Homework 5 - An analysis of Wines

Giacomo Sanna

12/15/2019

What makes a wine good?

Introduction

Discaimer: *for this assignment I use a somewhat informal language, as the main intended audience of this output are my classmates. This is meant to be a practical example of what can be achieved with the tools learnt during the "Decision Trees and Ensemble Methods" part of the course.*

For the "Decision Trees and Ensable methods" part of the course, I decided to analyze a dataset on wine that I obtained from *WineEnthusiast* (www.winemag.com), a website and magazine that specializes in wines, collecting ratings and reviews and publishing articles and guides. The dataset contains about 250'000 reviews.

The idea here is to apply different models and try to predict the rating that a certain wine will get, based on a set of observed characteristics. The dataset is large, but before we can start to play with it, we need to have a look at the data:

alcohol	category	country	price	rating	varietal	vintage
13.2	Red	US	185	95	Cabernet Sauvignon	2015
15.0	Red	Italy	80	94	Red Blend	2013
14.1	Red	US	62	94	Pinot Noir	2016
14.1	White	US	50	94	Chardonnay	2016

The table shows an example of the information available in the dataset. More columns related to the region (eg: California), subregion (eg: Napa), subsubregion (eg. St.Helena), and even designation, winery, description, weblink are available.

The first thing we should be concerned when working with a big dataset is the presence of missing values. After a simple exploratory analysis we notice quite a few missing values, especially in the columns related to subregion and subsubregion. For the purpose of this preliminary analysis, I decided to exclude these variables, as we already have information regarding the origin of the wine from the variables country and region. The variables that are excluded for the analysis are therefore: Subregion, Subsubregion, Title and Url.

To simplify our analysis further - in terms of computational power and time required - I decided to drop the observations whose frequency in terms of grape variety was less than 200. This is how you can do it:

```
# Keep only the observations whose frequency is more than 200
dataset <- as.data.table(dataset)[, N := .N, by = variety][N > 200]
```

Next, we check for the presence of possible outliers. After a quick search we can see that the majority of such outliers are due to data entry mistakes. Some notable examples are:

- in the variable **"ancohol"**, the year (eg. 2016) or the rating (eg. 90) was entered instead of the alcohol content. Given more time and resources, one could try to interpolate or research the alcohol content using the other information available. For this time, I decided to drop these observations.
- in the variable **"vintage"**, the year mentioned not always refers to the year of production, but instead to the name of the winery. We do not expect a wine from 1531 to cost 17 dollars: it's more likely that it comes from the Aimery Grand Cuvée 1531 winery. These observations are also dropped.
- in the variable **"price""**, a few wines reach very high values, but a quick search online confirms that the prices are legitimate (for example one bottle of *2005 Petrus Red Bordeaux Pomerol France* sells online for $ 3,774.58 ex. sales tax.

The final sample consists of 123'110 observations.

Model Selection

In the framework of model selection and prediction, there are many interesting things that could be attempted with this dataset: for example we could try to predict the country of origin of the wine, given its price, rating and type of grapes used; or we could try to predict the price, given the alcohol content and year of production. For this analysis, I will try to predict the rating that the user gave to the wine, using some of the variables available in the set.

The variable in question is named *"rating"*:

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
##   80.00   86.00   89.00   88.61   91.00  100.00
```

In addition, I created a few additional variables that I will introduce during this assignment, to overcome the computational limits of my poor laptop and the limitations of the different models. To summarize, a total of 4 new variables were created to simplify the analysis:

- **decade**: In which decade was the wine produced?
- **type**: Classifying the wine based on the alcoholic content;
- **variety**: Grouping the varietals of the wines in 52 levels;
- **dscrpt_length**: Length of the review;

In particular, I was really interested in incorporating the variable "description" into my analysis. I found an article online performing a similar analysis (see my references for more information), using *text vectorization*, which consists in learning a way to assign weights to different words in order to improve the prediction results; this goes beyond my knowledge at the moment, so I limited myself to incorporating the "length" of the descriptions; assuming that a good rating would also be more likely to get a longer review.

The models we will try to implement are, in order, the following:

1. Regression tree
2. Random forest regression
3. Boosted regression tree
4. Classification forest
5. Boosted classification tree

Training and Test set

We set aside a test set of roughly 37'000 observations (30% of the total set) and we further split the training set by holding out a validation set of roughly 26'000 observations (again, 30% of the remaining set, excluding the test set). I decided to have a bigger training set because I do not know a priori how these models will fit the data, I am more concerned with finding a good model.

Model 1: Decision tree

The decision tree is the first model we will try. The variables involved are reported in the line of code below:

```
# Decision tree
fit_regtree = rpart(formula = rating ~ dscrpt_length + price + variety + type
                      + country + decade,
                    data = train_set,
                    control = rpart.control(minsplit = 10),
                    method = "anova")
```

Compared to the code available in the slides, I added a new parameter: *"minsplit"*, which specifies the minimum value of observations that should be included before the algorithm attemps to perform another split; without this paramer we could end up making too many splits (even 1 split for each observation!)

Another interesting parameter is *"minbucket"* which specifies how many observations are required to be included in the terminal nodes, a higher number implies a lower tree depth and may be faster to calculate. I will not use it here, but I will mention something similar when modeling a random forest on the data.

The complexity parameter table

As we saw in class, sometimes the model can be improved by pruning the tree, using the parameter cp. This is the code that can be also found in the lecture notes.

```
# How to prune a tree: what these lines of code do
fit_regtree_pruned <- prune(fit_regtree,
                            cp=fit_regtree$cptable[which.min(
                              fit_regtree$cptable[,"xerror"]),"CP"])
```

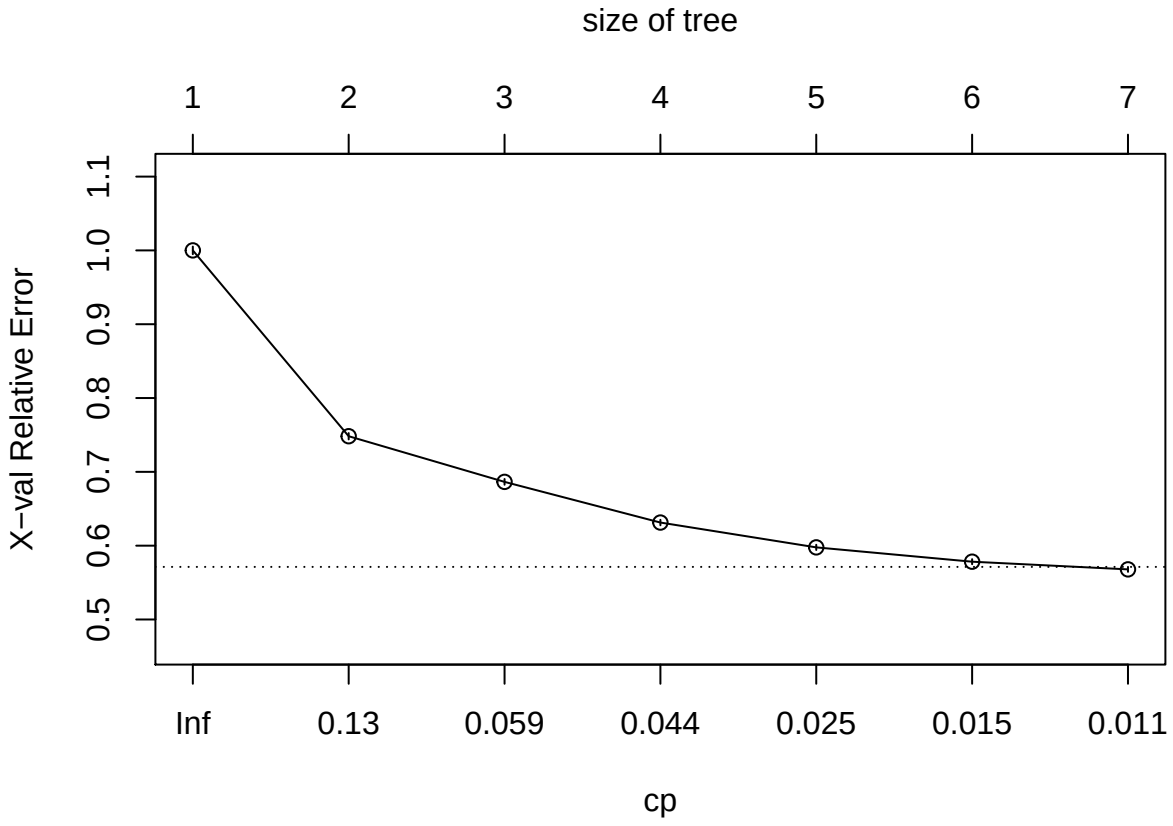
But what is the cp parameter?

Once we fit the model, we can get a lot of insights using the function printcp(), which returns the complexity parameter table. What the code does, is to pick the CP from the table below, from the row related to the split that has achieved the lowest cross validation error (xerror).

CP	nsplit	rel error	xerror	xstd
0.2517979	0	1.0000000	1.0000203	0.0051884
0.0630704	1	0.7482021	0.7482537	0.0042849
0.0551264	2	0.6851317	0.6864228	0.0038869
0.0344543	3	0.6300052	0.6313014	0.0037094
0.0182777	4	0.5955510	0.5976662	0.0035380
0.0122542	5	0.5772733	0.5782681	0.0034276
0.0100000	6	0.5650191	0.5678719	0.0033568

In short, the *cp parameter* is the amount by which an additional splits improves the relative error. To make a very simple illustration, compare the rel errors in the first two rows. The amount the error changed is given by the CP parameter in the first row.In other words, the algorithm stops the splitting when the relative error (rel error) does not improve by "cp" anymore.

Another way to see how the CP evolves with each number of splits can be seen in the following graph, obtained with the function plotcp():



But in this case, as the lowest CP is obtained in the last split, there is no point in pruning the tree: we would obtain the same results we already got. Pruning would choose the CP that we already have used.

What is the accuracy of the result obtained in the validation set? if we recall the that the MSE measures how much the predicted values are different from the actual values, then $R^2 = 1 - MSE/Variance$ is a simple way to report the prediction accuracy.

	x
MSE|err	5.8746751
R2|Accuracy	0.4348532

Model 2: Random Forest regression

Using the same regressors, I will next attempt to model a random forest regression.

Before the implementation of the model, I realized that there is a limit of factor levels a variable can contain (not more than 53) in order to run a random forest regression, and this would have meant dropping grape varieties as originally there were more than 800 kinds present in the set.

Instead, the variable variety was regrouped by allowing similar grapes to belong to the same group. As already mentioned in the introduction, the varieties that had a frequency of less than 200 were discarded, allowing us to retain this important variable in our analysis.

As a side note, we cannot encode the variable in a different way, for example using as.character(), because including them in the model in this way would return an error

```
# Random Forest regression
fit_regforest <- randomForest(rating ~ dscrpt_length + price + variety
                              + type + country + decade,
                              data = train_set, mtry = 3, ntree = 550,
                              nodesize = 250 )
```

The parameters chosen for this model are:

- *mtry*: the number of predictors that are randomly chosen for each split. Given that we are working with 6 regressors, I set mtry = 3 (if nothing is specified, the function uses p/3 as default);
- *ntree*: the number of trees to grow is 550, a bit more than the default 500;
- *nodesize*: 250 observations should be included at the terminal nodes. Unfortunately given the time required for the calculations, I was not able to reduce the size further. In fact, as we will see soon, a smaller node size in this case does not seem to improve the prediction accuracy by much.

The following table shows the prediction accuracy obtained in the validation set by setting different node size:

mtry	n.tree	nodesize	R2
3	500	10000	0.4943554
3	500	5000	0.5415327
3	500	2500	0.5671098
3	500	2000	0.5735710
3	500	1500	0.5805930
3	500	1000	0.5893330
3	500	500	0.6000310
3	500	250	0.6070550
3	550	250	0.6071890

As we can see from the table, setting a node size lower than 500 does not improve much the results, so due to computational limits of my computer I will not go over 550 trees. We therefore pick the last row as our best hypothesis for this model.

Model 3: Boosting a regression tree

Can we improve the results further? As we saw in class, another method that we can try is the boosting algorithm. As the the random forest model seems to perform better until this point, I decided to include the boosting method for regression trees. Another reason to do boost regression trees in this case is that in class we only covered boosting for classification problems and I wanted to learn how to apply the method to regression decision trees.

```
# Boosted regression tree
boost.fit_regforest <- gbm(rating ~ dscrpt_length + price + variety + type
                           + country + decade,
                           data = train_set, distribution="gaussian",
                           interaction.depth = 5, n.trees = 500, shrinkage = 1 )
```

Compared to boosting a classification tree, the difference is that we use as one of the parameters the distribution "gaussian" (instead of "adaboost", which assumes an exponential loss for 0-1, not adequate for this case).

Unfortunately I was not able to tune the parameters (shrinkage, depth) using cross validation (as shown in the code available in the lecture slides), because it would cause my computer to crash; alternatively, I settled for checking different values of the parameters manually.

depth	n.tree	shrink	MSE	R2
5	500	1.00	4.517692	0.5653957
4	500	1.00	4.430109	0.5738213
3	500	0.01	4.394387	0.5772577
4	500	0.01	4.306369	0.5857250
5	500	0.01	4.306369	0.5857250
3	500	1.00	4.286590	0.5876278
3	500	0.10	4.072724	0.6082018
4	500	0.10	4.041080	0.6112460
5	500	0.10	4.018742	0.6133950

depth	n.tree	shrink	MSE	R2
5	500	0.01	4.011347	0.6141064
5	500	0.10	4.004670	0.6147487
5	500	1.00	3.994215	0.6157545

The table just shown was sorted to present the hypotheses in ascending order of accuracy. The last row is our final hypothesis. We have been able to improve the accuracy of the regression tree, and even surpass the results obtained with the random forest, even if not by much.

Model 4: Random Forests of classification trees.

I was curious to see how a a random forest of classification trees would perform in this setting, but unfortunately it is not possible to use the dependent variable *"rating"* directly. The variable was re-encoded and renamed *"rank"*, classifying different wines in 4 groups based on their rating:

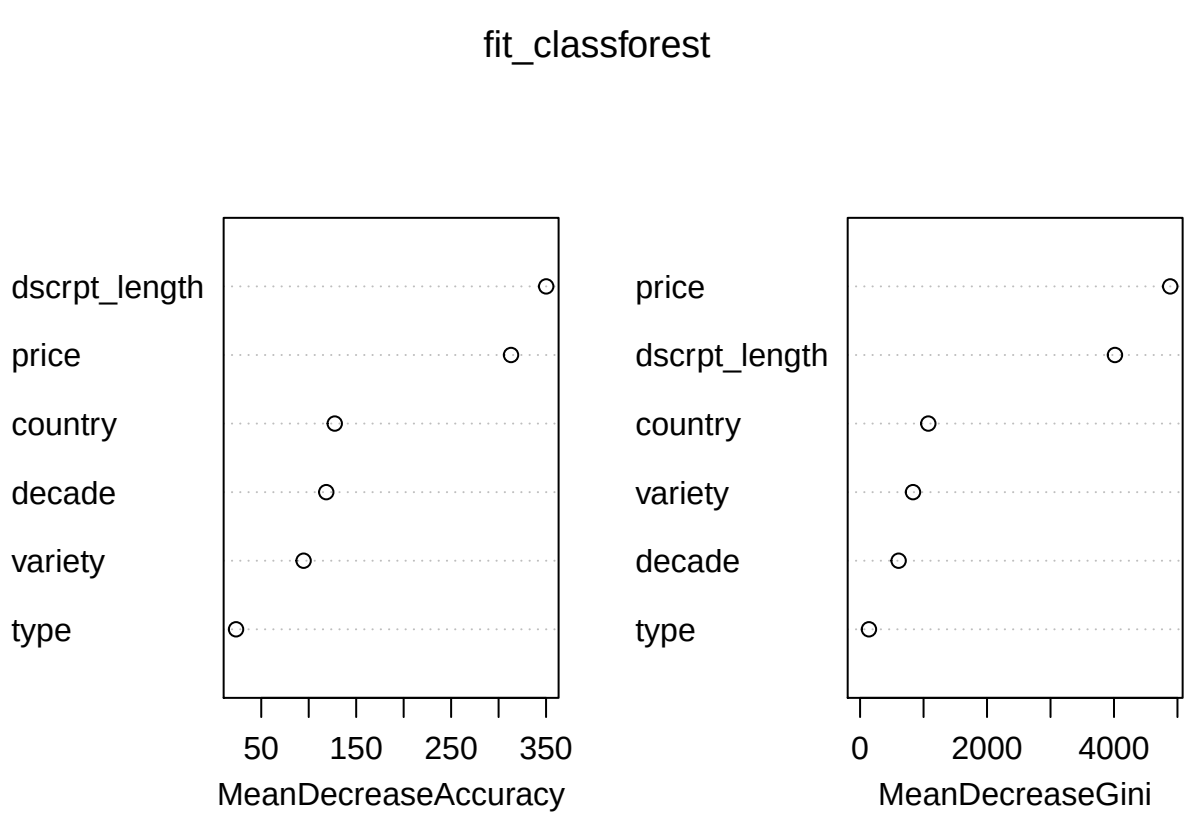
- "Excellent" wine: wines whose rating was between 96 and 100;
- "Good" wine: wines whose rating was between 91 and 95;
- "Average" wine: wines whose rating was between 86 and 90;
- "Low quality" wine: wines whose rating was between 80 and 85.

```
# Classification random forest
fit_classforest <- randomForest(rank ~ dscrpt_length + price + variety
                                + type + country + decade,
                                data = train_set, mtry = 3, ntree = 550,
                                nodesize = 250, importance = T)
```

The regressors used in this analysis are the same. To calculate the accuracy we use the mean difference between the predicted target observation and the actual observation. For the implementation, we used the same parameters used for the regression random forest. Like in the previous case, the following table shows the prediction accuracy (as 1 - error) obtained in the validation set by setting different node sizes:

mtry	n.tree	nodesize	accuracy
3	500	10000	0.6352418
3	500	5000	0.6432108
3	500	5000	0.6432108
3	500	250	0.6888975
3	550	250	0.6885106

An interesting insight can be drawn using varImpPlot(), included in the randomForest package. The graph below plots the variable importance in terms of both accuracy and gini.



The Confusion matrix

Another useful tool in the analysis of the results is the confusion matrix. Besides the accuracy, from the table we can derive the sensitivity, specificity and precision. The table is obtained with the function confusionMatrix(), from the caret package.

```
## Confusion Matrix and Statistics
##
##              actual
## predicted     Low Quality Average  Good Excellent
##   Low Quality        1961     788     5         0
##   Average            2553   11196  2616         4
##   Good                 69    1783  4634       241
##   Excellent             0       0     0         0
##
## Overall Statistics
##
##                Accuracy : 0.6882
##                  95% CI : (0.6826, 0.6939)
##     No Information Rate : 0.5326
##     P-Value [Acc > NIR] : < 2.2e-16
##
##                   Kappa : 0.4539
```

```
##
##  Mcnemar's Test P-Value : NA
##
## Statistics by Class:
##
##                      Class: Low Quality Class: Average Class: Good
## Sensitivity                     0.42789         0.8132      0.6387
## Specificity                     0.96271         0.5719      0.8874
## Pos Pred Value                  0.71206         0.6840      0.6889
## Neg Pred Value                  0.88647         0.7288      0.8629
## Prevalence                      0.17729         0.5326      0.2807
## Detection Rate                  0.07586         0.4331      0.1793
## Detection Prevalence            0.10654         0.6332      0.2602
## Balanced Accuracy               0.69530         0.6926      0.7631
##                      Class: Excellent
## Sensitivity                  0.000000
## Specificity                  1.000000
## Pos Pred Value                    NaN
## Neg Pred Value               0.990522
## Prevalence                   0.009478
## Detection Rate               0.000000
## Detection Prevalence         0.000000
## Balanced Accuracy            0.500000
```

You may notice that the matrix is very similar to the table we saw in class while discussing the "Classification and discete choice models", the only difference is that it the rows and columns are transposed. I am not sure why. Maybe nobody knows. Maybe this is why it's called confusion matrix.

Model 5: Boosting a classification tree

Finally, the last method we try is a boosted classification tree.

```
# Boosted classification tree
boost.fit_classtree <- gbm(rank ~ dscrpt_length + price + variety
                           + type + country + decade,
                           data = train_set,
                           distribution="multinomial", interaction.depth = 4,
                           n.trees = 500, shrinkage = 0.1 )
```

The difference with what we have seen in class is that the model here is multiclass. I was surprised to find out that in this case, the returned prediction is not exactly a "class", but the probability of falling withine each class. Here we can have an example of some of the observations.

	Low Quality.500	Average.500	Good.500	Excellent.500
1	0.0057524	0.0577301	0.8477901	0.0887274
38	0.0278024	0.6177195	0.3514275	0.0030506

	Low Quality.500	Average.500	Good.500	Excellent.500
53	0.0012674	0.0364383	0.2270806	0.7352137

In order to calculate the error I will assume that the model picks the category with the highest probability and then confront it to the actual value. After some data wrangling, we obtain the following results:

depth	n.tree	shrink	err	Accuracy
3	3	0.01	0.3228627	0.6771373
3	3	0.10	0.3099420	0.6900580
3	4	0.10	0.3095938	0.6904062
3	3	0.01	0.3095938	0.6904062
3	3	0.10	0.3089749	0.6910251

The final model

The following table contains the results obtained with the 5 models.

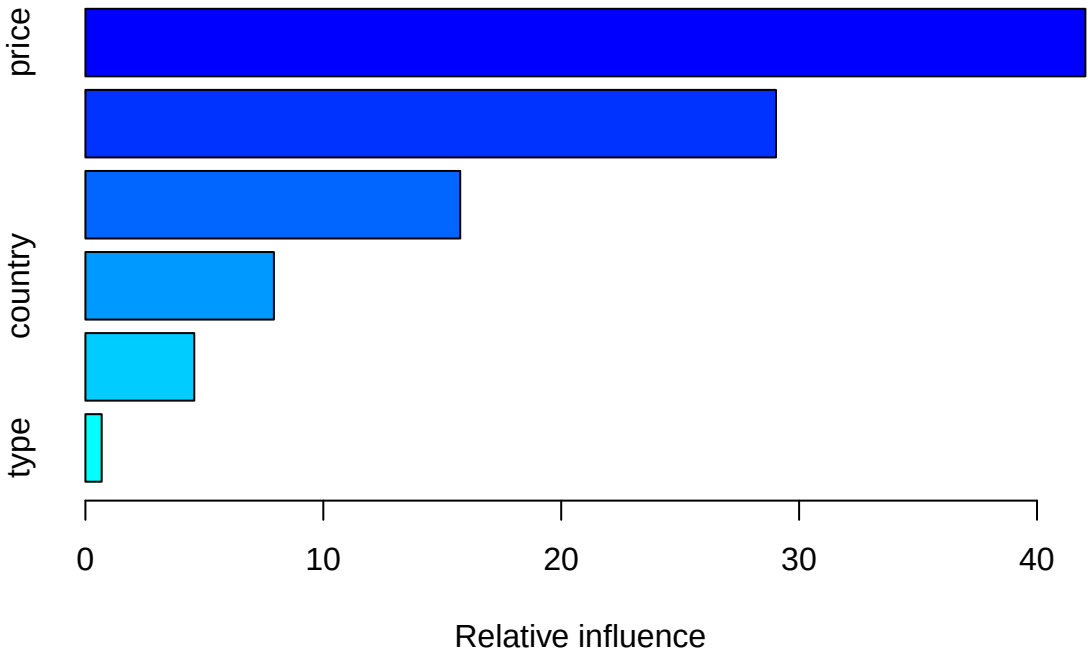
	MSE|err	R2|Accuracy
Regression Tree	5.8746751	0.4348532
Reg.Random Forest	4.0830787	0.6072057
Boosted Reg. Tree	3.9942150	0.6157545
Class Random Forest	0.3114894	0.6885106
Boosted Class. Tree	0.3089749	0.6910251

The best performing model is the boosted classification tree. Now that we have selected the best model, the training set and validation sets are combined, and the selected model is refitted. The performance of our new final hypothesis g** is tested on the test set. This will give us our estimate of the E_{out}.

```
# Refitted model
retrain <- rbind.data.frame(train_set, val_set) #combine train and validation sets

refit <- gbm(rank ~ dscrpt_length + price + variety + type
             + country + decade,
             data = retrain,
             distribution="multinomial", interaction.depth = 3,
             n.trees = 500, shrinkage = 0.1 )
```

In the table below we can see the variables in order of relative importance according to the refitted model:



	var	rel.inf
price	price	42.034026
dscrpt_length	dscrpt_length	29.028713
variety	variety	15.754527
country	country	7.923715
decade	decade	4.575400
type	type	0.683619

Here are the final results.

Error	Accuracy
0.3026587	0.6973413

Conclusions

For this assignment we tried many of the models introduced in class, and we applied to a dataset of wines in order to predict the rating a wine receives based on a set of variables such as the age of the wine, its price, origin.

Even though the final prediction accuracy is not great (almost 70%), I was surprised by the gain obtained by the random forest model compared to the decision tree model, and then again by using instead a random forest of classification decision trees. Many observations were dropped due to the physical limitations of my computer, with a more powerful machine it could be possible to work on a sample almost twice the size and to obtain much more accurate results.

An interesting development would be to perform text vectorization on the actual review the wine received, and also on the wine designation (more than 60'000 different designations are available on the dataset but where not used for this analysis).

References

Greg Ridgeway (2019). Generalized Boosted Models: A guide to the gbm package

https://www.kaggle.com/zynicide/wine-reviews/version/4

https://towardsdatascience.com/wine-ratings-prediction-using-machine-learning-ce259832b321

https://medium.com/@hinasharma19se/exploration-of-wines-6c9ddaa3271

https://github.com/activatedgeek/winemag-dataset

https://www.winemag.com/2019/01/02/wine-vintage-chart-2019

https://cran.r-project.org/web/packages/text2vec/vignettes/text-vectorization.html

https://cran.r-project.org/web/packages/tokenizers/index.html

https://mran.microsoft.com/snapshot/2016-02-06/web/packages/text2vec/vignettes/text-vectorization.html

https://towardsdatascience.com/wine-ratings-prediction-using-machine-learning-ce259832b321

https://www.kaggle.com/olivierg13/wine-ratings-analysis-w-supervised-ml

https://medium.com/@hinasharma19se/exploration-of-wines-6c9ddaa3271

https://machinelearningmastery.com/machine-learning-evaluation-metrics-in-r/