Classification Model Metrics

Ben Arancibia

May 2, 2015

Introduction

I begin with the lab portion and completed the lab in a separate file. I kept the important basic parts of the lab in the assignment and do the assignment exercises below under the Assignment heading.

1) Read in the csv

```
data <- read.csv("/users/bcarancibia/CUNY_IS_621/classificationmethods/classificationoutput.csv")
```

There are three key columns: * class: the actual class of the observation * Scored.Class: the predicted class of the observation * Scored.Probability: the predicted probability of success for the observation

2) Use the table() function to get the raw confusion matrix.

```
confusion.matrix <- table(data$class, data$Scored.Labels)

(119+27) / 181
```

```
## [1] 0.8066298
```

The columns represent the actual class and the rows represent the predicted class.

What the confusion matrix is, is an easy to quickly show two types of errors. The two types of errors are those errors which are false negatives or false positives. The overall error percentage is quite low, around 80% was scored correctly, but the errors for each predicted class are different. For example, the number of actual scored 0s is 149, but the predicted 0s is only 119, error rate of 20%. For 1s, it is an error rate of 16.6% (5/30).

3) Write a function that takes the dataset as a dataframe with actual and predicted classifications identified. Return the accuracy of the predictions.

$Accuracy = (TP + TN)/(TP + FP + TN + FN)$

```
#tp = true positive 0
#tn = true negative 1
#fp = false positve
#fn = false negative

TP <- 119
TN <- 27
FP <- 5
FN <- 30

accuracy <- (TP+TN)/(TP+FP+TN+FN)
accuracy
```

```
## [1] 0.8066298
```

4) Write a function that takes the dataset as a dataframe to get the error rate

```
error <- (FP+FN)/(TP+FP+TN+FN)
error
```

```
## [1] 0.1933702
```

```
accuracy + error
```

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

5) Precision function

```
precision <- TP/(TP+FP)
precision
```

```
## [1] 0.9596774
```

6) Recall of the predictions also know as sensitivity

```
recall <- TP/(TP+FN)
recall
```

```
## [1] 0.7986577
```

7) Specificity

```
specificity <- TN/(TN+FP)
specificity
```

```
## [1] 0.84375
```

8) F1 score of the predictions

```
F1 <- 2*((precision*recall)/(precision+recall))
F1
```

```
## [1] 0.8717949
```

Assignment

1. Write a function that generates a lift chart using a dataframe with actual clssification and scored probability.

Lift = expected response in a specific lot / expected response in a random lot

- Predict a set of samples that were not used in the model building process but have known outcomes. (Scored.Labels)
- Determine the baseline event rate, i.e., the percent of true events in the entire data set. (accuracy)
- Order the data by the classification probability of the event of interest.
- For each unique class probability value, calculate the percent of true events in all samples below the probability value.
- Divide the percent of true events for each probability threshold by the baseline event rate.

```
library(ROCR)
```

```
## Warning: package 'ROCR' was built under R version 3.1.3
```

```
## Loading required package: gplots
```

```
## Warning: package 'gplots' was built under R version 3.1.3
```

```
##
## Attaching package: 'gplots'
##
## The following object is masked from 'package:stats':
##
##     lowess
```

```
library(plyr)
data$classfactor <- ifelse(data$class=="0","Yes","No")
data$classfactorscored <- ifelse(data$Scored.Labels=="0","Yes","No")

pred.rocr <- prediction(data$Scored.Probabilities, data$classfactor)
performance(pred.rocr, 'auc')@y.values[[1]]
```

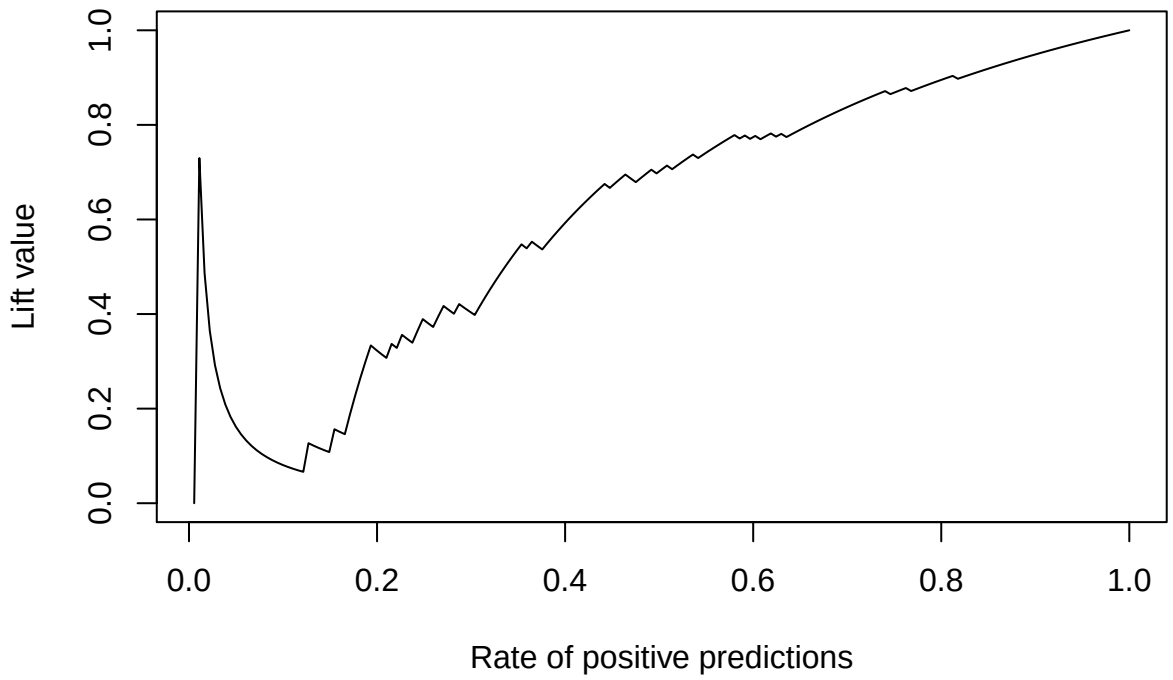
```
## [1] 0.1496887
```

```
dec.table <- ldply((1:10)/10, function(x) data.frame(
    decile=x,
    prop.yes=sum(data$class[1:ceiling(nrow(data)*x)])/sum(data$class),
    lift=mean(data$class[1:ceiling(nrow(data)*x)])/mean(data$class)))
print(dec.table, digits=2)
```

```
##    decile prop.yes lift
## 1     0.1    0.088 0.84
## 2     0.2    0.211 1.03
## 3     0.3    0.281 0.92
## 4     0.4    0.421 1.04
```

```
## 5     0.5    0.491 0.98
## 6     0.6    0.632 1.05
## 7     0.7    0.754 1.08
## 8     0.8    0.807 1.01
## 9     0.9    0.877 0.97
## 10    1.0    1.000 1.00
```

```
plot(performance(pred.rocr, 'lift', 'rpp'))
```



2.Investigate the lift function in the caret package. Use it to create a lift chart on the test data.

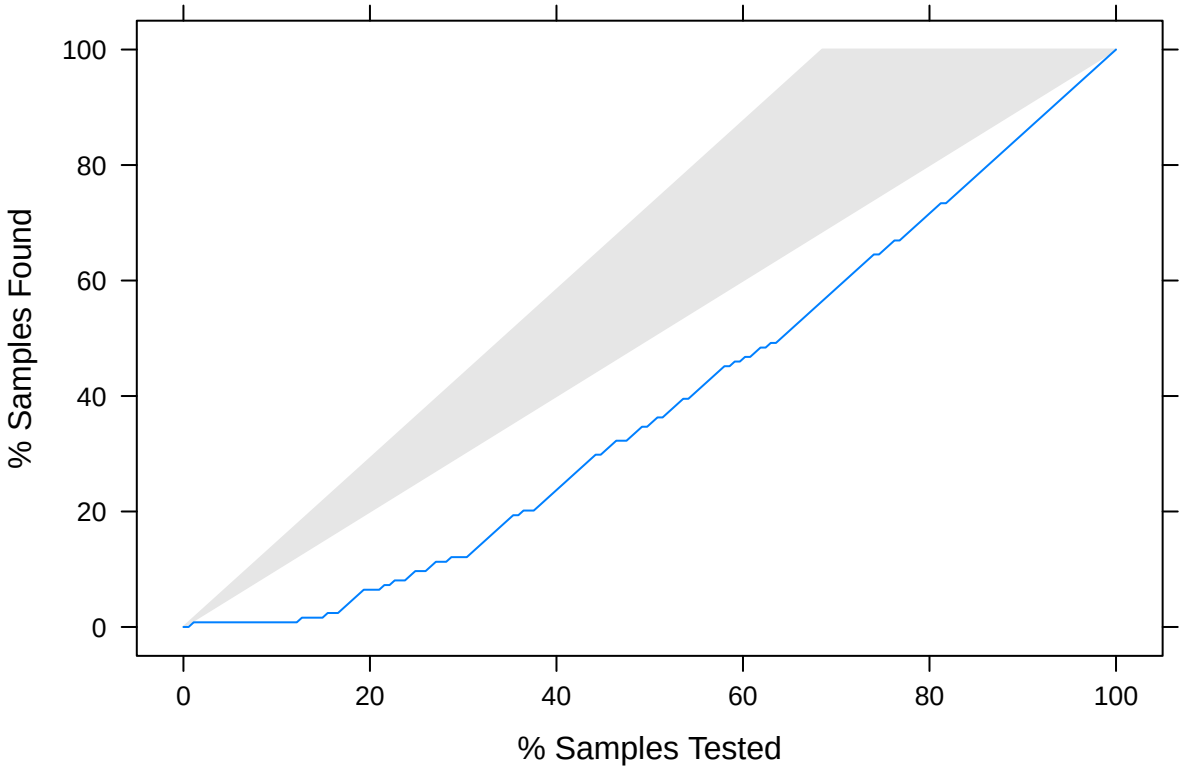
```
library(caret)
```

```
## Loading required package: lattice
## Loading required package: ggplot2
```

```
data$classfactor <- ifelse(data$class=="0","a","b")

liftcurve <- lift(classfactor ~ Scored.Probabilities, data = data)

xyplot(liftcurve, auto.key = list(columns =2, lines=TRUE, points=FALSE))
```



The lift function using the caret function is a lot more smooth and does not seem to show the same data...