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. For the steps taken during preprocessing see *models/preprocess_train.R*

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

```
## 'data.frame':    335 obs. of  12 variables:
##  $ fixed.acidity       : num  7.2 8.8 7.2 7.4 9.7 ...
##  $ volatile.acidity    : num  0.63 0.61 0.62 0.55 0.53 ...
##  $ citric.acid         : num  0 0.3 0.06 0.19 0.6 ...
##  $ residual.sugar      : num  1.9 2.8 2.5 1.8 2 ...
##  $ chlorides           : num  0.097 0.088 0.078 0.082 0.039 ...
##  $ free.sulfur.dioxide : num  14 17 17 15 5 ...
##  $ total.sulfur.dioxide: num  38 46 84 34 19 ...
##  $ density             : num  0.997 0.998 0.997 0.997 0.996 ...
##  $ pH                  : num  3.37 3.26 3.51 3.49 3.3 ...
##  $ sulphates           : num  0.58 0.51 0.53 0.68 0.86 ...
##  $ alcohol             : num  9 9.3 9.7 10.5 12.4 ...
##  $ quality             : Factor w/ 6 levels "1","2","3","4",..: 4 2 3 3 4 2 2 2 6 5 ...
```

Models

Using the dataset described above, I trained two simple ordinal forest models which predict sensory quality of wine samples from their physicochemical parameters. For the first model, *forest1*, I used the default settings for class probability prediction. For the second model, I forced the algorithm to consider the variable *residual.sugar* at every split in additon to the *mtry* candidate parameters for splitting (for code see *models/preprocess_train.R*).

```
load(here("models/forest1.rda"))
load(here("models/forest2.rda"))
```

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] 0.07461709
```

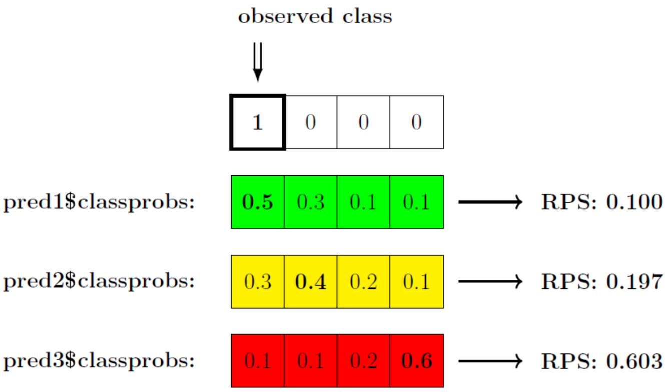
Now, what does that mean? Is that a good score? What even is the RPS?

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 (0=no, 1=yes) for the observation in class k.

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.



observed class

1	0	0	0

pred1$classprobs: 0.5 0.3 0.1 0.1 → RPS: 0.100

pred2$classprobs: 0.3 0.4 0.2 0.1 → RPS: 0.197

pred3$classprobs: 0.1 0.1 0.2 0.6 → RPS: 0.603

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 dataset, 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*.

```r
performance1$rpss
```

```
## [1] 0.616255
```

Confirm that we get the same result when explicitly passing probabilities derived from the class distribution as reference/baseline:

```r
prop <- table(testdata$quality) %>% prop.table()
bl <- as.vector(prop)
performance1_bl <- rps(obs=observed_quality, pred=predicted_quality, baseline=bl)
performance1_bl$rpss
```

```
## [1] 0.616255
```

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

```
performance1$rps.clim
```

```
## [1] 0.1944444
```

From the positive RPSS and the fact that $performance1\$rps < performance1\$rps.clim$, we can infer that the OF performs better than a "dummy model" which predicts sensory quality based solely on quality score distribution. We could use the baseline argument to compare our OF against other dummy models, e.g. one that always predicts class 1.

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

```
predicted_quality2 <- predict(forest2, newdata=testdata)$classprobs
performance2 <- rps(obs=observed_quality, pred=predicted_quality2, baseline=bl)
the_rps <- c(performance1$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)
```

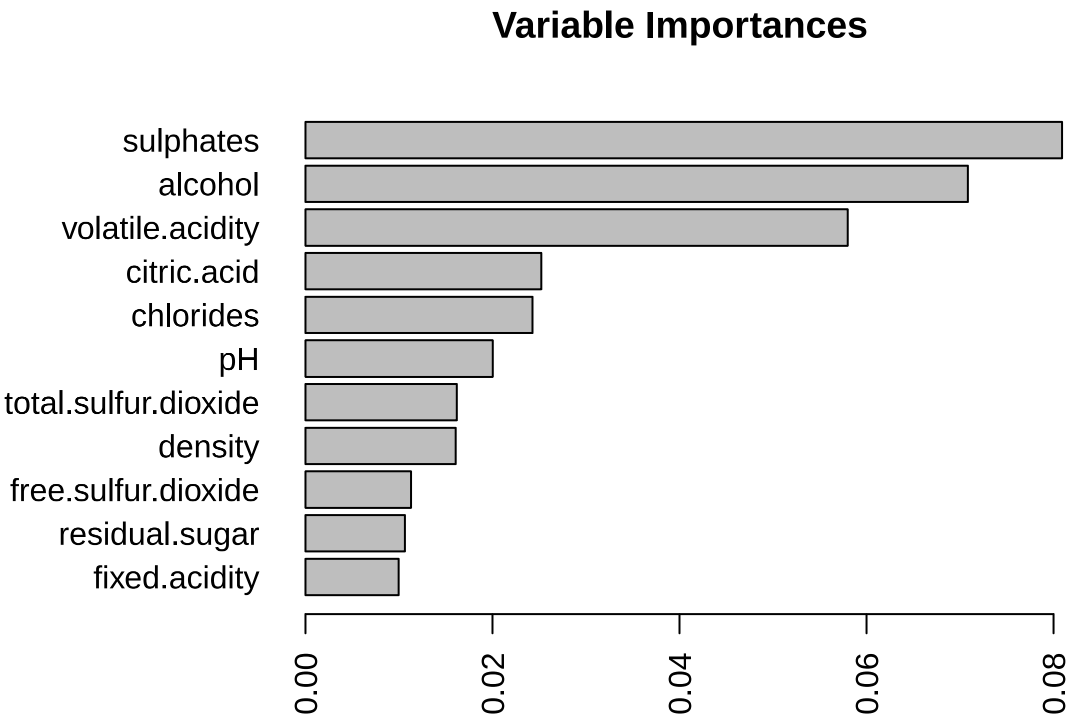
```
##            the_rps  the_rpss
## forest1 0.07461709 0.6162550
## forest2 0.07750780 0.6013884
```

forest1 has a slightly smaller RPS and a slightly larger RPSS than *forest2*, i.e., a larger relative improvement over the baseline when compared to *forest2*. We can infer that the model using only *mtry* randomly selected candidates at a given split performs slightly better.

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 *sulphates* to predict sensory quality, closely followed by *alcohol* and *volatile.acidity*.

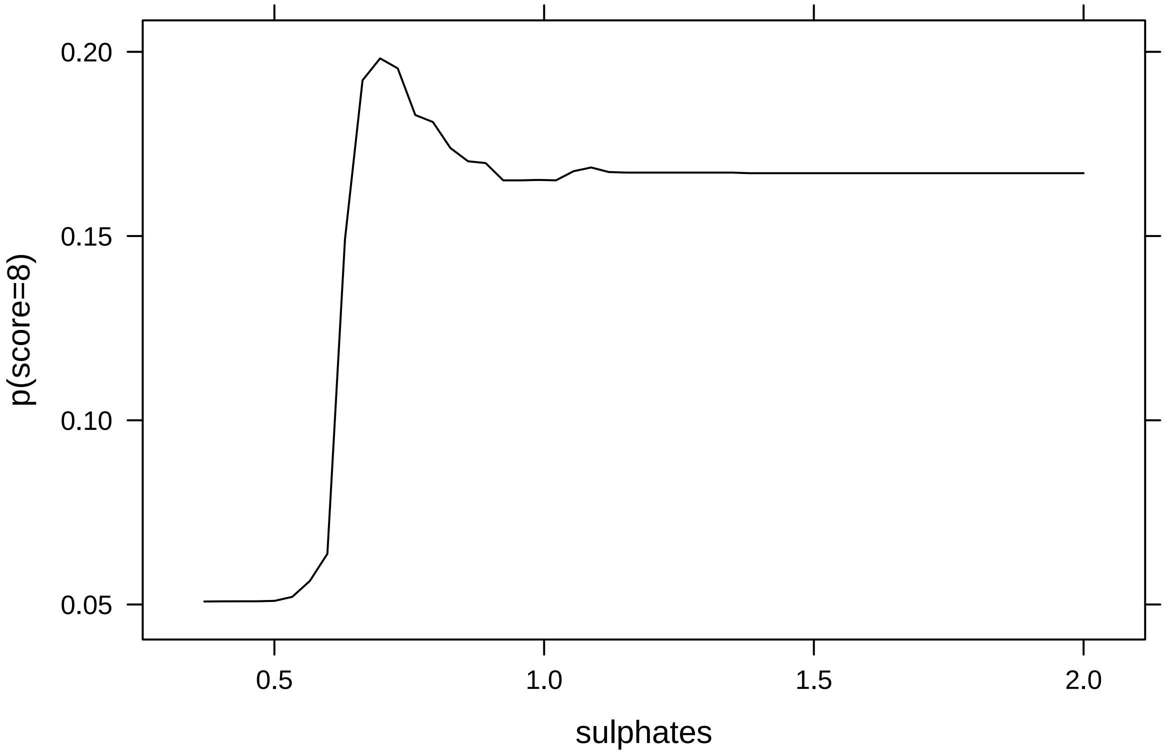
Partial Dependence Plots for Feature Effects

We don't just want to know that our model most strongly relies on *sulphates* to predict sensory quality, but *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 *sulphates* (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.

```r
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 = "sulphates",
                   pred.fun = pred.best, which.class = 8,
                   train = traindata) %>% plotPartial(ylab = "p(score=8)")
```



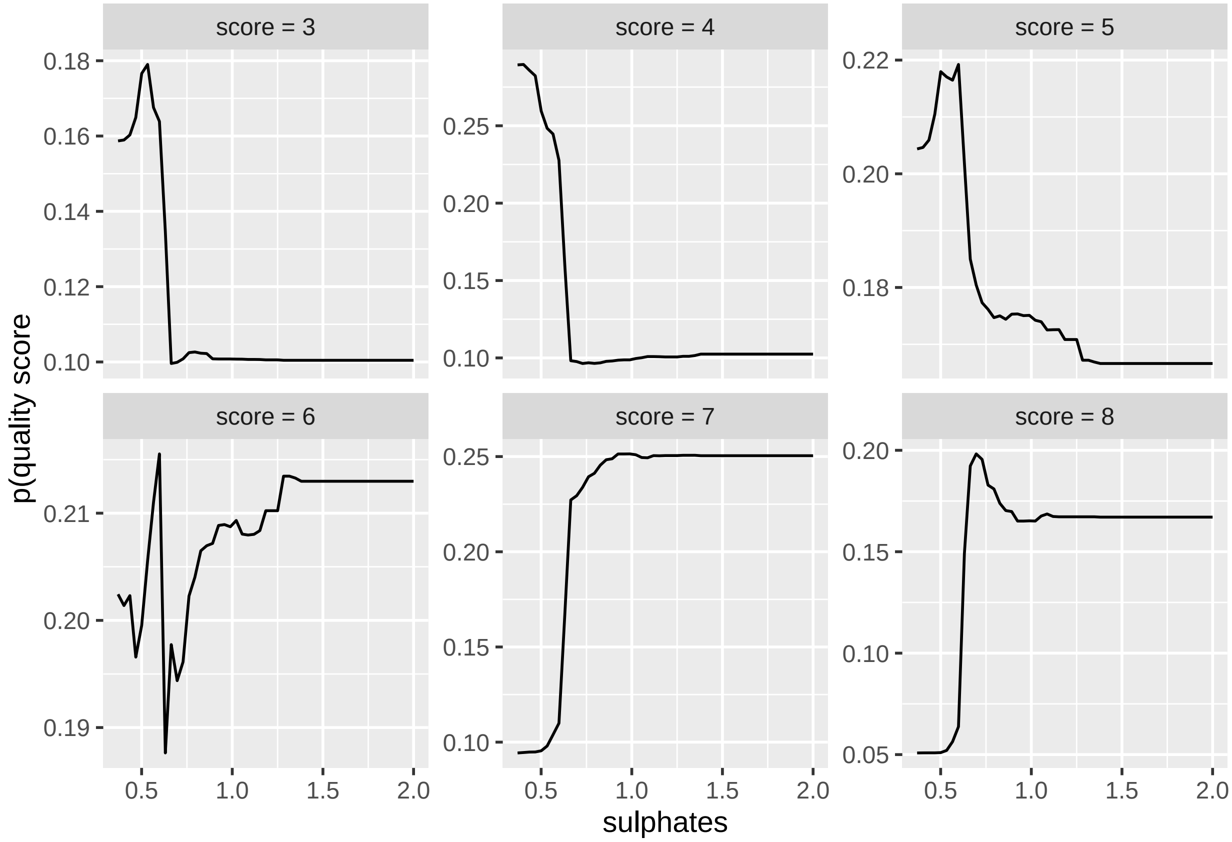
We can see that there is a sharp increase in predicted probability of the best observed class (8) between *sulphates* values of around 0.5 and .75. Between 0.75 and 1.0, predicted probability slightly drops. Above 1.0, predicted probability stays pretty much constant.

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="sulphates",
             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 *sulphates*. For a nice and neat display, I take advantage of *ggplot2* facets with a free y scale:

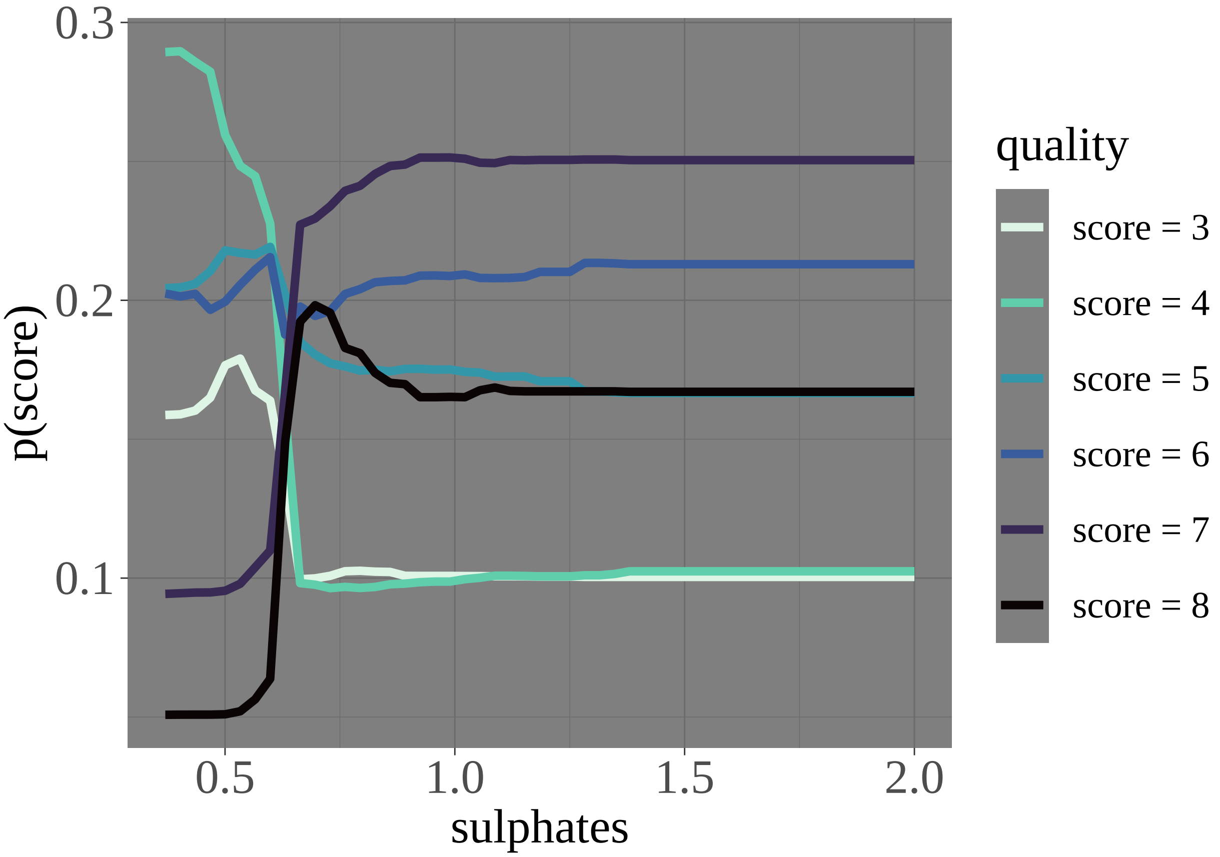
```
pdp_curves <- ggplot(pdps, aes(sulphates, yhat)) +
             geom_line()
pdp_curves + facet_wrap(~yhat.id, scales="free_y") + labs(y="p(quality score")
```



Obviously, these plots are not as intuitively interpretable as a single curve. However, we can see that for the lower classes, increasing *sulphates* lead to a drop in predicted probability, whereas higher classes see an increase in predicted probability with rising *sulphates* values. Predicted probability of the highest observed score (8), however, declines again as *sulphates* exceeds 0.75. When comparing the y-scales of the sublots, we can see that the class corresponding to a quality score of 4 seems to have the strongest dependence on *sulphates*. Predicted probability of this class varies between ∼ 0.1 and ∼ 0.3. Predicted probability of score 6, on the other hand, only varies in an extremely narrow range between ∼ 0.1875 and ∼ 0.215. In other words, a maximum difference in predicted probability of only ∼ 2.75% can be attributed to *sulphates*. For score 5, the maximum difference is slightly larger, but still very small at ∼ 5.5%. This suggests that the perception of "medium 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(sulphates, 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="sulphates", y="p(score)")
```



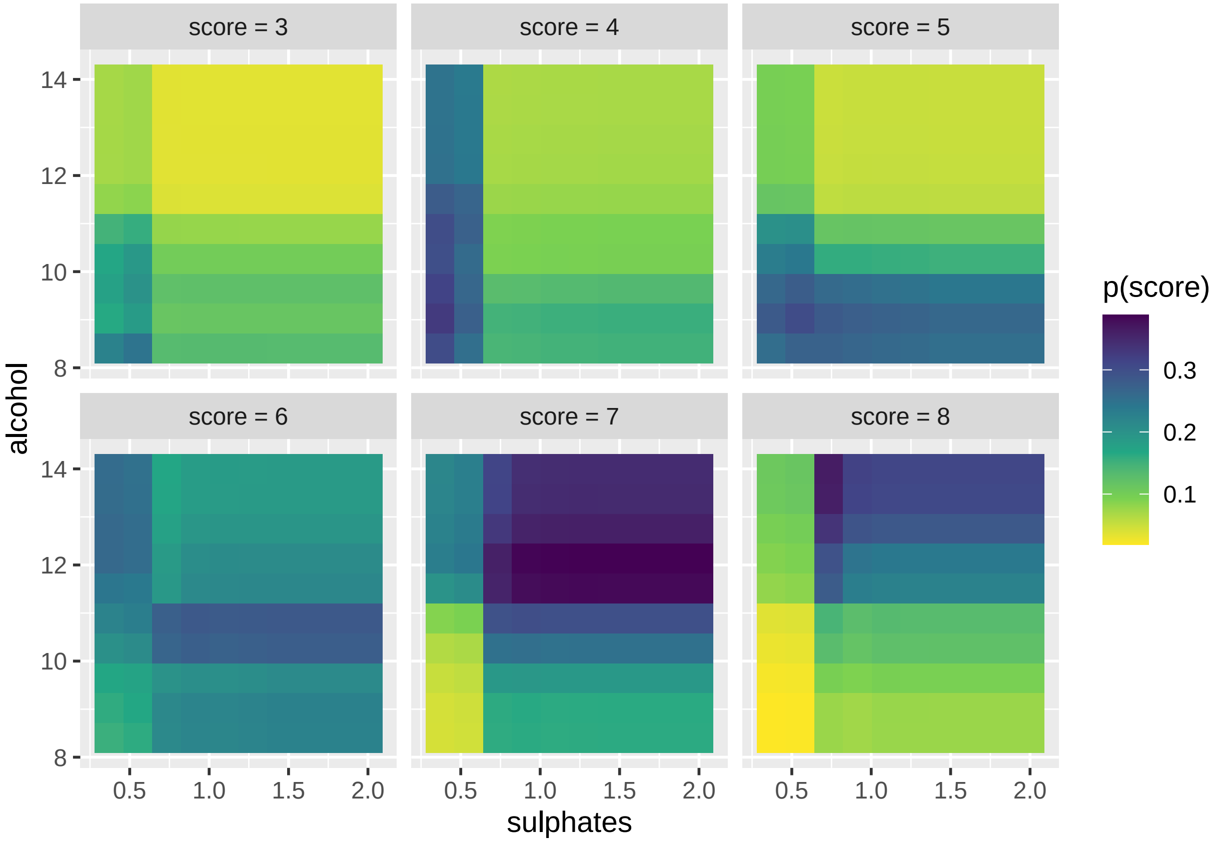
Looking at this plot we see that samples with *sulphates* levels below ∼ 0.75 are most strongly associated with a score of 4 - the second lowest score observed in the data - while *sulphates* levels above that value are most strongly associated with a score of 7, which is the second highest observed. We can also see that while there is a slight increase in predicted probability of score 7 between *sulphates* levels of 0.75 and 1.0, predicted probability of score 8 declines. For *sulphates* values above 1.0, there is no significant change in predicted probability of any class.

Overall, the partial dependecens suggest that the 0.3 - 1.0 range of *sulphates* is crucial for sensory quality of *Vinho Verde*, and that samples with high values of this parameter are predicted to achieve higher-than-average but not excellent scores. It should be noted, however, 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 *sulphates* and *alcohol*, 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("sulphates", "alcohol"),
                       pred.fun=pred.funall, train=traindata,
                       grid.resolution=10)

pdp_i <- ggplot(pdps_i, aes(sulphates, alcohol, 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 see that the worst observed score (3) is most strongly associated with both low alcohol and sulfates. The best observed score of 8 has the highest predicted probability for a *sulphates* value of ~ 0.75 and an alcohol level > 13 %. The plot also suggests that predicted probability of score 4 more strongly depends on *sulphates* than *alcohol*, and that the opposite is true for class 5. Both variables only marginally affect predicted probability of score 6. For the class corresponding to a score of 7, predicted probability is highest for *sulphates* ≥ 0.75 and *alcohol* values between ~ 11 and 12.5 %.

We can also see that there is some interaction between the two variables, especially in the classes corresponding to scores 6 and 7.