To be Wined and Splined: Building an Ensemble Model to Predict Wine Quality

William Chen | Yuan Jiang
wchen@college.harvard.edu | yuanjiang@college.harvard.edu

Introduction

For our Stat 149 final project, we were tasked with predicting the quality of wine given certain physicochemical properties. Our Kaggle dataset was composed of the binary response variable `good` (good = 1, bad = 0), and the following predictor variables: `fixed.acidity, volatile.acidity, citric.acid, residual.sugar, chlorides, density, pH, sulphates, alcohol, free.so2, total.so2, good`. All computational analysis was completed in R.

While many of our models were not covered in class, we want to make clear that our procedures and methodology were heavily motivated by the material that we covered in Stat 149. We will discuss our motivations and inspiration throughout this report.

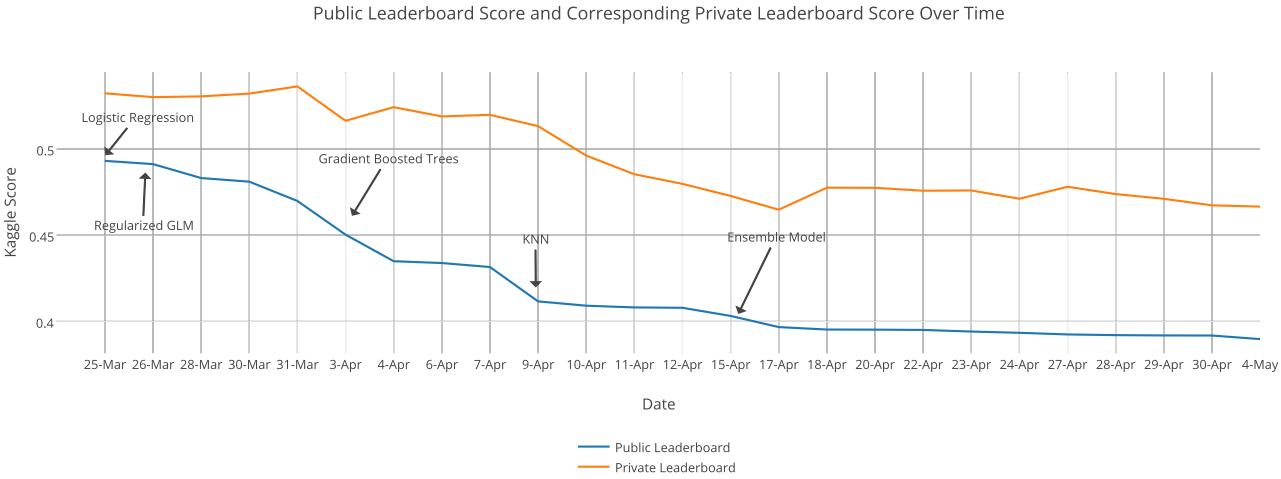

Figure 1: Plot of our public and private leaderboard scores. Notable decreases in score can be observed when we used gradient boosted trees and K-nearest neighbors. Once we started using ensemble methods, however, we started getting very minor improvements in score. The improvements were not reflected in the private leaderboard scores, which may be indicative of overfitting in the public leaderboard dataset or anomalies in the private leaderboard dataset.

Individual Models

Our final prediction was an ensemble of 10 models and 7 methods. We allocate a section to each method, although elaborate on KNN and RF the most since they performed the best. We allocate less space to the other models due to space constraints. In a later section we will describe how we stacked all of our models.

Table 1: Our final model was an ensemble prediction using all 10 of the below models. They are presented in order of their score on the public leaderboard.

Model	Data	Public Score	Private Score
KNN	Raw	0.40788	0.47977
KNN	Log Transformed	0.41329	0.49216
RF	Raw	0.41634	0.47865
GBT	Raw	0.43297	0.51151
RF	Dim-reduced via PCA	0.43329	0.50097
GAM	Log Transformed	0.45551	0.52233
Regularized GLM	Piecewise	0.45983	0.53526
SVM	Raw	0.46090	0.50725
SVM	Dim-reduced via PCA	0.46806	0.53244
ANN	Raw	0.47440	0.55028

K-Nearest Neighbors

K-nearest-neighbors was the best two models that we tried, with public leaderboard scores of `0.40788` and `0.41329`, for raw and log-transformed data. The basic idea behind this model is as follows: to predict the goodness of any given wine, we identify wines that are very close to it in chemical composition, and then take a weighted average of the goodness of those similar wines to predict the goodness of our given wine. The three parameters that we tuned for this model are as follows:

Value of k - How many 'neighbors' we use for each wine that we want to predict

Choice of kernel - If $k > 1$, we have a choice of weighting the closer wines more than the farther wines. The choice of kernel dictates how we weight the wines when determining the final estimate. Please see Figure 2 for a graphical comparison of the some of the various kernels that we attempted, and Figure 3 for their results.

Prior strength (k') - KNN posed problems since we were getting a few incorrect 100% predictions, which broke our deviance calculation. We addressed this issue by introducing a prior - fake neighbors that help bring the probability estimate away from 100% and closer to the mean. This is similar to the bias-reduced GLM method that we learned in class. We liked the method from class so we wanted to introduce it to our KNN model by introducing fake data points to help with model stability. If $\hat{p}$ is the raw estimate, $\bar{y}$ is the empirical mean of the goodness indicator on the training dataset, k is the number of neighbors in our dataset, and k' is the number of neighbors in our prior, then our new estimate of the probability $\hat{p}'$ is:

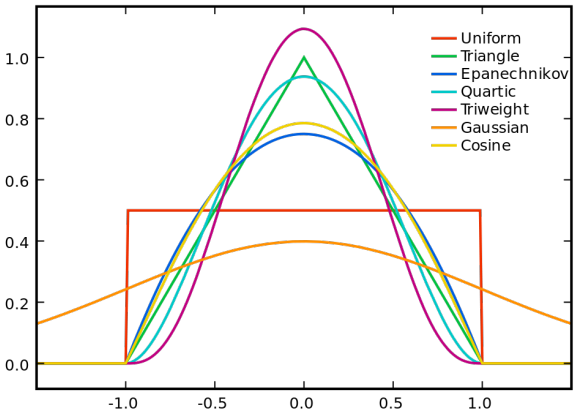

Figure 2: Comparison of various kernels. We ran cross-validation and compared the performance of our KNN algorithm on the triweight, biweight, gaussian, triangular, cos, epanechnikov, and optimal kernels, which were the 7 kernels available in the `kknn` package in *R* that performed somewhat well in our exploratory runs. Figure from Wikipedia

$$\hat{p}' = \frac{k}{k+k'}\hat{p} + \frac{k'}{k+k'}\bar{y}$$

To tune the parameters, we run cross-validation on all three of the tuning parameters. The best tuning parameters ended up being `k = 35, k' = 3, kernel = 'triweight`. Please see Figure 3 for a visualization of the cross-validation.

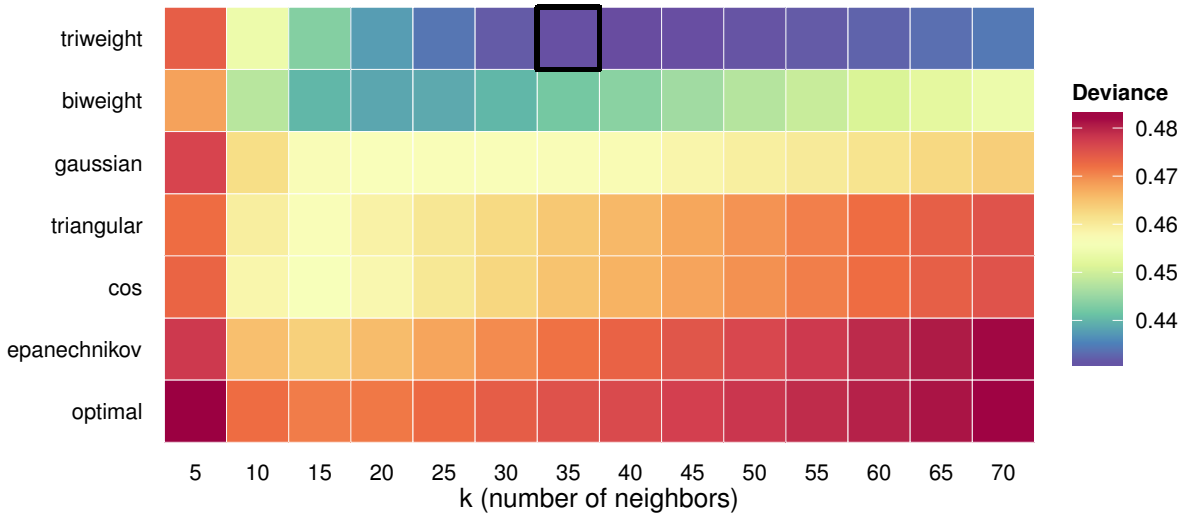

Figure 3: (KNN) Visualization of 20-fold cross-validation results with `k' = 3` fixed and tuning kernel and `k`. The best tuning parameter (boxed in the visualization) was `k = 35` and `kernel = 'triweight'` with a resulting deviance of `0.4320047`. Any tuning parameter combination with `k' != 3` did not do as well.

Random Forest

Random forests are an ensemble of decision trees and are used to deal with the issue of high variance and biasedness in classification and regression trees. Random forests are advantageous in that they

are effectively non-parametric (defuault parameters work well for most models), and can be created by generating many bootstrap samples from the data, fitting a tree model for each sample, and then averaging the models that are created. Since random forests perform well without significant parameter tuning, they are a good model to go to when we have a large dataset or are unsure of the underlying model.

Our random forest model on the raw dataset was the third-best performing model, with a public leaderboard score of `0.41634`. The random forest method was the second-best performing method, since both of the top two models were KNN models. We stuck with the default values of `mtry` $= \sqrt{10}$ (the number of randomly selected predictors to try at each tree node) and we used `ntree = 10000` just for as much precision as possible. Below is a summary of our model:

```
Call:
 randomForest(formula = good ~ ., data = train.new, ntree = 10000)
               Type of random forest: classification
                     Number of trees: 10000
No. of variables tried at each split: 3

        OOB estimate of error rate: 19.16%
Confusion matrix:
    0    1 class.error
0 831  428   0.3399523
1 276 2139   0.1142857
```

From the R output above, we observe that our model has an overall misclassification error of 0.1916 and that the number of predictors tried at each node was 3. There are multiple factors that can affected the misclassfication error: 1. Increasing the correlation between two trees in the forest will increase the forest error rate and 2. Using trees with lower error rates decreases the forest error rate.

Regularized Generalized Linear Model

Logistic regression models are used to predict Bernoulli distributed outcomes. The model is fitted using maximum likelihood estimation. The probability function of a logistic regression model is $P(y_i = 1) = \frac{\exp(\boldsymbol{x_i}'\boldsymbol{\beta})}{1+\exp(\boldsymbol{x_i}'\boldsymbol{\beta})}$, where x_i is the ith observation and the $\boldsymbol{\beta}$ are the coefficient estimates.

Given that we were predicting binary outcomes, the most obvious choice was logistic regression. However, instead of doing normal logistic regression, we chose to use regularized GLM instead because regularized GLM penalizes heavy weights in a logistic regression model. Due to the great amount of variance present in the dataset, we do not wish for one predictor to be too dominant. Regularized GLM penalizes big coefficients by increasing cross validation scores. We reduce the variance of the estimator by shrinking the estimates closer to 0 and controlling for large weights. After creating histograms of each predictor separately, we decided to log-transform `fixed.acidity`, `volatile.acidity`, `chlorides`, `sulphates`, `free.so2`, `total.so2`, and `residual.sugar`. We used these same log transformations for many of our other models. Our initial logistic regression included many interaction terms. We included pairwise, triple, quadruple, and quintuple interaction terms, ultimately choosing the quadruple interaction terms since it produced the lowest logloss score.

One of the other important aspects that we introduced into our model is the use of indicators. After creating several plots of each predictor (some transformed) and the response variable, we noticed that many predictors appeared to have a non-linear relationship in the binnedplot of the average of the response variable. One such example is illustrated in Figure 4.

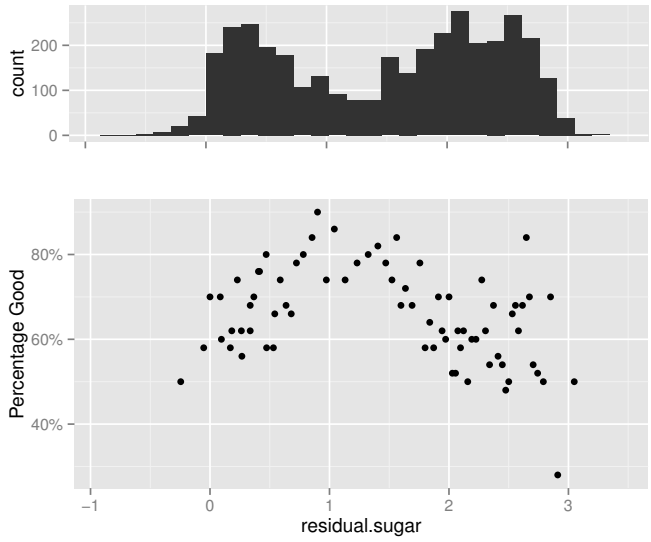

Figure 4: From the scatter plot above, we observe that there seems to be a quadratic relationship present. Residual sugar is logged here.

After recognizing that several predictors were non-linear, we then marked the breakpoints for each predictor, and created new predictor variables from each breakpoint. Finally, after significant parameter tuning, our final regularized GLM incorporates all pairwise interaction terms, and uses an alpha value of 0.3. Please see Figure 5 for a simple demonstration of the power of including breakpoints to address the concerns about non-linearity in our dataset.

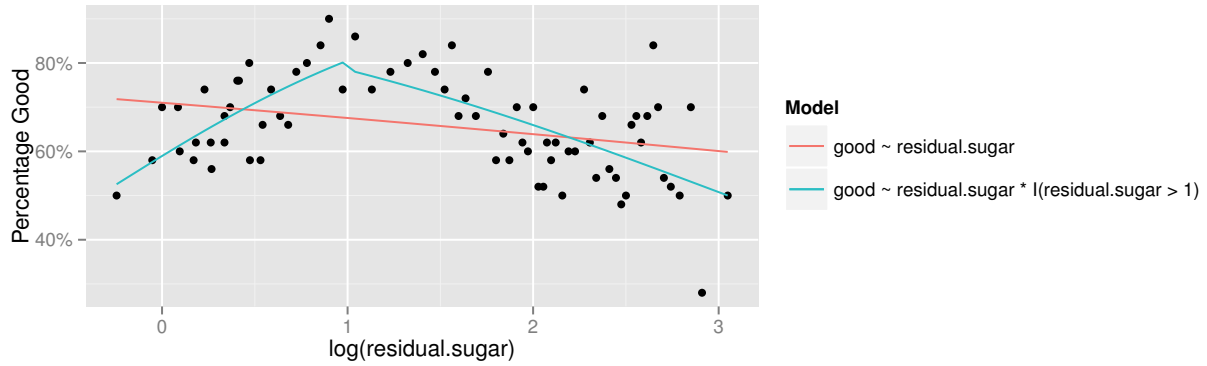

Figure 5: We see that a model that interactions with indicators of a predictor being greater than a breakpoint can explain the data a lot better.

Our final fitted model is `train.glmnet <- cv.glmnet(train.new, factor(train$good), alpha`

= 0.3, family = c("binomial"), nfolds = 10) .

General Additive Model

GAMS are fitted using an algorithm that iteratively creates weighted additive model by backfitting. We can view GAMs as an extension of GLMs (or GLMs as a special case of GAMs), where the link function of the mean of response does not have to depend linearity on the predictors, but instead can have a non-linear and even non-monotonic dependence. We used GAMs on a log-transformed version of the dataset as we were concerned about the potential influence of outliers.

As hinted in class, we first use a decision tree on the raw data to assess the importance of each variable. From our decision tree we get that the relative effect of each variable is as follows:

1	alcohol	239.2198475
2	density	142.5683227
3	volatile.acidity	113.7953792
4	chlorides	98.0396981
5	total.so2	60.8401872
6	residual.sugar	43.0113102
7	sulphates	14.8703164
8	free.so2	13.4990594
9	pH	8.6750071
10	citric.acid	0.7094956

We note that `fixed.acidity` is not used at all in our decision tree. We now fit a sequence of 11 GAM models, each with one more predictor than the last, doing this in order of the importance from our decision tree. We use cross-validation performance to assess which model to use. From our cross-validation function, we determined that the full model produced the lowest cross-validation score (with a CV score of `0.4720821`), though the residual diagnostics were poor. Please see Figure 6 for our residual plot.

We then went through each model and finally chose the GAM with the best residual diagnostics. Our final GAM is `form11 <- good ~ s(alcohol) + s(density) + s(volatile.acidity) + s(chlorides) + s(total.so2) + s(residual.sugar) + s(sulphates) + s(free.so2) + s(pH) + s(fixed.acidity) + s(citric.acid)` . Below is a plot of the fitted values against the deviance residuals.

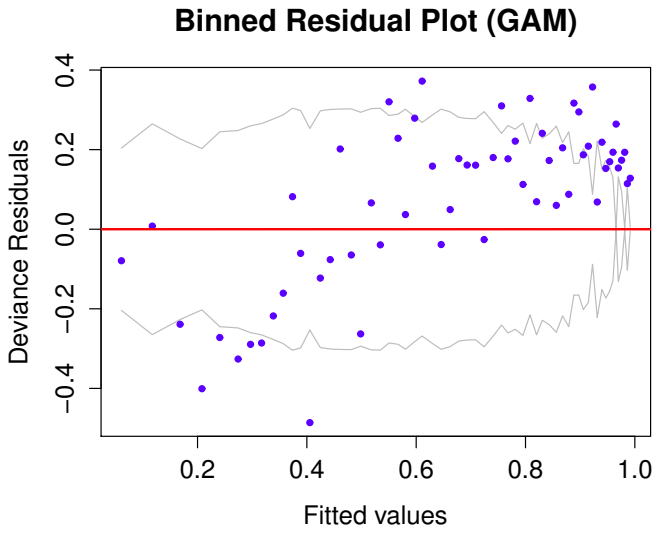

Figure 6: Residual diagnostics were poor for our final GAM model, although cross-validation predictive deviance was good.

From the residual plot above, we observe that there appears to be strange clustering and unusual patterns in the residuals. Given more time, we would refit our GAM, but since our current one produced the lowest Kaggle score, we will continue with this model.

Our GAM model scored us `0.45551` on the public leaderboard, which was within our range of expectations given our cross-validation score of `0.4720821` and the variation we would expect given the inevitable differences between the training and the test set and the high level of sensitivity of the deviance metric. We note that the `0.4720821` cross-validation score of our GAM is much better than the `0.5082535` cross-validation score that we got using GLM on the exact same log-transformed data. From our understanding and learning from class, this is because we applied a smoothing spline fit with the default `df = 4` to each of our predictors, and allowed the model to find the best parameters for the smoothing to best capture the information in the data.

The loosening of the restriction of linearity combined with the clear non-linear effects in the dataset (See Figure 4 for an example - clearly a linear pattern will not do!), allows greater predictive accuracy compared to a linear regression. The lack of need to transform X to get the right pattern of linearity is also troublesome since it involves a large degree of trial and error, as we learned in class! I note that GAM performed better than our Regularized GLM on both the public and private leaderboard, despite the fact that our Regularized GLM model included interaction terms and despite the fact that we introduced indicators to model separate parts of non-linear patterns separately. This supports the claim that we should let the optimizer choose the breakpoints and the patterns for us, instead of doing them manually.

However, I will note that the GAM method did not perform as well as our KNN or our Tree ensemble methods. I note that KNN and GAM methods are similar in that they use nearby neighbors to determine the best estimates. However, I postulate that KNN performed better as it was not limited by a functional form, and considered all of the predictors jointly, while our GAM did not consider the interactions between the variables. Given more time we would have attempted GAM

with interactions between the varabies. I postulate that GAM did not do better than our either of our forest methods (Random Forest and GBT), as the GAM gives a single prediction, while the random forest and the gradient boosted trees are an emsemble of predictors on slightly varying datasets that were averaged to form an ensemble prediction.

Gradient Boosted Trees

We used a gradient boosted tree model as a complement to the random forest model that we learned in the class. Both of them are similar in that they are a collection of decision trees, with each decision tree made on a slightly different version of the data. One main difference between random forests and gradient boosted trees is that random forests used bagged datasets (resamplings from the original dataset), while gradient boosted trees uses boosted datasets (the same dataset with different weightings, with datapoints with high residuals weighted more in the next iteration).

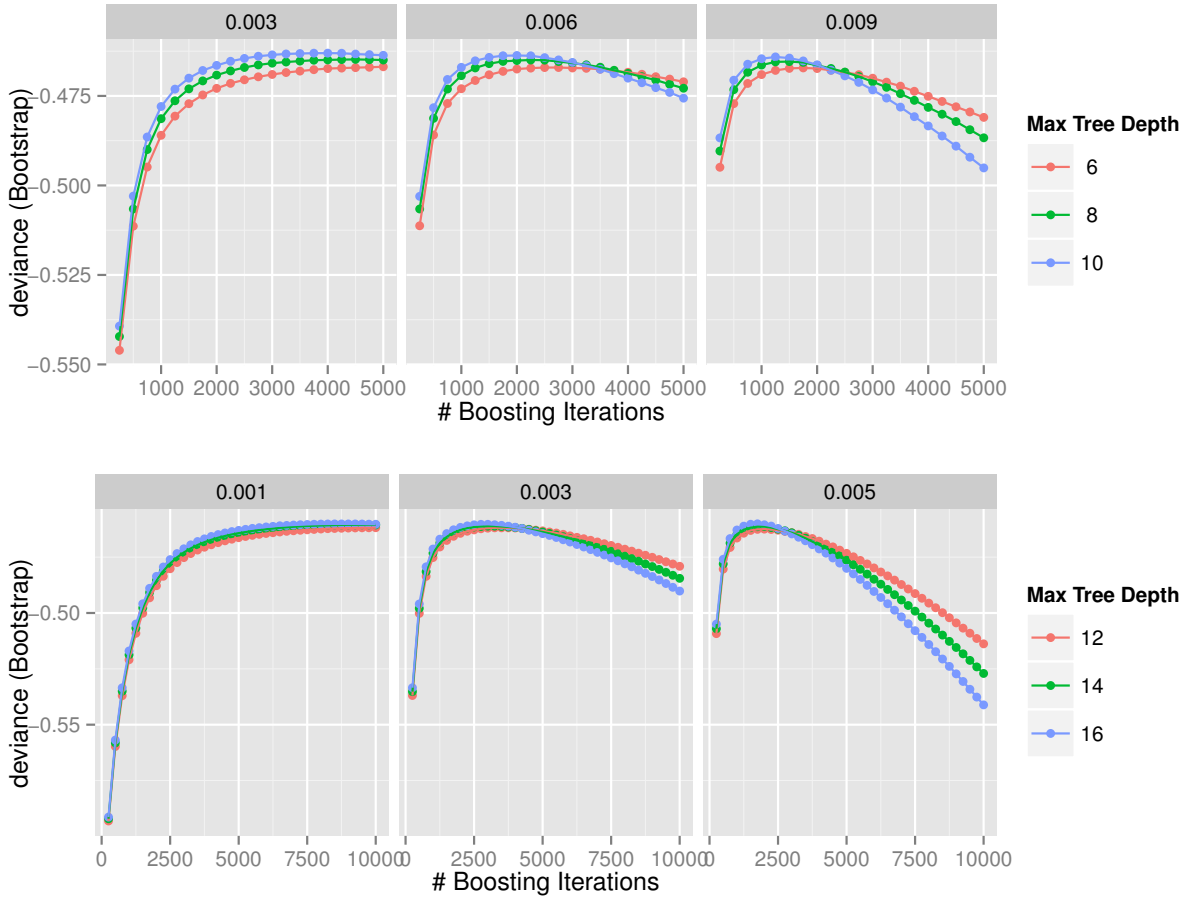

Figure 7: We performed cross validation to select the optimal tuning parameters for our Graident Boosted Trees model.

The final tuning parameter that we used for our Gradient Boosted Tree algorithm is `ntree = 1750, shrinkage = 0.005, interaction = 16`. We limit our explanation in the interest of space and to focus on topics that we covered in class.

Support Vector Machines

Support vector machines (SVMs) are learning models that recognize patterns in data, and often used for classification purposes. Conceptually, SVMs attempt to draw a line through the data points that maximizes the distances from each observation in each group to the line, thus separating the data into groups. Our SVM model aimed to classify the wines into a "good" and "bad" group. We used our cross-validation function to help tune the parameters. However, the results on the public leaderboard from submitting the tuned model did not outperform the model with the default parameters, so we just stuck with the default parameters and ended up with scores of `0.46090` and `0.46806` using the raw dataset and a dimensionality-reduced dataset, respectively.

We limit our explanation for the following three reasons 1) This is definitely not within the scope of the class 2) This was one of our worse performing individual models 3) We have limited space on this report.

Artificial Neural Networks

We tuned the `size` parameter and reached an optimum value of `20`. We made a single-layer artificial neural network with a middle layer of size 20, and achieved a public leaderboard score of `0.47440`. We will limit focus on ANN in this report for the same three reasons as above.

Transformations

For some of the models, we inputted datasets with log transformations as we postulated that fixing some of the outliers and making the predictors more normal would improve our predictions. We logged `fixed.acidity, volatile.acidity, chlorides, sulphates, free.so3, total.so2, and residual sugar`. Logging did not help KNN, as our public score decreased from `0.40788` to `0.41329` after logging. We do the log transformations as we are inspired by the material in class about Cook's distance and influential outliers. We know that oftentimes a data point with high cook's distance / influence is caused by it being an outlier in the predictor distribution, so we wanted to log the data to see if we can make a better model by making the predictors more normal and better behaved.

The piecewise transformation used in the regularized GLM is described in the regularized GLM section.

We will not discuss dimensionality reduction using PCA because it is not within the scope of the class, we are low on space, and it did not improve our predictive accuracy for either Random Forest or SVM models. The basic idea was that we wanted to reduce our dataset from 11 dimensions to 5 dimensions while still keeping most of the information, to reduce issues with using high dimensional datasets. We suspect that the performance was not as good with the reduced dataset since we lost information in the reduction.

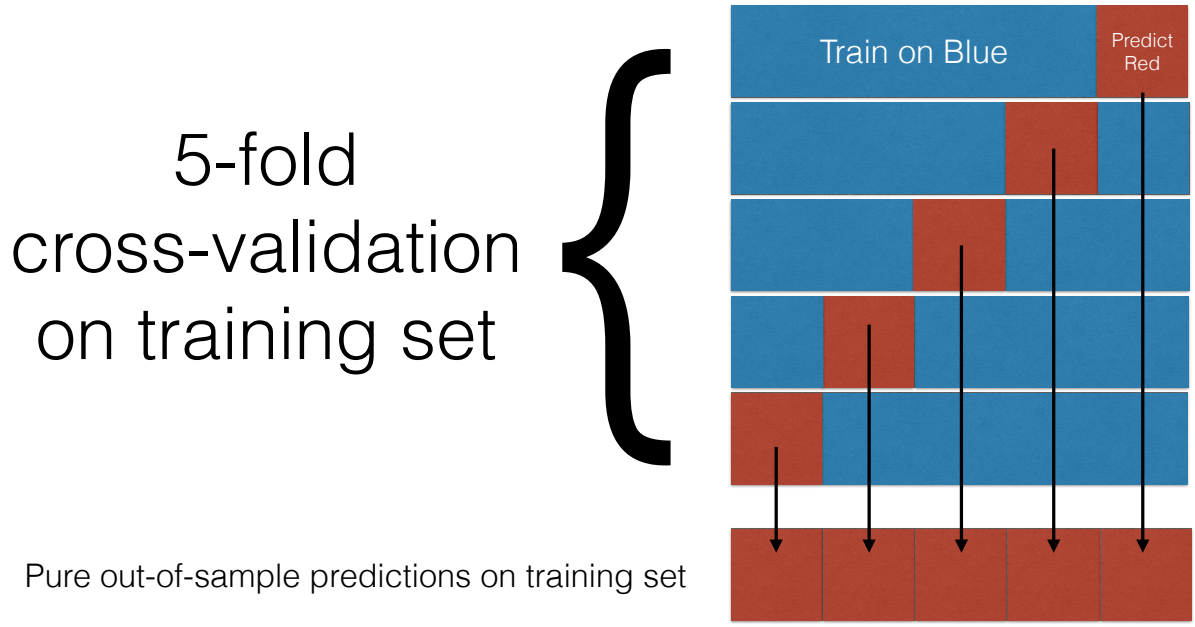

Figure 8: Each row represents the training dataset. For each row, we train on the blue portions to make a prediction on the red portion. The prediction on the red portion is usable since it does not incorporate the actual answer. We can put all of these red portions together to get predicted probabilities for the whole dataset, all of which are out-of-sample (e.g. do not incorporate the actual answer). From these out-of-sample probabilities from 10 models, we can put them in a logistic regression model which will automatically weight all of the models for me to combine the probability estimates into one meta-estimate that I will submit.

Ensemble Model

The premise of our ensemble model is simple. We use probability estimates from the 10 models, and then combine them together with a regularized logistic regression to pool together the estimates from the model and create one meta-prediction. We use use cross-validation again to choose the optimal regularization strength parameter.

One key caveat is that our probability estimates for the wines must be out-of-sample probability estimates. That is, our predicted probability for wine A being good in model B *cannot* actually incorporate the the actual indicator that wine A is good. If it incorporates the actual indicator, then we would worry that our ensemble model fit on the training dataset will not actually be able to predict on our test dataset, because of course the test dataset will not give us access to the actual goodness indicator. I will address this issue by making sure that all of the probability estimates from the 10 models are out-of-sample probability estimates. I can do this by pulling out the out-of-sample probability estimates that I made in cross-validation. Please see Figure for a pictorial explanation.

Evaluation

One thing that surprised us was that our initial logistic regression model performed poorly compared to other models such as random forest and K-nearest neighbors. Even after adding indicators, transformations, and interaction terms, our GLM did not produce as low of a cross validation score as we had originally expected.

We were also surprised that our random forest model performed so well despite very little parameter tuning. We would like to tune our random forest model even more to see if improvements are possible.

Given more time, we would have tuned SVM more as well. There was a lot of potential for parameter tuning in our SVMS, and next time we would use a cross validation function to help tune the gamma and C parameters.

We also would use principal component analysis to detect clusters in the dataset. Another direction we could have gone is predicting the quality of wine separately for red and white wines. This is an interesting direction because the kind of wine may potentially affect the score it receives from the taster.