Classification: Moving Beyond Majority Vote

36-290 – Statistical Research Methodology

Week 8 Tuesday – Fall 2021

Data

Below we load in some variable star data. Much of the detail is unimportant, and so is consigned to an "unechoed" code chunk. (You can see the contents of this chunk if you view the contents of the `R Markdown` file.)

The data represent two classes: contact binary stars (or `CB`s) and non-contact binary stars (or `NON-CB`s). In pre-processing, we retain 5000 examples of each. The research question is: can we learn a model that discriminates between the two classes?

```
response = factor(response.new,levels=c("NON-CB","CB"))
set.seed(101)
s = sample(length(response),round(0.7*length(response)))
pred.train = predictors[s,]
pred.test  = predictors[-s,]
resp.train = response[s]
resp.test  = response[-s]
```

Note the first line above. When one defines a factor variable, by default the levels are defined via alphabetical order. But here, we envision that `NON-CB` is a "negative" response (and thus naturally to be associated with value 0) and `CB` is a "positive" response, so we assign the levels to be in this order. (You don't have to do this, so long as you are careful when interpreting your confusion matrix.)

Random Forest Analysis

```
suppressMessages(library(randomForest))
rf.out = randomForest(resp.train~.,data=pred.train)
resp.pred = predict(rf.out,newdata=pred.test,type="response") # response == majority vote
(rf.mcr = mean(resp.pred!=resp.test))
```

```
## [1] 0.2003333
```

```
table(resp.pred,resp.test)
```

```
##          resp.test
## resp.pred NON-CB   CB
##    NON-CB   1112  253
##    CB        348 1287
```

Let's look at the estimated probabilities for each class, by redoing the prediction above with argument `prob` instead of `response`:

```
predict(rf.out,newdata=pred.test,type="prob")[1,]
```

```
## NON-CB     CB
##  0.012  0.988
```

Unlike the case for `predict.glm()`, `predict.randomForest()` outputs a matrix of class probabilities. Here, the first row represents test object 1: random forest estimates that there is a 98.8% chance that it is a contact binary, and a 1.2% chance that it is not. Because the default threshold proportion is 50%, the object will be predicted to be a contact binary.

Altering the Classification Threshold

We can change the threshold by utilizing the `cutoff` argument. `cutoff` can be a bit tricky to work with: if you have K classes, you have to specify K cutoff values that sum to 1, but really only the first $K-1$ matter. So here, if we want the threshold for contact binary identification to be 0.8, we'd add the argument `cutoff=c(0.2,0.8)`:

```
resp.pred = predict(rf.out,newdata=pred.test,type="response",cutoff=c(0.2,0.8))
(rf.mcr = mean(resp.pred!=resp.test))
```

```
## [1] 0.315
```

```
table(resp.pred,resp.test)
```

```
##          resp.test
## resp.pred NON-CB   CB
##    NON-CB   1395  880
##    CB         65  660
```

We find that changing the threshold to 0.8 may not actually be a good thing to do: the MCR went up. Besides that, we see that our ability to identify contact binaries is diminished.

However: our ability to identify `NON-CB`s is *greatly* enhanced. If your goal is to produce a "pure" catalog of variables that are not contact binaries, then increasing the threshold from, e.g., 0.5 to 0.8 is actually a good thing to do. (In fact, maybe you want to set the threshold even higher!) (Another way of stating this: if constructing a pure catalog representative of one of the classes is your goal, then the MCR is not going to be the correct loss function to use.)

Receiver Operating Characteristics (ROC) Curve

One can examine how the threshold matters by repeatedly calling `predict` with different cutoff arguments. But a more efficient approach is to generate a so-called ROC curve. Let's simply run one and look at the output:

```
suppressMessages(library(pROC))
resp.pred = predict(rf.out,newdata=pred.test,type="prob")[,2]
roc.rf = suppressMessages(roc(resp.test,resp.pred))
names(roc.rf)
```

```
##  [1] "percent"            "sensitivities"      "specificities"      "thresholds"         "direction"
##  [6] "cases"              "controls"           "fun.sesp"           "auc"                "call"
## [11] "original.predictor" "original.response"  "predictor"          "response"           "levels"
```

The first part of the output to highlight is "thresholds": this is a vector of different cutoff values...for our example, 498 of them!

```
roc.rf$thresholds[1:10]
```

```
##  [1]  -Inf 0.001 0.003 0.005 0.007 0.009 0.011 0.013 0.015 0.017
```

ROC Curve

We also notice "sensitivities" and "specificities." These make most sense if I lay out the following matrix first:

True Negative (TN)	False Negative (FN)
False Positive (FP)	True Positive (TP)

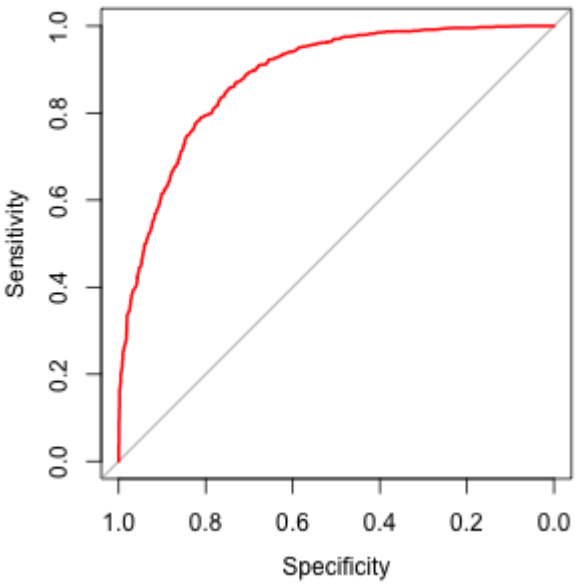
(Note: some sources, like wikipedia, switch the rows and columns; as long as you know what is where, it ultimately doesn't make a difference which convention you choose.) A "true positive," or TP, is classifying a `CB` as a `CB`, while a "false positive," or FP, is classifying a `NON-CB` as a `CB`, etc.

- sensitivity or recall or true positive rate or completeness: TP/(TP+FN), evaluated on right column
- specificity or true negative rate: TN/(TN+FP), evaluated on left column
- purity or positive predictive value: TP/(TP+FP), evaluated on bottom row

ROC Curve

A ROC curve is a curve that shows sensitivity (how well we identify the positive class) versus specificity (how well we identify the negative class); the curve is constructed by taking specificity-sensitivity pairs for each threshold value and "connecting the dots":

```
plot(roc.rf,col="red",xlim=c(1,0),ylim=c(0,1))          # Another non-ggplot production
```



The diagonal line represents the baseline of "random guessing." The better the overall performance of a classifier, the more the curve moves towards the upper left.

Area Under Curve (AUC)

One means to compare classifiers that goes beyond misclassification rates based on majority vote is to compute the area under the ROC curve. For random guessing, the area is 0.5. For perfect classification at all thresholds, the AUC is 1.

```
cat("AUC for random forest: ",roc.rf$auc,"\n")
```

```
## AUC for random forest:  0.8829419
```

The advantage of AUC over simple MCR via majority vote is that it attempts to take into account all possible threshold values at once, i.e., it allows us to identify which classifier performs the best over a range of conditions.

Determining an Optimal Threshold Value

By introducing ROC curves and the concept of AUC, we now have a new metric for model selection. But ultimately, we do need to choose *one* threshold value so as to generate predictions. We can use the ROC curve to do this, while acknowledging that there is no unique means to choose that one value.

An often-used choice is *Youden's J* statistic:

$$\mathrm{J} \;=\; \text{sensitivity} \;+\; \text{specificity} \;-\; 1\,.$$

Basically, we want to determine the threshold value that allows us to best identify `NON-CB`s and `CB`s at the same time. For our example:

```
J = roc.rf$sensitivities + roc.rf$specificities - 1
w = which.max(J)
cat("Optimum threshold for random forest: ",roc.rf$thresholds[w],"\n")
```

```
## Optimum threshold for random forest:  0.473
```

The value is close to 0.5, which is not surprising because our data had balanced classes. When the data feature unbalanced classes, optimal thresholds may differ greatly from 0.5.

A Classification Analysis Workflow

We will wrap this up by stating what you might do when analyzing binary categorical response data:

1. Learn a model as before and generate response class probabilities for each test-set datum using `predict()`.
2. Using these probabilities as input, generate a ROC curve using the `roc()` function.
3. Use the output from `roc()` to determine the optimal threshold for your chosen metric (e.g., Youden's J statistic). (Report the metric and the threshold.)
4. Rerun `predict()` with the optimal threshold encoded in the `cutoff` parameter and generate *class predictions* for each test-set datum.
5. Use the output to create a confusion matrix (for visualization) and, e.g., a final misclassification rate.
6. Ultimately pick the model with, e.g., the highest AUC value, or the lowest MCR value, or...

Done.