Report on the wine quality prediction

Advanced statistical learning (80619A) - March 2020

Liudmyla Kotusenko

In this assignment, we aim to predict the wine quality score (Y) given wine characteristics (Xs). Y is an ordinal variable where 1 refers to inferior, 2 to average, 3 to superior wine quality. We aim to maximize the accuracy.

Our variable has three categories and the naïve approach would be to set our task as a multi-class classification problem, treating it as a categorical variable. However, as the paper by Gutierrez et al. (2015) suggests, when predicting an ordinal variable, we can leverage information from the categories order and thus get better accuracy compared to treating the same variable as categorical. We try and compare both approaches.

First, we explored the data set and found out the following:

- Y classes 1, 2, 3 are distributed in proportion 33.25%, 45.25% and 21.50% respectively, so our data set is somewhat imbalanced. If we predicted all cases as the majority class ("average quality") we would be correct 45.25% of the time. We can treat this accuracy rate as our naïve benchmark. We can also try applying class weights in models that allow for them and explore its impact on accuracy.
- Many of our Xs have a distribution skewed to the right with a long tail. We might consider some pre-processing of Xs for regression-based methods or use methods that are invariant to monotone transformations of predictors such as trees and random forests.
- We plotted Y against Xs using boxplots and density plots. Across many predictors, the density curves of a given X for different classes of Y tend to overlap, and boxplots show little differentiation between classes. We conclude that most predictors have little power in discriminating the classes of our Y. Thus, we should not expect an outstanding accuracy in this task.

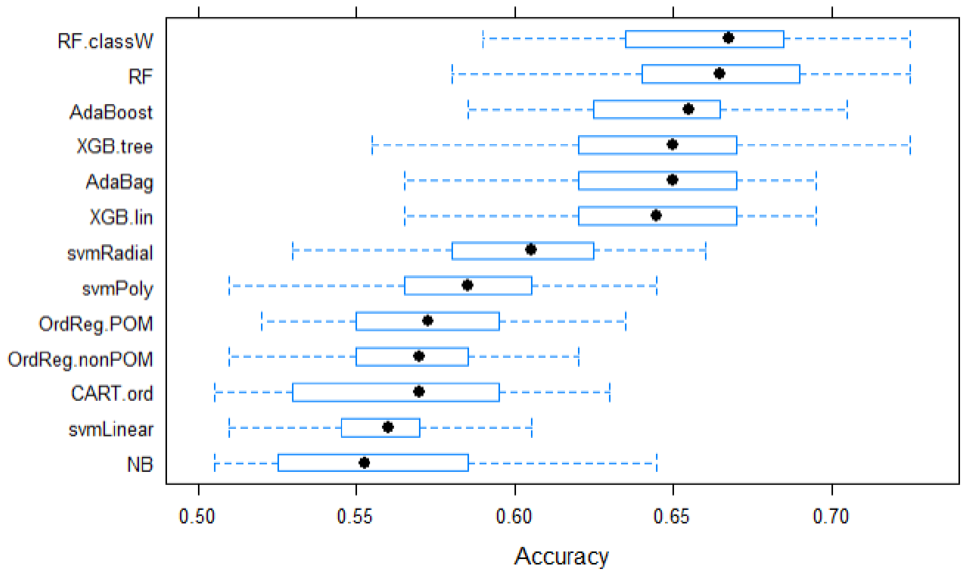
Early in the modeling process we concluded that splitting our data set with 2000 observations into a training and a test set did not work well. Depending on a random seed used to split the data, the test accuracy varied sometimes 10-15 percentage points for the same model type. The ranking of models by performance also changed from split to split. To select the best model, we could not rely on the accuracy of a given validation set as our predictions tended to have high variance. To get more reliable accuracy estimates, we decided to use 10-fold cross-validation (CV) repeated 5 times. We conducted CV on the whole training set of 2000 samples (datrain.csv), so the validation accuracy shown below is calculated on the whole data set. This, however, increased running times substantially.

To deal with CV, we relied on the *caret* package [3]. Using its function *createMultiFolds*, we created the same splits for CV that could be reused to fit different models. This would ensure that we always compare "apples to apples" and that some models won't outperform others just due to chance. Caret's *train* function allows running models with CV from various packages while using the same CV folds, and the list of models supported is substantial [2]. Also, using this function, we can tune hyperparameters either with the caret's pre-specified tuning grid, or using our own grid. Applying the caret's functionality allows us to unify approach to model fitting, so we used *caret::train* to train our models whenever possible.

Model-wise, we could not find R implementations of the best performing threshold models recommended by Gutierrez et al. (2015) such SVOREX and SVORIM. We also found it difficult to implement approaches with nominal and ordinal decompositions in R, since most methods we've learnt don't support such decompositions in Y (apart from SVMs which do one-vs-one decompositions when Y has more than two classes, fit k(k-1)/2 binary classifiers and make prediction using majority voting [4]). As a result, we considered and fitted the following models with the use of *caret::train* unless otherwise specified:

- **Ordinal regression with elastic net penalty** (*ordinalNet::ordinalNet, method = 'ordinalNet'*)[1]. Using a penalized model allows us not to exclude irrelevant features manually. We trained models with proportional (OrdReg.POM) or non-proportional odds (OrdReg.nonPOM), logit and probit link functions and various values of a mixing parameter α. For the best model in CV, we refit it twice allowing PCA pre-processing and scaling/centering of predictors. This did not impact the accuracy of the model with non-proportional odds, but results were slightly better with PCA pre-processing in POM. Since these models had low accuracy vs other models and were expensive to train, we did not pursue them further, e.g., trying log transformations of X, using square terms and interactions etc.
- **SVMs** (*kernlab::ksvm, methods = {'svmLinear', 'svmRadial', 'svmPoly'}*). We tried **linear and radial kernels** with various costs, and a limited number of costs for a polynomial kernel (degree 2 or 3).
- **CART on ordinal responses** (*rpartScore::rpartScore, method = 'rpartScore'*). It allows cost-based learning *"where the misclassification costs are given by the absolute or squared differences in scores assigned to the categories of the response. Pruning is based on the total misclassification rate or on the total misclassification cost."* [5] We tried models with absolute and squared misclassification costs, along with total misclassification rate or total misclassification cost as a pruning criterion. However, this model did not perform well overall.
- **Random forests** (*ranger::ranger, method = 'ranger'*). We tuned a hyperparameter *mtry* along with a splitting criterion – Gini index or extratrees ("extremely randomized trees" [7,8]). Along with a non-weighted RF, we also trained a RF with class weights inversely proportional to the class shares (this allows cost-sensitive learning, but documentation lacks details). This did not lead to a significant change in the model performance. **RF performed the best in this task and was one of the fastest.**

Figure 1. Comparison of accuracy of the best performing models in each class



- **Ordinal forests** (*ordinalForest::ordfor*, not supported by *caret::train* and not shown on Figures). Due to high running times, we experimented with hyperparameter tuning on the training/test split and found a value of mtry=5 to be a good one. Among two performance functions, "equal", which ensures each class has approximately the same accuracy irrespective of its size, and "proportional", which aims to classify correctly the maximum number of observations, the latter performed better. We ran the best model in a loop using the same CV folds as for other models. Despite applying a more sophisticated algorithm than RF forests, **an ordinal forests' accuracy was 2pp lower than in RF.**
- **eXtreme Gradient Boosting** (*xgboost::xgboost, method = {'xgbTree', 'xgbLinear'}*). We trained it with a tree as a base learner and tuned a learning rate, trees number, and a tree depth. Trees with low

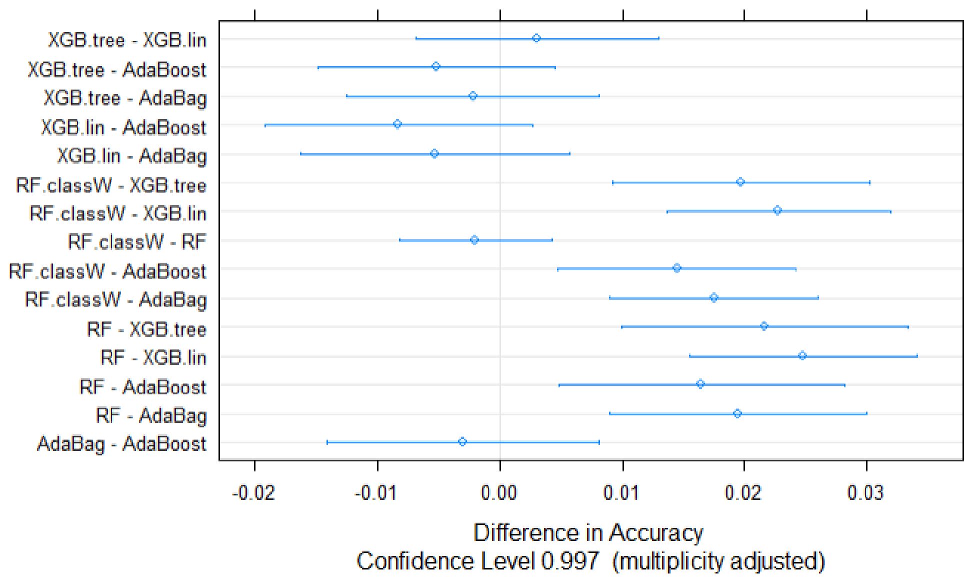
[1] In parentheses, we provide the name of the package and the function used and the method corresponding to that function in caret::train.

max depth (1-2) had limited accuracy, even if we increased the number of trees or changed the learning rate. Trees with higher max depths (6-8-10) performed better. XBG.tree on a Figure 1 had nrounds = 250, max_depth = 8, eta = 0.05. We also trained a model with a linear booster (XGB.lin) and a tuning grid from *caret::train*. It performed almost the same as XGB.tree.

- **AdaBag** (*adabag::bagging, method = 'AdaBag'*) and **AdaBoost** (*adabag::boosting, method = 'AdaBoost.M1'*) performed similar to XGBoost models. For AdaBag, the best model had 150 trees and maxdepth = 15. For **AdaBoost**, we tuned the number of trees (130) and a max tree depth (10) while using the learning coefficient type of 'Freund'.
- **Naive Bayes** (*naivebayes::naive_bayes, method = 'naive_bayes'*): We used the kernel to allow for estimating the class conditional densities of continuous predictors and we tuned its width.

All models that accounted for ordinal information performed worse than ones that treated Y as categorical. This might be because our Y has just 3 categories and there is not much information hidden in this order. To select the best model, we use a *caret::diff* function to run a t-test to compare accuracy across resamples for top-6 models from the Figure 1, using Bonferroni adjustment for multiple comparisons. Figure 2 shows test results: RF models score significantly higher than XGBoost, AdaBag and AdaBoost, while there is no difference between RF fit with and without class weights. We select a non-weighted RF as our best model. We also compare its accuracy to that obtained with the majority voting by top-5 models from the figure 1, and RF outperforms (66.51% vs 66.09%). So, we use RF as the final model to make predictions on the unseen data. This model has the following parameters: mtry = 2, splitrule = "extratrees" and min.node.size = 5.

Figure 2. Testing differences in top-6 models' resamples accuracy



References

[1] Gutierrez, P. A., Perez-Ortiz, M., Sanchez-Monedero, J., Fernandez-Navarro, F., & Hervas Martinez, C. (2015). Ordinal regression methods: survey and experimental study. IEEE Transactions on Knowledge and Data Engineering, 28(1), 127-146.
[2] Models: A List of Available Models in train. https://rdrr.io/cran/caret/man/models.html
[3] Max Kuhn. The caret Package. 2019-03-27. http://topepo.github.io/caret/index.html
[4] Package 'kernlab'. November 12, 2019. https://cran.r-project.org/web/packages/kernlab/kernlab.pdf
[5] Package 'rpartScore'. February 20, 2015. https://cran.r-project.org/web/packages/rpartScore/rpartScore.pdf
[6] Package 'ranger' January 10, 2020. https://cran.r-project.org/web/packages/ranger/ranger.pdf
[7] ranger: A Fast Implementation of Random Forests. https://github.com/imbs-hl/ranger
[8] Geurts, P., Ernst D., Wehenkel, L. Extremely randomized trees. Mach Learn (2006) 63: 3–42. https://link.springer.com/content/pdf/10.1007/s10994-006-6226-1.pdf