Evaluation Metrics for Classification Models

Brooke Anderson

January 26, 2016

```
knitr::opts_knit$set(fig.path = '../figures/EvalMetrics-',
                     root.dir = '..')
```

Note: I was tired of my Rmd documents, which I put in the sub-directory "Rscripts", running from a different working directory than my R session when I'm working on this project (which was running from "Titanic", the parent directory of that "Rscripts directory"). This was making me have to use one relative pathname when I knitted and a different when I tested code chunks interactively. Therefore, I used the `root.dir` option in `opts_knit` to change the working directory that will be used when I knit to the parent directory of where the Rmd file is saved.

Load required libraries.

```
library(dplyr)  ## Data wrangling
library(klaR) ## Includes `NaiveBayes` function
```

Bring in the data:

```
train <- read.csv("data/train.csv") %>%
  mutate(Survived = factor(Survived),
         Pclass = factor(Pclass),
         Name = as.character(Name),
         Sex = factor(Sex),
         child = as.factor(as.character(Age < 6)))
test <- read.csv("data/test.csv") %>%
  mutate(Pclass = factor(Pclass),
         Name = as.character(Name),
         Sex = factor(Sex),
         child = as.factor(as.character(Age < 6)))
```

For this, I'll do different evaluation metrics for a Naive Bayes with only sex as a predictor. I think that the first of these might be the benchmark models for Kaggle, with the "predict survival if woman, death otherwise" model that some folks in class put in.

This time, I'll try using the `NaiveBayes` function from the `klaR` package. Based on Kuhn and Johnson, in comparing this with the `e1071` package:

> "Both offer Laplace corrections, but the version in the `klaR` package has the option of using conditional density estimates that are more flexible."

```
nb_sex <- NaiveBayes(Survived ~ Sex, data = train)
```

This function's output provides some of the same values as the other Naive Bayes function I tried, including $P(Y)$ and $P(X|Y)$:

```
nb_sex$apriori
```

```
## grouping
##         0         1
## 0.6161616 0.3838384
```

```
nb_sex$tables
```

```
## $Sex
##         var
## grouping    female      male
##        0 0.1475410 0.8524590
##        1 0.6812865 0.3187135
```

Evidently, you can use `predict` with this function, including with an optional argument specifying `newdata`. To predict with the training data:

```
pred_train_sex <- predict(nb_sex, newdata = train)
```

This prediction includes two elements: the class predictions (0 = died; 1 = survived) and the posterior class probabilities:

```
names(pred_train_sex)
```

```
## [1] "class"     "posterior"
```

```
head(pred_train_sex$class)
```

```
## [1] 0 1 1 1 0 0
## Levels: 0 1
```

```
head(pred_train_sex$posterior)
```

```
##              0         1
## [1,] 0.8110919 0.1889081
## [2,] 0.2579618 0.7420382
## [3,] 0.2579618 0.7420382
## [4,] 0.2579618 0.7420382
## [5,] 0.8110919 0.1889081
## [6,] 0.8110919 0.1889081
```

I checked to see if this always predicts that women survive and men die:

```
table(train$Sex, pred_train_sex$class)
```

```
##
##            0   1
##   female   0 314
##   male   577   0
```

Yep. So, this should give the results from the benchmark "Sex" model in the Kaggle competition.

To create predictions to submit to Kaggle:

```
pred_test_sex <- predict(nb_sex, newdata = test)
```

And write them out to a comma-separated file. Note that the `pred_test$class` is saved as a factor variable, so if you don't convert to a character and then a number using the `as.*` functions, you might get outputs of 1s and 2s instead of 0s and 1s, which would cause problems when you submit to Kaggle.

```
out <- cbind(test$PassengerId,
             as.numeric(as.character(pred_test_sex$class)))
colnames(out) <- c("PassengerId","Survived")
write.csv(out, file = "predictions/nb_sex.csv", row.names = FALSE)
```