2 * conf_mtx$byClass[i, 1] * conf_mtx$byClass[i,
##             3])/(conf_mtx$byClass[i, 1] + conf_mtx$byClass[i,
##             3])
##         if (is.nan(newf)) {
##             newf <- 0
##         }
##         f_stat <- c(f_stat, newf)
##     }
##     return(mean(f_stat))
## }
```

Model Selection

Given the relatively weak predictive accuracy (at T=0.5) for the models in the first paper, it appears that a single model may not be all that great at classification. In these instances, one of the more interesting approaches is *ensemble learning* - essentially combining predictions from multiple models into one super-model.[4]

In terms of specific ensemble learning implementation, we are going to use the approach suggested by Jeff Leek of John's Hopkins University in the Coursera online course "Practical Machine Learning"[5].

In order to build an ensemble model, we first need to build a number of individual models to combine into the final approach. In an attempt to bring together all we've done in the course (plus a little bit more!), we're going to use:[6]

- multinomial logistic regression (multinom or mn)
- Naïve Bayes (nb)
- Random Forest (rf)
- Boosting; specifically boosting with trees (gbm)
- Linear Discriminant Analysis (lda) - in case‘ there are any interesting differences with Naive Bayes if we don't assume variable independence

We are specifically excluding the two models of the original paper (NN and SVM) to see if we can do better without the models used there.

[4] Numerous articles have shown this, but see specifically: Rokach L. Ensemble-based classifiers. Artificial Intelligence Review. 2010;33(1):1-39.

[5] Course notes available at: http://sux13.github.io/DataScienceSpCourseNotes/8_PREDMACHLEARN/Practical_Machine_Learning_Course_Notes.pdf, starting on p. 67

[6] Although we didn't go over all of these in class, I am confident that I have a good understanding of them, and would be happy to discuss in further detail if you have questions as to that end.

Also, because the SMOTE oversampling algorithm relies on a knn methodology, we will not be including knn in the base models in order to avoid potentially introducing erroneous re-classification.

Finally, note that for the final model, the model we will use is a simple plurality vote among the model responses (the modal value).

The code for the specific model being run is in Appendix A.

k-fold Cross-validation

Note that our model evaluation is set using 10 repetitions of 10-fold cross-validation. This is set through the trainControl function in the R's caret library.

Normalizing Data

As per the original paper, we will be evaluating the feed-in models based on standardized data. This is done through the preProcess function in R's caret library.

Dimensionality Reduction

As noted earlier, we can use PCA to simplify the data set to 2 principal components and with those, capture 99% of the variance in the original data set. I have not actually implemented this, although it would likely simplify and speed up some of the computation that follows. It is worth considering, especially if we attempt to apply this to the white wine data set which is much larger.

Implementation of Ensemble Model

Model baseline testing

As a baseline, before dealing with the class imbalance, let's see how our model performs:

```
# First, get the predictions for the training set
base_results <- ensemble_eval(training)
```

```
# confusion matrix for predictions on training data
cm <- confusionMatrix(base_results[[1]], training$quality)
cm$table
```

```
##            Reference
## Prediction   3   4   5   6   7   8
##          3   4   0   1   0   0   0
##          4   0  15   1   1   0   0
##          5   3  16 436 126   3   0
##          6   1  11 102 357  55   5
##          7   0   1   5  27 102   4
##          8   0   0   0   0   0   6
```

```
cm$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   7.176287e-01   5.497836e-01   6.921176e-01   7.421378e-01   4.251170e-01
## AccuracyPValue  McnemarPValue
##   1.297381e-99            NaN
```

The model gives us a pretty good result overall, with a predicted 71% accuracy on the training data, better than the 64% accuracy (at a 0.5 threshold) from the original paper.

The f-stats for the minority classes (3,4,7 and 8), however, are noticeably better for our base model than for the original paper:[7]

```
# Average non-average-quality f-stat for our model
f_stat <- find_minor_f(cm)

# Average non-average-quality f-stat for original paper
paper_f <- mean(c(0, 0, (2*(1/53)*(1/5))/(1/53+1/5),
                  (2*(82/199)*(82/140))/(82/199+82/140)))

print(paste('base model f-stat:', f_stat))
```

```
## [1] "base model f-stat: 0.592271858576207"
```

```
print(paste('original paper f-stat', paper_f))
```

```
## [1] "original paper f-stat 0.129564642457532"
```

Baseline against test set

As a test, we'll apply this model to the test data and see if our results are legitimate or due to overfitting.

```
base_test <- ensemble_eval(training, testing)
```

```
cm2 <- confusionMatrix(base_test$final, testing$quality)
cm2$table
```

```
##           Reference
## Prediction   3   4   5   6   7   8
##          3   0   0   1   0   0   0
##          4   0   0   0   0   0   0
##          5   1   8 103  38   4   0
##          6   1   2  31  82  20   2
##          7   0   0   1   7  15   1
##          8   0   0   0   0   0   0
```

```
cm2$overall
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   6.309148e-01   3.927899e-01   5.751893e-01   6.841686e-01   4.290221e-01
## AccuracyPValue  McnemarPValue
##   3.848496e-13            NaN
```

[7] Note that because the original paper predicted 0 true positives for class 3 and class 8, the f-stat for each of those is technically NaN. I have substituted with zero to compare.

```
# Average non-average-quality f-stat for our model
test_f <- find_minor_f(cm2)
test_f
```

```
## [1] 0.1190476
```

When we apply our model to the testing set, we see a relatively significant degradation in performance - essentially to the level of the original paper, likely indicating some overfitting in our model.

Importantly, our model is doing a very poor job at predicting minority classes in the test set – especially quality ratings of 3, 4, and 8, which have the weakest representation in our data.

So, in an effort to improve on this, we need to turn to our function for undersampling and SMOTE oversampling.

Ensemble model vs. component models

It's also worth looking at how well our ensemble model is working in comparison to each of the individual component parts. As noted earlier, in theory, we should be getting better results.

```
cm_mn <- confusionMatrix(base_test$mn, testing$quality)
cm_nb <- confusionMatrix(base_test$nb, testing$quality)
cm_rf <- confusionMatrix(base_test$rf, testing$quality)
cm_gbf <- confusionMatrix(base_test$gbm, testing$quality)
cm_lda <- confusionMatrix(base_test$lda, testing$quality)

# Multinomial
cm_mn$overall[1]; find_minor_f(cm_mn)
```

```
##  Accuracy
## 0.6088328
```

```
## [1] 0.1052632
```

```
# Naive Bayes
cm_nb$overall[1]; find_minor_f(cm_nb)
```

```
##  Accuracy
## 0.6025237
```

```
## [1] 0.147096
```

```
# Random Forest
cm_rf$overall[1]; find_minor_f(cm_rf)
```

```
##  Accuracy
## 0.7160883
```

```
## [1] 0.1532258
```

```
# Boosting
cm_gbf$overall[1]; find_minor_f(cm_gbf)
```

```
##  Accuracy
## 0.6340694
```

```
## [1] 0.1426426
```

```
# Linear Discriminant Analysis
cm_lda$overall[1]; find_minor_f(cm_lda)
```

```
##  Accuracy
## 0.6119874
```

```
## [1] 0.1268657
```

This shows some interesting results. Namely that the random forest model seems head and tails above the other ones - including our ensemble model - in both overall prediction accuracy, and in the f-score it returns for the minority classes. The f-score itself isn't eye-poppingly better than either Naive Bayes or Boosting, but the combination of overall accuracy and f-score makes this modelling approach stand out as perhaps the best way of analyzing this data.

Also, the ensemble approach does not appear to be working that well in comparison to its constituent models. The ensemble's overall accuracy is about the mean of the constituent models, and it's minority class f-score is actually lower than the average of its constituents.

At this point, it would probably be most logical to drop multinomial regression and Linear Discriminant Analysis from our approach, as neither shows excellence on either performance metric we have adopted. But in a spirit of inquiry, let's forge ahead and see what happens to all the models once we reshape the data.

Undersampling/SMOTE oversampling implementation

As discussed earlier, I have decided to implement the "Wrapper Undersample SMOTE Algorithm" developed by Chawla et al as a way to re-balance the provided data.

Simplified, the algorithm works as follows:

1) Establish a baseline performance level for each majority and minority class
2) For each majority class, undersample by 10% until either minority class predictions stop improving, or until majority class predictions degrade by 5%. There is no guarantee that the majority class will in fact be undersampled.
3) For each minority class, SMOTE oversample by 100%, and continue this until that minority class performance is increased by less than 5% three times. This guarantees at least a 800% oversample rate for each class.[8]

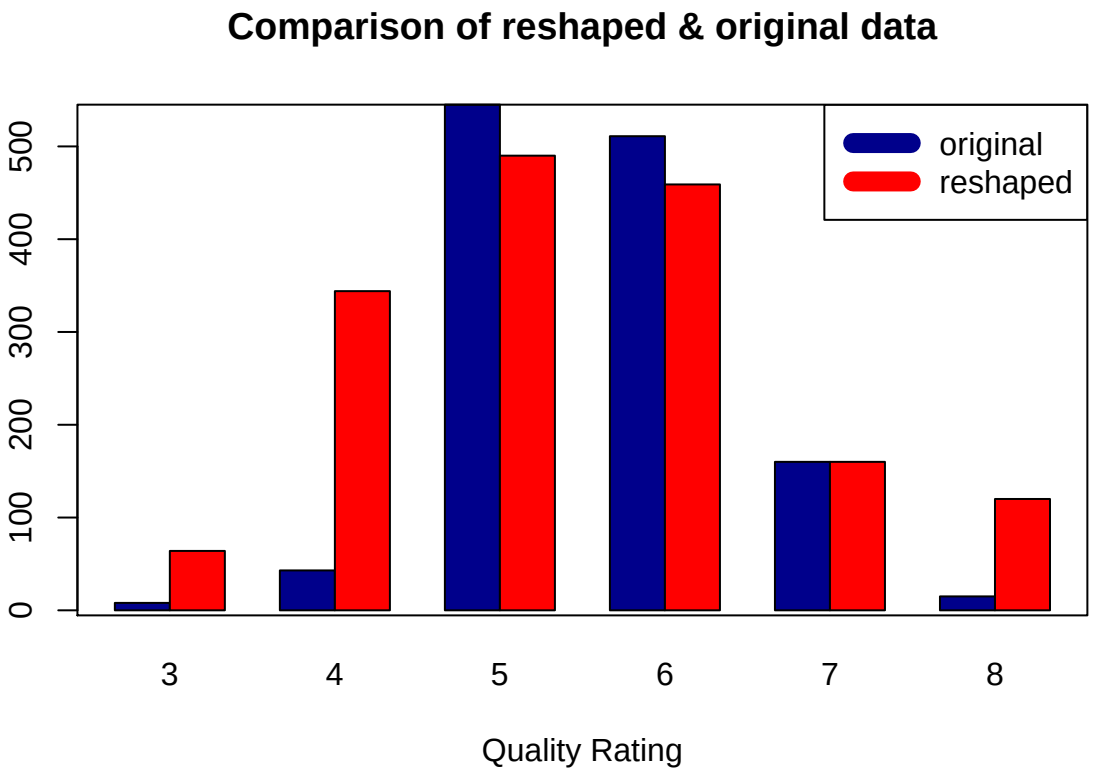
The specific R code for my implementation is shown in Appendix 2. The algorithm returns a reshaped data set that will be used to generate our final predictive model.[9]

[8]The first oversample is based on the original data, giving us a class size double the original. The next oversample is based on the oversampled *and* original data, giving us results that are 4x the original data, etc.

[9]Because I don't have a great grasp of how R memory allocation works, I was unable to run this through knitr. My laptop ran out of memory in executing the algorithm. Instead I ran the code through R's console with occasional memory purges, saved the final data set to disk and loaded it into the RMD file for further manipulation.

```
# algorithm needs a named vector of class categories
#       with -1 = minority; 1 = majority; 0 = neutral
class_list <- c(-1,-1,1,1,0,-1)
names(class_list) <- c(3,4,5,6,7,8)
new_training <- wrapper(class_list = class_list, train_set = training,
                        eval_fun = ensemble_eval, seed=TRUE)
```

First, let's see how our new distribution compares to the original training data:



As you can see, the majority classes decreased by a small amount, while the majority classes have seen a significant increase in representation. Class 4 now approximates the majority classes. Class 3 and 8 are still relatively underrepresented.

But more importantly, let's see if our reshaped data gives us a better model. We will look at how the model works for our testing data set.

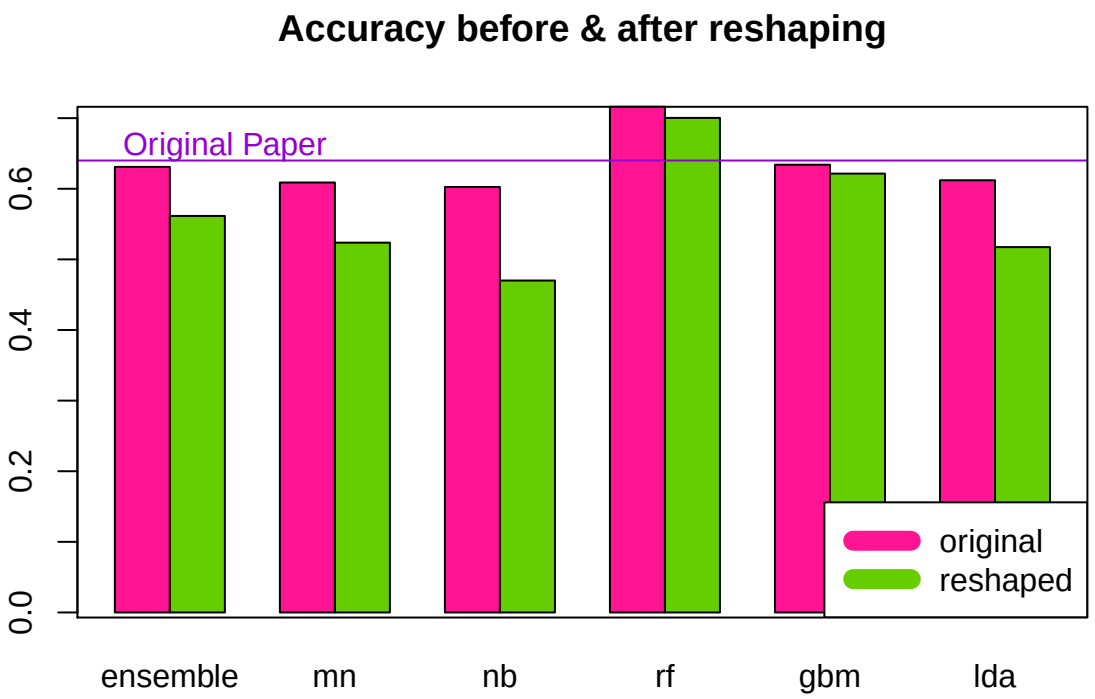
```
new_test_results <- ensemble_eval(new_training, testing)
```

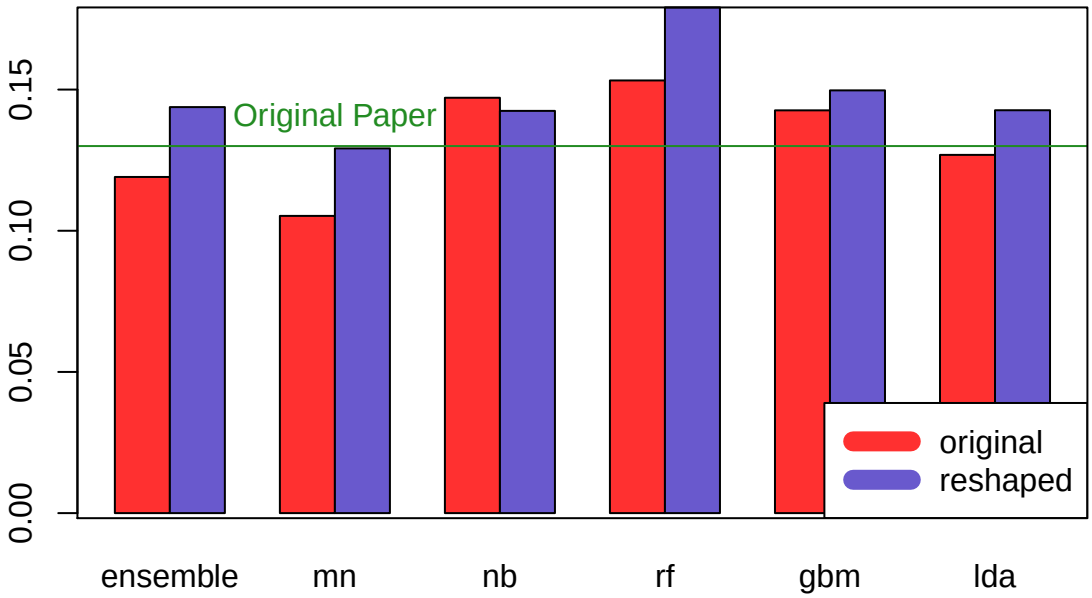
```
##            Reference
## Prediction  3  4  5  6  7  8
##          3  0  0  0  0  0  0
##          4  1  2 16 10  2  0
##          5  0  6 91 33  2  0
##          6  1  2 27 69 16  1
##          7  0  0  1  9 16  2
##          8  0  0  1  6  3  0
```

```
##       Accuracy          Kappa  AccuracyLower  AccuracyUpper   AccuracyNull
##   5.615142e-01   3.361307e-01   5.049543e-01   6.169138e-01   4.290221e-01
```

```
## AccuracyPValue  McnemarPValue
##   1.439978e-06            NaN
```

```
##                      model  accuracy    f_stat
##                      paper 0.6400000 0.1295646
##             base_ensemble 0.6309148 0.1190476
##          reshaped_ensemble 0.5615142 0.1437932
##              base_multinom 0.6088328 0.1052632
##          reshaped_multinom 0.5236593 0.1291667
##           base_naive_bayes 0.6025237 0.1470960
##       reshaped_naive_bayes 0.4700315 0.1424559
##         base_random_forest 0.7160883 0.1532258
##     reshaped_random_forest 0.7003155 0.1790909
##              base_boosting 0.6340694 0.1426426
##          reshaped_boosting 0.6214511 0.1497131
##  base_linear_discriminant 0.6119874 0.1268657
##   reshaped_linear_discrim 0.5173502 0.1426761
```





With a model based on our reshaped data, we are seeing a decrease in overall accuracy for both our ensemble model, and each of the individual components of that model. LDA and Naive Bayes both show rather drastic drop-offs in accuracy, perhaps reinforcing our earlier observation that these models should have been dropped after our initial analysis.

On the flip side, we're seeing that, other than with the Naive Bayes model, the minority class f-score average improves with the resampled data.

This is pretty much what we'd expect to see, as the undersampling causes some accuracy loss with the majority classes. Similarly, minority class performance improves as oversampling provides increased information which our model can use to identify characteristics of these minority classes.

Comments on Naive Bayes Weakness

One of the more surprising results of this work (at least to me) was the severe degradation of the Naive Bayes Model's accuracy as a result of our class rebalancing.

On reflection, however, this isn't all that shocking. Naive Bayes' main strengths as a classifier are that it is fairly light-weight, with good accuracy, and that it doesn't need tons of data to work well.[10] This compares rather starkly with the Random Forest approach which is quite resource-intensive to build, and that readily improves with additional data points.

In other words, class imbalance seems to be less of a problem with the probabilistic methodologies that underpin both Naive Bayes and LDA. As such, if a light-weight, average performance model is acceptable, Naive Bayes is probably a good choice, especially in that class rebalancing does not appear to be particularly necessary, or even helpful.

[10] See, for example Forman and Cohen, p12, who note that "Naive Bayes models are often relatively insensitive to a shift in training distribution"

Comparison of Results to Original Paper and Conclusions

Differences in Approach

As noted above, there are three main differences between our approach and that taken in the original paper:

1. We chose to apply a classification, rather than regression approach
2. We relied on an ensemble learning approach, rather than a single model.
3. We transformed the original data through over/under sampling.

Ensemble model vs. SVM (Original model)

The ensemble approach we used actually returned slightly weaker accuracy than the original paper's SVM approach. In addition, the average f-statistic for minority classes was weaker in our model, when compared with the SVM approach in the original paper.

Part of the reason for this weaker performance was the relative weakness of the probabilistic approaches of Naive Bayes and LDA, and the relative overweight that these models had in contributing to the final evaluation.

A better practice to building the ensemble model would be to first evaluate the predictive effectiveness of a number of different approaches, and then combine the most effective approaches into a final model.

Impact of class rebalancing

The impact of the over/under sampling algorithm we introduced was interesting. It reduced the overall accuracy of the ensemble model by about 10%, while boosting the average f-stat for minority classes by about 20%. Overall predictive ability for minority classes was still rather weak even after this transformation.

With our random forest model, on the other hand, the gains in minority class predictive quality were a bit more pronounced, with our f-stat rising to 0.18, with a very small drop-off in overall predictive ability. Still, though, this is not a particularly impressive result when we look at how well we did at predicting good or bad wines.

Overall, it appears, from this one example, that if minority class prediction is an important metric for model evaluation in multi-class classification probems, then class rebalancing may be worthwhile if the original data is highly unbalanced. If the main criterion is overall accuracy, then rebalancing may not be worth the effort, as it will almost certainly reduce accuracy.

On predicting wine quality

Finally, it's worth noting that this entire exercise in predicting quality based on chemical ratings may be flawed to begin with. For example, it has been shown that people will physically find a wine to be more pleasurable (and therefore of higher quality), if they believe it has a higher price point. This difference in quality perception has been verified through MRI brain scans.[11] In other words, if we know that people's *physical* experience of pleasure from a wine can be influenced by the wine's price point, it's hard to believe that chemical analysis is really going to be able to provide a strongly-reliable set of predictors of wine quality.

This malleable aspect of quality ratings probably puts an upper bound on the ability of any model to predict quality based on chemical ratings. In the original paper, the white wine predictions were not noticeably better than the red wine predictions (SVM gave 64.6% accuracy at T=0.5 for white, vs 62.4% for red). This is in spite of the fact that the white win data set is three times the size of the red wine data set.

[11] Plassman et al, "Marketing Actions Can Modulate Neural Representations of Experienced Pleasantness"

There is definitely some room for improvement, likely through an ensemble learning model composed of more and better components, and possibly using a better second-level model than plurality vote. Given the strength of random forest and tree-based bagging, it seems likely that a few other tree approaches could help build a stronger ensemble model. Even with a better selection of constitutent models, however, I am doubtful that we could see significant increases in predictive performance.

Appendix 1 - Model code

This function has been coded **only** to run and evaluate the classification algorithm. It doesn't actually return the models that are developed - it only returns the predictions for the final model and each of the intermediate models. If the actual models are needed, it's a fairly trivial change to actually return them.

```
# Helper function to get mode of a vector
getMode
```

```
## function (vect)
## {
##     unq_vect <- unique(vect)
##     return(unq_vect[which.max(tabulate(match(vect, unq_vect)))])
## }
```

```
# Main code, including test set evaluation algorithm
ensemble_eval
```

```
## function (train_set, test_set = NULL)
## {
##     evaluate_test <- function(model_list, test_set) {
##         test_set <- predict(model_list[[7]], test_set)
##         ensemble_data <- data.frame(matrix(NA, nrow = nrow(test_set),
##             ncol = 0))
##         for (i in 2:6) {
##             new_pred <- predict(model_list[[i]], test_set)
##             ensemble_data <- cbind(ensemble_data, new_pred)
##         }
##         colnames(ensemble_data) <- c("mn", "nb", "rf", "gbm",
##             "lda")
##         test_pred <- apply(ensemble_data, 1, function(x) getMode(x))
##         ensemble_data <- cbind(final = test_pred, ensemble_data)
##         return(ensemble_data)
##     }
##     fit_control <- trainControl(method = "repeatedcv", number = 10,
##         repeats = 10)
##     norm_obj <- preProcess(train_set[, -12], method = c("center",
##         "scale"))
##     train_set <- predict(norm_obj, train_set)
##     set.seed(15954)
##     mn_model <- train(quality ~ ., data = train_set, method = "multinom",
##         trControl = fit_control, trace = FALSE)
##     mn_pred <- predict(mn_model, train_set)
##     set.seed(15954)
##     nb_model <- train(quality ~ ., data = train_set, method = "nb",
##         trControl = fit_control)
##     nb_pred <- predict(nb_model, train_set)
##     set.seed(15954)
##     rf_model <- train(quality ~ ., data = train_set, method = "rf",
##         trControl = fit_control)
##     rf_pred <- predict(rf_model, train_set)
##     set.seed(15954)
##     gbm_model <- train(quality ~ ., data = train_set, method = "gbm",
```

```
##         trControl = fit_control, verbose = FALSE)
##     gbm_pred <- predict(gbm_model, train_set)
##     set.seed(15954)
##     lda_model <- train(quality ~ ., data = train_set, method = "lda",
##         trControl = fit_control)
##     lda_pred <- predict(lda_model, train_set)
##     ensemble_data <- data.frame(mn = mn_pred, nb = nb_pred, rf = rf_pred,
##         gbm = gbm_pred, lda = lda_pred)
##     final_pred <- apply(ensemble_data, 1, function(x) getMode(x))
##     if (missing(test_set)) {
##         return(cbind(final = final_pred, ensemble_data))
##     }
##     else {
##         return(evaluate_test(list(final_model = final_pred, mn_model = mn_model,
##             nb_model = nb_model, rf_model = rf_model, gbm_model = gbm_model,
##             lda_model = lda_model, norm_obj = norm_obj), test_set))
##     }
## }
```

Appendix 2 - Over/Under Sample and Multiclass SMOTE Algorithm code

Pseudocode for the wrapper, undersample & undersample algorithms can be found in the Chawla et al "Automatically Countering Imbalance".

Pseudocode for the SMOTE algorithm can be found in Chawla et al "SMOTE Synthetic Minority Oversampling"

```
wrapper
```

```
## function (class_list, train_set, eval_fun, seed = FALSE)
## {
##     require(caret)
##     require(FNN)
##     class_data <- data.frame(rbind(orig = class_list))
##     colnames(class_data) <- names(class_list)
##     class_data <- rbind(class_data, maj = class_list == 1)
##     class_data <- rbind(class_data, min = class_list == -1)
##     class_data <- rbind(class_data, under = as.numeric(class_data[1,
##         ] == 1))
##     class_data <- rbind(class_data, smote = rep(0, ncol(class_data)))
##     class_data <- rbind(class_data, base_f = evaluate_model(eval_fun,
##         train_set))
##     class_data <- rbind(class_data, best_f = class_data["base_f",
##         ])
##     original_data <- train_set
##     if (sum(class_data["maj", ]) > 0) {
##         new_list <- wrap_undersample(class_data, train_set, eval_fun,
##             seed)
##         train_set <- new_list[[1]]
##         class_data <- new_list[[2]][rownames(new_list[[2]]) !=
##             "stop_sampling", ]
##     }
##     if (sum(class_data["min", ]) > 0) {
##         new_list <- wrap_smote(class_data, train_set, original_data,
##             eval_fun, seed)
##         train_set <- new_list[[1]]
##         class_data <- new_list[[2]]
##     }
##     return(train_set)
## }
```

```
wrap_undersample
```

```
## function (class_data, train_data, eval_fun, seed = FALSE)
## {
##     original <- train_data
##     class_data <- rbind(class_data, stop_sampling = !class_data["maj",
##         ])
##     while (sum(class_data["stop_sampling", ]) < ncol(class_data)) {
##         for (i in 1:ncol(class_data)) {
##             if (class_data["stop_sampling", i] == FALSE) {
```

```
##                 train_len = nrow(train_data)
##                 train_data <- undersample(train_data, original,
##                   colnames(class_data)[i], class_data, eval_fun,
##                   seed)
##                 if (nrow(train_data) == train_len) {
##                   class_data["stop_sampling", i] <- TRUE
##                 }
##             }
##         }
##         new_f <- evaluate_model(eval_fun, train_data)
##         class_data["best_f", ] <- pmax(as.numeric(class_data["best_f",
##             ]), new_f)
##     }
##     return(list(train_data, class_data))
## }
```

```
undersample
```

```
## function (train_data, original, class_val, class_data, eval_fun,
##     seed = FALSE)
## {
##     sample_decrement <- 0.1
##     increment_minimum <- 0.05
##     other_classes <- train_data[train_data[, ncol(train_data)] !=
##         class_val, ]
##     curr_class <- train_data[train_data[, ncol(train_data)] ==
##         class_val, ]
##     if (seed) {
##         set.seed(2292)
##     }
##     curr_class <- curr_class[sample(nrow(curr_class), floor(nrow(curr_class) *
##         (1 - sample_decrement))), ]
##     new_train <- rbind(other_classes, curr_class)
##     new_f <- evaluate_model(eval_fun, new_train, original)
##     for (i in length(new_f)) {
##         if (is.nan(new_f[i])) {
##             new_f[i] <- 0
##         }
##     }
##     if (mean(new_f[class_data["orig", ] == 1]) < (1 - increment_minimum) *
##         mean(as.numeric(class_data["best_f", class_data["orig",
##             ] == 1]))) {
##         return(train_data)
##     }
##     else if (mean(new_f[class_data["orig", ] == -1]) < mean(as.numeric(class_data["best_f",
##         class_data["orig", ] == -1]))) {
##         return(train_data)
##     }
##     else {
##         return(new_train)
##     }
## }
```

```
wrap_smote
```

```
## function (class_data, train_data, original, eval_fun, seed = FALSE)
## {
##     class_data <- rbind(class_data, stop_sampling = !class_data["min",
##         ])
##     class_data <- rbind(class_data, lookup_ahead = 1)
##     while (sum(class_data["stop_sampling", ]) < ncol(class_data)) {
##         for (i in 1:ncol(class_data)) {
##             if (class_data["stop_sampling", i] == FALSE) {
##                 train_len = nrow(train_data)
##                 smote_results <- smote_sample(train_data, original,
##                   colnames(class_data)[i], class_data, eval_fun,
##                   seed)
##                 train_data <- smote_results[[1]]
##                 class_data <- smote_results[[2]]
##                 if (nrow(train_data) == train_len) {
##                   class_data["stop_sampling", i] <- TRUE
##                 }
##             }
##         }
##         new_f <- evaluate_model(eval_fun, train_data)
##         class_data["best_f", ] <- pmax(as.numeric(class_data["best_f",
##             ]), new_f)
##     }
##     return(list(train_data, class_data))
## }
```

```
smote_sample
```

```
## function (train_data, original, class_val, class_data, eval_fun,
##     seed = FALSE)
## {
##     increment_minimum <- 0.05
##     lookup_ahead_value <- 3
##     other_classes <- train_data[train_data[, ncol(train_data)] !=
##         class_val, ]
##     curr_class <- train_data[train_data[, ncol(train_data)] ==
##         class_val, ]
##     synth_class <- smote(curr_class, train_data, seed)
##     new_f <- evaluate_model(eval_fun, rbind(train_data, synth_class),
##         original)
##     for (i in length(new_f)) {
##         if (is.nan(new_f[i])) {
##             new_f[i] <- 0
##         }
##     }
##     if (mean(new_f[class_data["orig", ] == -1]) < (1 + increment_minimum) *
##         mean(as.numeric(class_data["best_f", class_data["orig",
##             ] == -1]))) {
##         if (class_data["lookup_ahead", class_val] < lookup_ahead_value) {
##             class_data["lookup_ahead", class_val] <- class_data["lookup_ahead",
##                 class_val] + 1
```

```
##             return(list(rbind(train_data, synth_class), class_data))
##         }
##         else {
##             return(list(train_data, class_data))
##         }
##     }
##     else {
##         return(list(rbind(train_data, synth_class), class_data))
##     }
## }
```

```
smote
```

```
## function (minor_class, train_data, seed = FALSE)
## {
##     k <- 6
##     synth_data <- data.frame(matrix(NA, nrow = 0, ncol = ncol(minor_class)))
##     colnames(synth_data) <- colnames(minor_class)
##     if (seed) {
##         set.seed(2292)
##     }
##     nbr_idxs <- get.knnx(train_data[, 1:(ncol(train_data) - 1)],
##         minor_class[, 1:(ncol(train_data) - 1)], k = k)[[1]]
##     for (i in 1:nrow(minor_class)) {
##         near_nbrs <- train_data[nbr_idxs[i, 2:6], ]
##         class_rep <- minor_class[i, ]
##         synth_data <- rbind(synth_data, populate(class_rep, near_nbrs,
##             k - 1))
##     }
##     return(synth_data)
## }
```

```
populate
```

```
## function (class_rep, near_nbrs, k)
## {
##     os_rate <- 1
##     new_samples <- data.frame(matrix(NA, nrow = 0, ncol = ncol(near_nbrs)))
##     colnames(new_samples) <- colnames(near_nbrs)
##     while (os_rate > 0) {
##         nn <- sample(1:k, 1)
##         synthetic <- class_rep[1, ]
##         for (i in 1:(ncol(near_nbrs) - 1)) {
##             dif <- near_nbrs[nn, i] - class_rep[1, i]
##             gap <- runif(1, 0, 1)
##             synthetic[1, i] <- class_rep[1, i] + gap * dif
##         }
##         new_samples <- rbind(new_samples, synthetic)
##         os_rate <- os_rate - 1
##     }
##     return(new_samples)
## }
```

```
# Custom evaluation function
evaluate_model
```

```
## function (eval_fun, train_set, test_set = NULL)
## {
##     if (missing(test_set)) {
##         cm <- confusionMatrix(eval_fun(train_set)$final, train_set[,
##             ncol(train_set)])
##     }
##     else {
##         cm <- confusionMatrix(eval_fun(train_set, test_set)$final,
##             test_set[, ncol(test_set)])
##     }
##     f_vals <- (2 * cm$byClass[, 1] * cm$byClass[, 3])/(cm$byClass[,
##         1] + cm$byClass[, 3])
##     return(f_vals)
## }
```

Bibliography

Chawla NV, Cieslak DA, Bowyer KW, Kegelmeyer WP SMOTE Synthetic Minority Over-sampling TEchnique Journal of Artificial Intelligence Research. 2002;16:341-78

Chawla NV, Cieslak DA, Hall LO, Joshi A. Automatically countering imbalance and its empirical relationship to cost. Data Mining and Knowledge Discovery. 2008;17(2):225-52.

Cortez P, Cerdeira A, Almeida F, Matos T, Reis J. Modeling wine preferences by data mining from physicochemical properties. Decision Support Systems. 2009;47(4):547-53.

Foreman, George and Cohen, Ira, Learning from Little: Comparison of Classifiers Given Little Training, Hewlett-Packard Research Laboratories, 15th European Conference on Machine Learning and the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases, 20-24 September 2004, Pisa, Italy http://www.ifp.illinois.edu/~iracohen/publications/precision-ecml04-ColorTR-final.pdf

Plassmann H, O'Doherty J, Shiv B, Rangel A. Marketing Actions Can Modulate Neural Representations of Experienced Pleasantness. Proceedings of the National Academy of Sciences of the United States of America. 2008;105(3):1050-4.

Qian Y et al., A resampling ensemble algorithm for classification of imbalance problems, Neurocomputing (2014), http://dx.doi.org/10.1016/j.neucom.2014.06.021i

Rokach L. Ensemble-based classifiers. Artificial Intelligence Review. 2010;33(1):1-39.