Evaluating and Interpreting Ordinal Forests (Demo Project)

Gerdis Anderson

2022-09-07

Required Packages

```
library(ordinalForest)
library(verification) #for RPS calculation
library(pdp) #partial dependence calculation/visualization
library(ggplot2) #fancy plotting options
```

Data

For this demo project I used the "red wine" subset of the Wine Quality Data Set. It describes samples of Portuguese *Vinho Verde* that were rated with respect to sensory quality. Each sample is described in terms of 11 physicochemical parameters. Quality is given as a score between 0 and 10.

The dataset is imbalanced with classes 0, 1, 2, 9 and 10 completely absent, and only a few instances of classes 3 and 8.

I used the SMOTE method to create a balanced distribution of classes 3 - 8 in the training data. For the steps taken during preprocessing see *data/preprocess.R*

```
cC <- c("character", rep("numeric", 11), "factor")
traindata <- read.table(here("data/preprocessed_data/OF_traindata.dat"),
             colClasses = cC)
testdata <- read.table(here("data/preprocessed_data/OF_testdata.dat"),
            colClasses = cC)
str(traindata)
```

```
## 'data.frame':    324 obs. of  12 variables:
##  $ fixed.acidity       : num  11.6 11.02 6.14 8.44 9.3 ...
##  $ volatile.acidity    : num  0.41 0.496 0.994 0.404 0.48 ...
##  $ citric.acid         : num  0.54 0.606 0.04 0.406 0.29 ...
##  $ residual.sugar      : num  1.5 4 1.74 2.2 2.1 ...
##  $ chlorides           : num  0.095 0.0786 0.0689 0.0702 0.127 ...
##  $ free.sulfur.dioxide : num  22 5.54 24.51 8.01 6 ...
##  $ total.sulfur.dioxide: num  41 17.6 74.8 17.6 16 ...
##  $ density             : num  0.997 0.998 0.995 0.996 0.997 ...
##  $ pH                  : num  3.02 3.19 3.52 3.35 3.22 ...
##  $ sulphates           : num  0.76 0.672 0.548 0.728 0.72 ...
##  $ alcohol             : num  9.9 12.3 9.87 11.95 11.2 ...
##  $ quality             : Factor w/ 6 levels "1","2","3","4",..: 5 6 2 6 3 6 5 6 5 3 ...
```

Models

Using the dataset described above, I train two simple ordinal forest models which predict sensory quality of wine samples from their physicochemical parameters. For the first model, *forest1*, I use the default settings for class probability prediction. For the second model, I force the algorithm to consider the variable *residual.sugar* at every split in additon to the *mtry* candidate parameters for splitting.

```
set.seed(123)
forest1 <- ordfor(depvar="quality", data=traindata, perffunction="probability")

forest2 <- ordfor(depvar="quality", data=traindata, perffunction="probability",
                  always.split.variables = c("residual.sugar"))
```

Tasks

- measure how well each model predicts sensory quality of unseen samples. The error measure should reflect the ordinal nature of the response variable; that is, predictions that are very far from the observed quality should be penalized more harshly.
- compare model performance
- rank the predictors by importance
- for the better model, analyze how the most important variable affects predicted wine quality

Performance Evaluation

ranked probability score (RPS)

For each sample in the test dataset, predict the probability of each of the 6 observed quality scores.

```
predicted_quality <- predict(forest1, newdata=testdata)$classprobs
```

Compare that against the actual observed quality, and calculate the ranked probability score (RPS). Note that the RPS as a performance metric is not implemented in *ordinalForest*, but has to be imported via the *verification* package.

```
observed_quality <- testdata$quality
performance1 <- rps(obs=observed_quality, pred=predicted_quality)
performance1$rps
```

```
## [1] 0.08417765
```

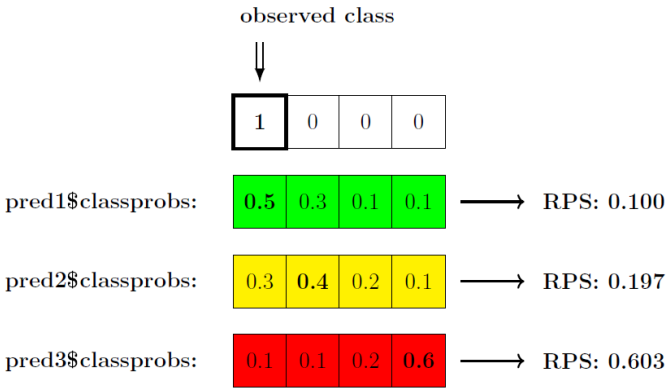
Now, what does that mean? Is that a good score? What even is the RPS?

The RPS was originally developed as a scoring system for weather forecasts. For the mathematical derivation see Epstein (1969). In short, the RPS does exactly what you'd expect from a performance measure for ordinal regression: it penalizes errors more harshly if classes that are assigned high probabilities are far from the observed class on the ordinal scale. The RPS can take values between 0 and 1. *verification/ordinalForest* use the ***negative*** RPS, with smaller values signifying better performance. It is defined as:

$$RPS = \frac{1}{M-1}\sum_{m=1}^{M}\left[\left(\sum_{k=1}^{m} p_k\right) - \left(\sum_{k=1}^{m} o_k\right)\right]^2,$$

where M is the number of classes, p_k is the predicted probability of class k, and o_k is an indicator for the observation in class k (0=no, 1=yes).

Consider the following example, where the RPS is calculated for a single observation and class probability predictions *pred1$classprobs, pred2$classprobs, and pred3$classprobs* made by 3 different models. Note that the observation was one-hot encoded merely for illustrative purposes; the obs argument of rps() expects a simple vector of factor levels.



As can be seen, the RPS values get larger (worse) with growing distance on the ordinal scale between the observed class (1) and the class(es) with the highest predicted probabilities. *pred2$classprobs* still receives a score pretty close to 0, although the class with the largest predicted probability != 1, whereas *pred3$classprobs* receives a score > 0.6.

ranked probability skill score

To say that a score is "small" or "close to 0" is still pretty vague. A more objective way to assess the performance of a model is to compare it to a baseline. Conveniently, the rps() function also calculates the ranked probability skill score (RPSS) which measures relative improvement of a prediction over a reference prediction:

$$RPSS = 1 - \frac{\overline{RPS}}{\overline{RPS}_{reference}}$$

The RPSS can take values between minus infinity and 1. 0 indicates no improvement over the reference, and negative values indicate that the predictions made by the model are worse than the reference. By default, reference predictions are calculated based on the distribution of the classes in the test data, but a user-defined vector of predictions can be passed to the rps() function via the baseline argument.

Let's take a look at the RPSS of *forest1* with the default baseline.

```
performance1$rpss
```

```
## [1] 0.003795923
```

We can see that *forest1* only has a minimal performance advantage over a model which relies solely on class distribution. This is disappointing, but not really surprising given that the original dataset is severely imbalanced, with very few examples in some classes, and we resampled only the training data. Now let's see how *forest1* does compared to a different baseline model which predicts zero probability for all but the two most frequent classes.

```
abs_frequencies <- table(testdata$quality)
abs_frequencies
```

```
##
##   3   4   5   6   7   8
##   4   8 138 128  40   3
```

```
performance1_bl <- rps(obs=observed_quality, pred=predicted_quality,
                       baseline=c(0.0, 0.0, 0.6, 0.4, 0.0, 0.0))
performance1_bl$rpss
```

```
## [1] 0.08440543
```

The RPSS is still very small, but about an order of magnitude larger than the one for the default baseline, meaning *forest1* has a larger performance advantage over this new dummy/baseline model.

Note that the RPS of the baseline itself can be accessed via the $rps.clim value:

```
performance1$rps.clim
```

```
## [1] 0.0844984
```

Now let's see how *forest2* performs, and compare it to the first model.

```
predicted_quality2 <- predict(forest2, newdata=testdata)$classprobs
performance2 <- rps(obs=observed_quality, pred=predicted_quality2)
the_rps <- c(performance1$rps, performance2$rps)
the_rpss <- c(performance1$rpss, performance2$rpss)
overview <- data.frame(the_rps, the_rpss)
row.names(overview) <- c("forest1", "forest2")
print(overview)
```

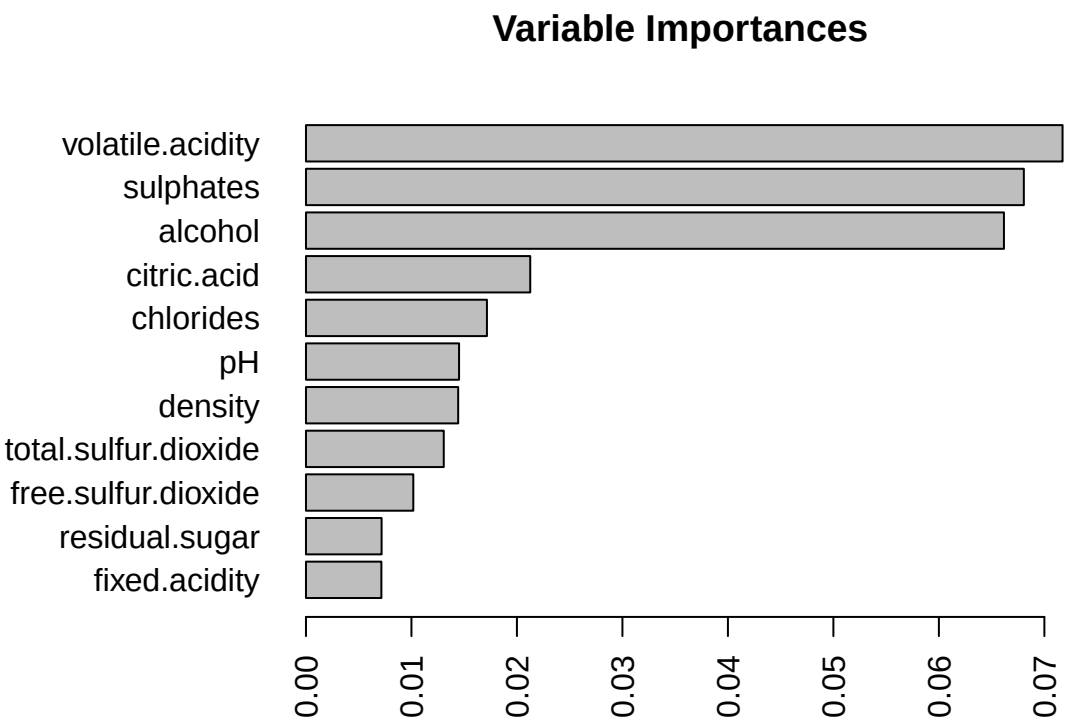
```
##            the_rps     the_rpss
## forest1 0.08417765  0.003795923
## forest2 0.08490509 -0.004812963
```

forest1 has a slightly smaller RPS than *forest2* and a positive albeit small RPSS, whereas the RPSS of *forest2* is < 0. We can infer that the model using only *mtry* randomly selected candidates at a given split has a small performance advantage over one that always considers *residual.sugar*. However (likely due to the severe imbalance in the test data), even the better model only has a small advantage over a dummy that simply relies on the class distribution to predict wine quality.

Variable Importances

Now that we have an idea of how well our models perform, let's calculate and visualize variable importances for the better one (variable importances are computed by performing random permutation of variables and measuring the resulting increase in RPS - the more important a variable is, the larger the decline in performance will be).

```
importances <- sort(forest1$varimp)
par(mar=c(5.1, 8.5, 4.1, 4.1), las=2)
barplot(importances, horiz=TRUE, main="Variable Importances")
```



We can see that the model relies most strongly on *volatile.acidity* to predict sensory quality, closely followed by *sulphates* and *alcohol*.

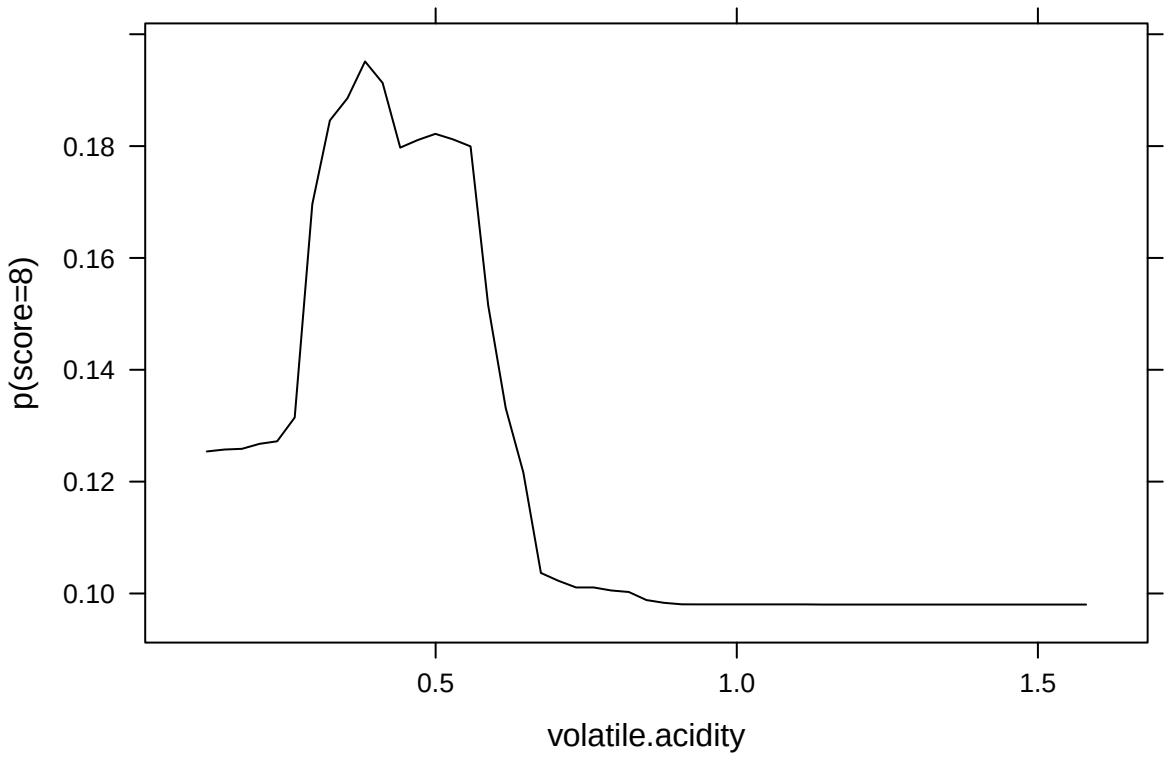
Partial Dependence Plots for Feature Effects

We don't just want to know that our model most strongly relies on *volatile.acidity* to predict sensory quality, but understand *how* this variable influences the predicted probability of the different quality scores.

Partial Dependence Plots provide a powerful tool for the visualization of feature effects in "black box" models. We're going to use the *pdp* package to calculate the partial dependences for *volatile.acidity* (simply put, the average reponse values for the different values of this predictor).

The *partial()* function from *pdp* needs instructions on how to compute the average responses of different parameter values with an ordinal forest object. If we just want the partial dependence of a single class, we define a function which computes the probability of said class for all observations, and returns the mean. When calling *partial()*, this function is passed via the pred.fun argument.

```
pred.best <- function(object, newdata){
preds <- predict(object, newdata = newdata)$classprobs
mean(preds[,6]) #preds[,1] corresponds to score = 3!
}
pdp::partial(forest1, type = "classification", pred.var = "volatile.acidity",
                    pred.fun = pred.best, which.class = 8,
                    train = traindata) %>% plotPartial(ylab = "p(score=8)")
```



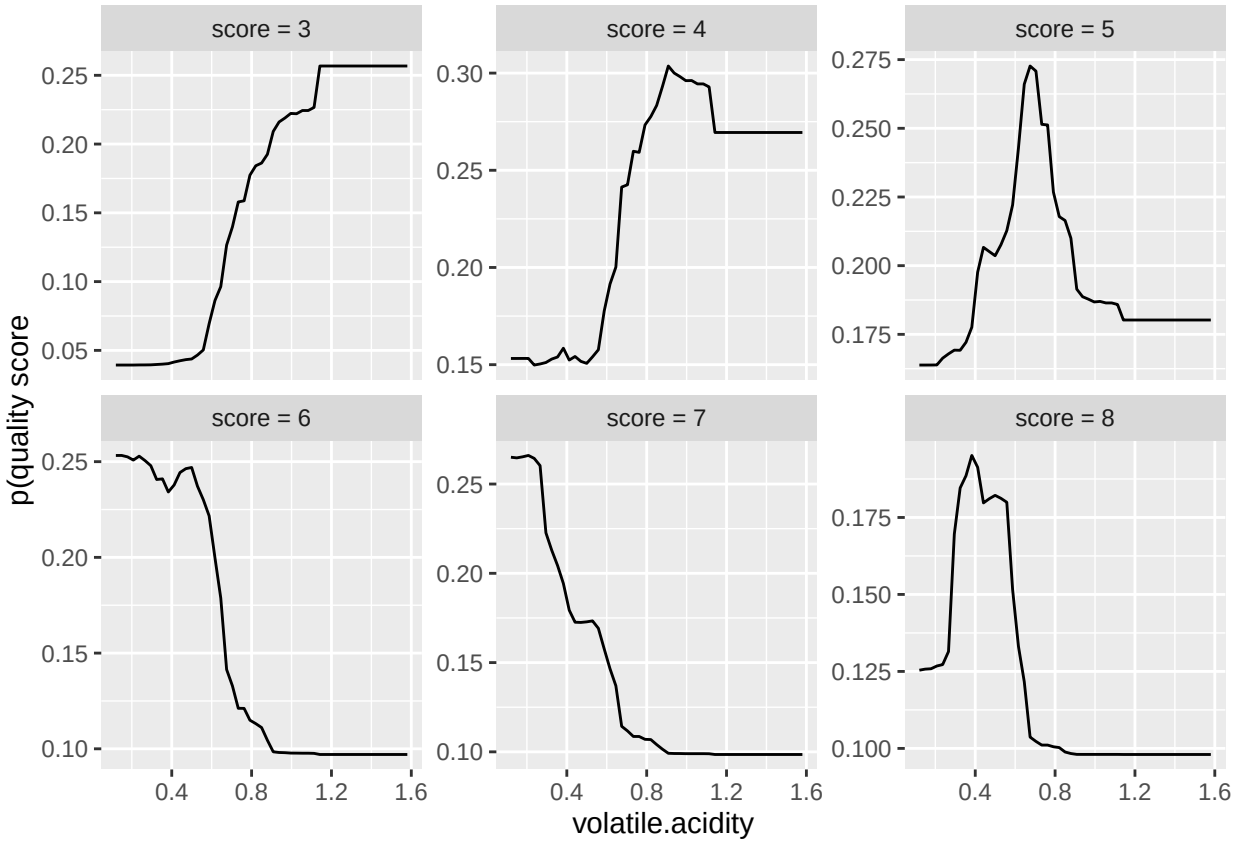
We can see that there is an increase in predicted probability of the best observed class (8) between *volatile.acidity* values of around 0.1 and 0.4, followed by a slight decline between circa 0.4 and 0.6, and a sharp decline between circa 0.6 and 0.7. Above *volatile.acidity* values of around 0.9, predicted probability stays pretty much constant at only around 0.1. We could gain more insight into the effect of this variable by displaying the partial dependences for all classes at once. For this purpose, we need a function which returns the average predicted probabilities of all classes.

```
pred.funall <- function(object, newdata){
preds <- predict(object, newdata=newdata)$classprobs
colnames(preds) <- c("score = 3", "score = 4", "score = 5",
                     "score = 6", "score = 7", "score = 8")
colMeans(preds)
}
pdps <- pdp::partial(forest1, type="classification", pred.var="volatile.acidity",
             pred.fun=pred.funall, train=traindata)
```

We now have 6 curves corresponding to the 6 observed quality scores. Each curve shows how the probability

of the corresponding class is affected by *volatile.acidity*. For a nice and neat display, I take advantage of *ggplot2* facets with a free y scale:

```
pdp_curves <- ggplot(pdps, aes(volatile.acidity, yhat)) +
              geom_line()
pdp_curves + facet_wrap(~yhat.id, scales="free_y") + labs(y="p(quality score")
```

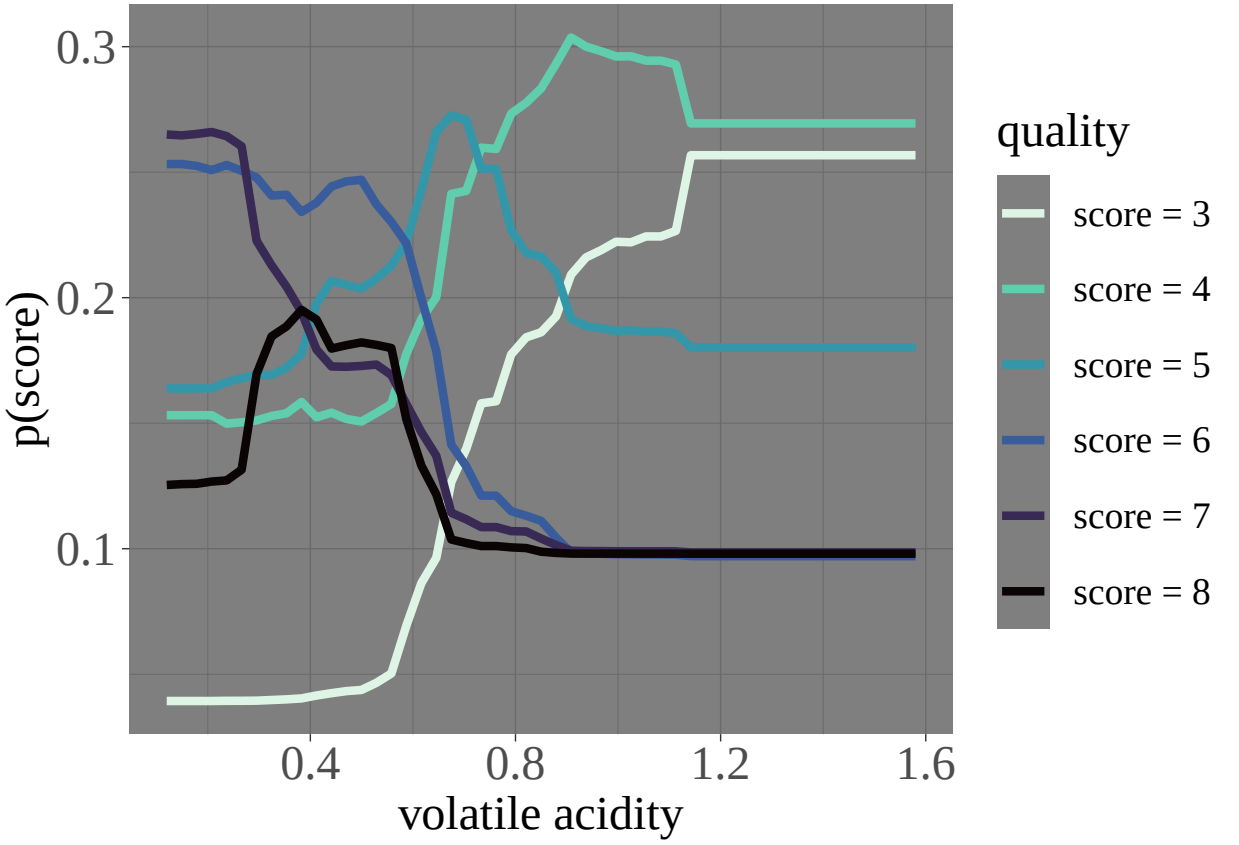


Obviously, these plots are not as intuitively interpretable as a single curve. However, what we can clearly see is that high *volatile.acidity* is associated with low sensory quality, medium levels of *volatile.acidity* are associated with medium quality, and low levels are associated with high quality. Interestingly however, the highest observed class is more likely to receive high predicted probabiliy when *volatile.acidity* is *rather* low than when it is *very* low.

When comparing the y-scales of the subplots, i.e. the ranges within which predicted probability varies, we can also see that the class corresponding to the lowest observed quality seems to have the strongest dependence on *volatile.acidity*. Predicted probability of this class varies between ~ 0.04 and ~ 0.26. For the class corresponding to the highest observed quality on the other hand, predicted probability only varies between ~ 0.1 and ~ 0.195, suggesting that the perception of high quality depends strongly on other variables.

We can also display all curves in one plot and use color to differentiate classes. Since the classes are ordered, I suggest using a sequential palette such as "mako".

```
legend_title <- "quality"
ggplot(pdps, aes(volatile.acidity, yhat, color=yhat.id)) +
  geom_line(size=1.5) +
  scale_colour_viridis_d(name=legend_title, option="mako", direction=-1) +
  theme_dark(base_family = "Times") +
  theme(legend.title = element_text(size=18), legend.text = element_text(size=14),
        axis.title=element_text(size=18), plot.title=element_blank(),
        axis.text=element_text(size=18),
        legend.key.height= unit(1.0, 'cm'),
        legend.key.width= unit(0.7, 'cm')) +
  labs(x="volatile acidity", y="p(score)")
```



This display shows us which class would be predicted for what level of *volatile.acidity* if no other variable were taken into consideration. Below ~ 0.3, it would be the second-best class (score=7), and between ~ 0.3 and ~ 0.5, it would be the class corresponding to a score of 6. Between ~ 0.5 and ~ 0.7, the model would predict a score of 5, and for all values above that, it would be the "second-worst" class observed in the data, which corresponds to score 4. Interestingly, neither the best nor the worst observed class would get predicted if only *volatile.acidity* were taken into consideration (for values > 1.1, however, the difference in predicted probability between the worst and second-worst class is minimal).

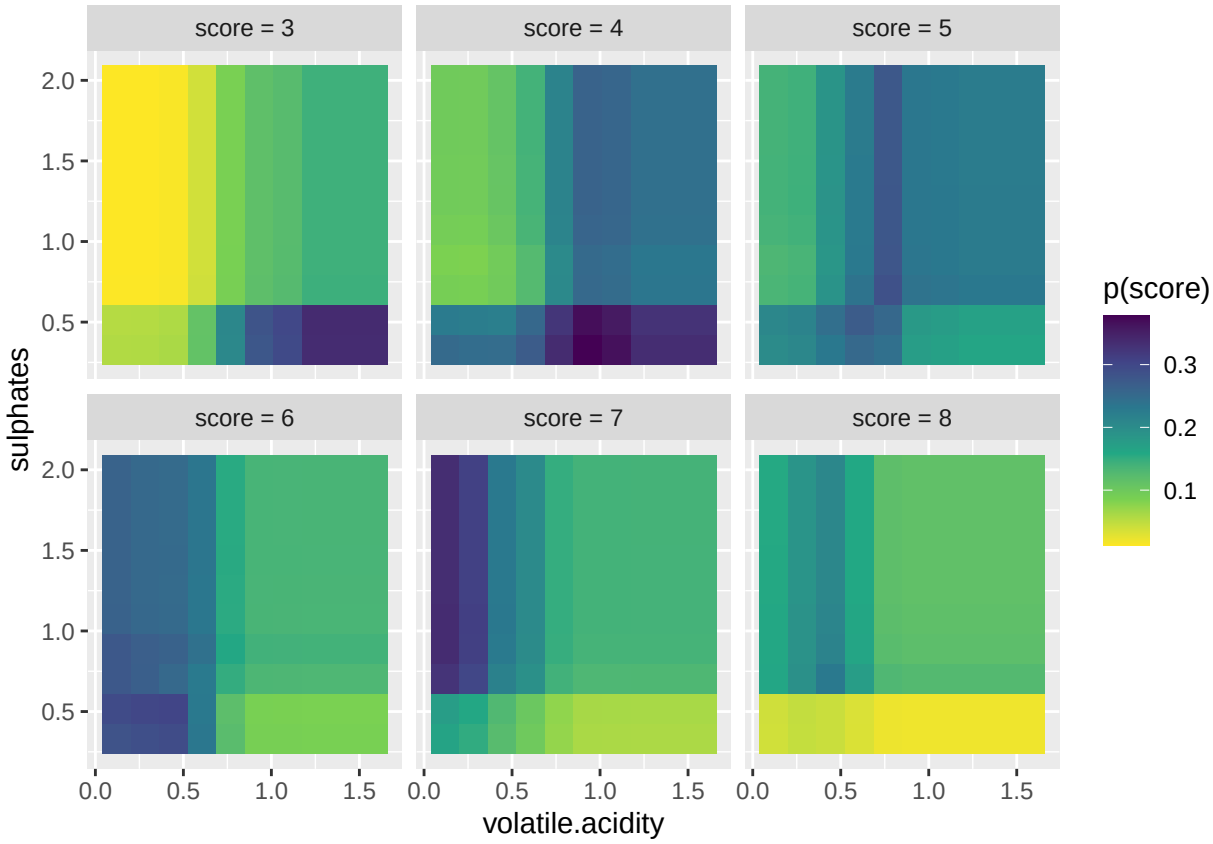
We can conclude that increasing *volatile.acidity* appears to be associated with a decline in sensory quality. However, other variables seem to play an important role in differentiating quality classes at either end of the spectrum. It should also be noted that partial dependence plots can be misleading in the presence of interactions.

As a last step, let's take a look at the joint effects of *volatile.acidity* and *sulphates*, the second most important variable. We can use the same prediction function *pred.funall* and pass a vector of predictors to

partial(). I then use *ggplot2* to create a faceted heatmap.

```
pdps_i <- pdp::partial(forest1, type="classification",
                       pred.var=c("volatile.acidity", "sulphates"),
                       pred.fun=pred.funall, train=traindata,
                       grid.resolution=10)

pdp_i <- ggplot(pdps_i, aes(volatile.acidity, sulphates, fill=yhat)) + geom_tile() +
         #scale_fill_gradient(high = "red", low = "yellow")
         scale_fill_viridis_c(name="p(score)", direction=-1)
pdp_i + facet_wrap(~yhat.id)
```



We can now see what combinations of *volatile.acidity* and *sulphates* each class is most strongly associated with. The worst observed class, for example, sees the highest predicted probability for high *volatile.acidity* and low *sulphates*. If *volatile.acidity* decreases while *sulphates* stays low, the model is more likely to predict the second-worst class.

Another thing we can tell from this plot is that there is very little change in predicted probability of any class in the 0.6 - 2.0 range of *sulphates* unless *volatile.acidity* also changes. In other words, if *sulphates* > 0.6, changes in sensory quality can mostly be attributed to *volatile.acidity*.