Sensitivity and Specificity

Renaldo Williams

October 5, 2015

Readings

APM

- ***Chapter 5 Measuring Performance in Regression Models*** (esp. ***5.2 The Variance Bias Trade-Off***) (5 pages)
- ***Chapter 11 Measuring Performance in Classification Models*** (~20 pages)
- ***Chapter 7.4 K-Nearest Neighbors (regression)*** (2 pages)
- ***Chapter 13.5 K-Nearest Neighbors (classification)*** (3 pages)

```
##
## Attaching package: 'dplyr'

## The following objects are masked from 'package:stats':
##
##     filter, lag

## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union

## Loading required package: lattice
```

EXERCISE 1: Resampling

`x` is a random variable. We want to not only know what the `mean(x)` is but want to calculate the uncertainty of `mean(x)`. Measuring the uncertainty requires repeated measurements of `mean(x)`.

1. Calculate the mean of `x`.
2. Calculte the `sd( mean(x) )` using the **using 10-fold cross-validation**. Create your own folds, show your work. (An example is for the Bootstrap is given as a hint.)

```r
#set.seed(1)
#x <- runif(20,1,20) #take random number from uniform distribution take 20 numbers from 1 - 20
#x_mean = mean(x)

k=10

# CROSS-VALIDATION
# Take
x <- runif(20,1,20) #take random number from uniform distribution take 20 numbers from 1 - 20
x_mean <- numeric()
for(k in 1:10) {
  x_mean <- append(x_mean, sample(x, replace=TRUE) %>% mean)
}
sd_cv <- x_mean %>% sd # This is the standard deviation of the mean of k folds

# BOOTSTRAP (EXAMPLE)
# shorthand function
# (RW) This takes 20 samples from vector x and finds the mean. It does it 10 times since k = 10. It the
sd_boot <- sapply(1:k, function(i) sample(x,replace=TRUE) %>% mean ) %>% sd #take my vector, x, and tak

# sort(sample(x,replace=TRUE))
```

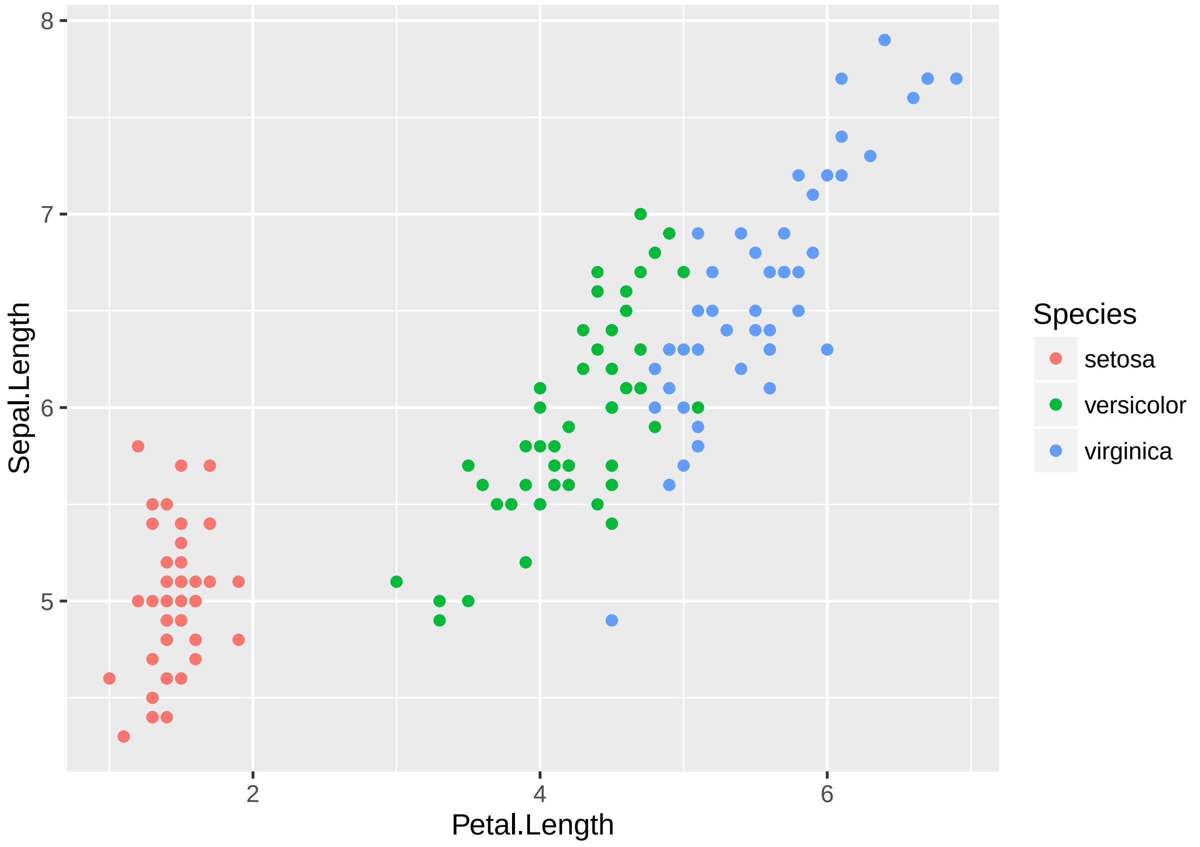
- sd_cv is: 0.9688132
- sd_boot is: 1.0179103

Exercise 2: Binomial Metrics

Here's a really simple Model of Versicolor iris based on the **iris** data :

```
data(iris)

qplot( data=iris, x=Petal.Length, y=Sepal.Length, color=Species )
```



```
# Create Dependent Variable
iris$Versicolor <- ifelse( iris$Species == 'versicolor', "versicolor", "other" ) %>% as.factor
```

```r
iris$Species = NULL
nrow(iris)
```

```
## [1] 150
```

```r
#RW take sample 75 samples from 1 - 150, save row id's
wh <- sample.int( nrow(iris), size=nrow(iris)/2 ) # this samples rows from 1 to n=150. Only store half

train <- iris[ wh,] # 75 items for training
test <- iris[ -wh, ] # 75 items for testing


fit.glm <- glm( Versicolor ~ . - Sepal.Length, data=train, family=binomial )
fit.pred <- predict(fit.glm, test)

summary(fit.glm)
```

```
##
## Call:
## glm(formula = Versicolor ~ . - Sepal.Length, family = binomial,
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -1.8722  -0.7660  -0.3626   0.8040   2.2275
##
## Coefficients:
##              Estimate Std. Error z value Pr(>|z|)
## (Intercept)    8.6188     3.7943   2.272  0.02312 *
## Sepal.Width   -3.4061     1.1187  -3.045  0.00233 **
## Petal.Length   1.1106     0.6945   1.599  0.10979
## Petal.Width   -2.7682     1.6172  -1.712  0.08696 .
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 94.030  on 74  degrees of freedom
## Residual deviance: 72.026  on 71  degrees of freedom
## AIC: 80.026
##
## Number of Fisher Scoring iterations: 5
```

```r
summary(fit.pred)
```

```
##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
## -5.8090 -2.0390 -0.6506 -0.8219  0.4425  2.9250
```

```r
table(test$Versicolor)
```

```
##
##      other versicolor
##         49         26
```

```r
cor(iris[c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")])
```

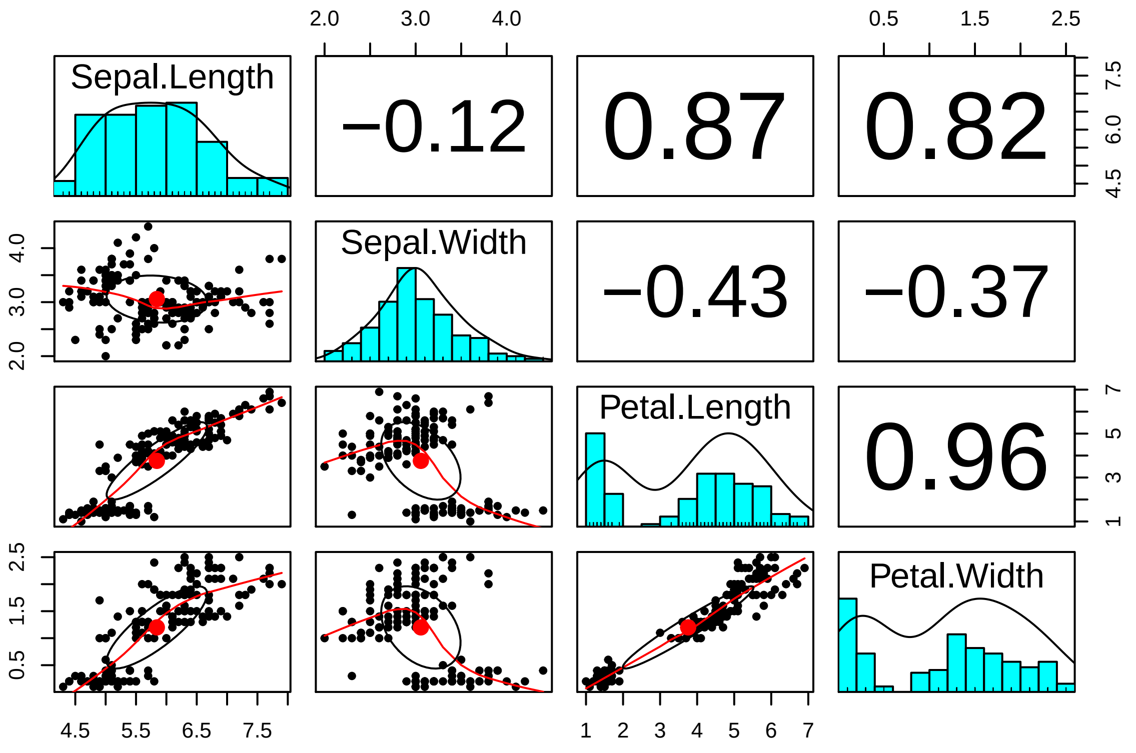
```
##              Sepal.Length Sepal.Width Petal.Length Petal.Width
## Sepal.Length    1.0000000  -0.1175698    0.8717538   0.8179411
## Sepal.Width    -0.1175698   1.0000000   -0.4284401  -0.3661259
## Petal.Length    0.8717538  -0.4284401    1.0000000   0.9628654
## Petal.Width     0.8179411  -0.3661259    0.9628654   1.0000000
```

```r
#install.packages("psych")
library("psych")
```

```
##
## Attaching package: 'psych'

## The following objects are masked from 'package:ggplot2':
##
##     %+%, alpha
```

```r
pairs.panels(iris[c("Sepal.Length", "Sepal.Width", "Petal.Length", "Petal.Width")])
```



```r
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Versicolor  : Factor w/ 2 levels "other","versicolor": 1 1 1 1 1 1 1 1 1 1 ...
```

Use the models to and write functions to calculate:

- Prevalence
- Accuracy
- Error Rate / Misclassification Rate
- True Positive Rate

- False Positive Rate
- True Negative Rate

- False Negative Rate
- Sensitivity
- Specificity
- Recall
- Precision

The functions should take two logical vectors of the same length, `y` and `yhat`

```
# EXAMPLE: fpr
# The FPR is THE NUMBER OF FALSE POSITIVES / NEGATIVES (TN+FP)

threshold = 0.5
y = test$Versicolor == 'versicolor' # y is the positive sample in this case

# (RW) if probability is greater than .5 then TRUE meaning is versicolor
yhat = predict(fit.glm, test, type="response") > threshold
#yhat2 = predict(fit.glm, test, type="response")

#install.packages("gmodels")
library("gmodels")

# important to note: CrossTable uses y, yhat (actual, predicted) as order of arguments BUT confusionMat
CrossTable(y, yhat, prop.chisq = FALSE, prop.c = FALSE, prop.r = FALSE, dnn = c('actual versicolor','pr
```

```
##
##
##    Cell Contents
## |-------------------------|
## |                       N |
## |         N / Table Total |
## |-------------------------|
##
##
## Total Observations in Table:  75
##
##
##                    | predicted versicolor
## actual versicolor |     FALSE |      TRUE | Row Total |
## -------------------|-----------|-----------|-----------|
##              FALSE |        41 |         8 |        49 |
##                    |     0.547 |     0.107 |           |
## -------------------|-----------|-----------|-----------|
##               TRUE |        12 |        14 |        26 |
##                   |     0.160 |     0.187 |           |
## ------------------|-----------|-----------|-----------|
##      Column Total |        53 |        22 |        75 |
## ------------------|-----------|-----------|-----------|
##
##
```

```
#*Remember confusion matrix uses (predicted, actual) as order of operations
confusionMatrix(yhat, y, positive = "TRUE")
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction FALSE TRUE
##      FALSE    41   12
##      TRUE      8   14
##
##                Accuracy : 0.7333
##                  95% CI : (0.6186, 0.8289)
##     No Information Rate : 0.6533
##     P-Value [Acc > NIR] : 0.08913
##
##                   Kappa : 0.3893
##  Mcnemar's Test P-Value : 0.50233
##
##             Sensitivity : 0.5385
##             Specificity : 0.8367
##          Pos Pred Value : 0.6364
##          Neg Pred Value : 0.7736
##              Prevalence : 0.3467
##          Detection Rate : 0.1867
##    Detection Prevalence : 0.2933
##       Balanced Accuracy : 0.6876
##
##        'Positive' Class : TRUE
##
```

```
#PREVALENCE
prevalence <- function(y,yhat) {
  actual_p <- sum(y)
  total <- length(y)
  result <- actual_p / total # Total Actual P / Total
  result
}

prevalence(y,yhat)
```

```
## [1] 0.3466667
```

```
#FPR
fpr <- function(y,yhat)
  sum(y & (y != yhat) ) / # FP
```

```r
  sum(! y)                 # Total Actual N

fpr(y,yhat)
```

```
## [1] 0.244898
```

```r
#TPR
tpr <- function(y,yhat)
  sum(y & (y == yhat) ) / # TP
  sum(y)                 # Total Actual P

tpr(y,yhat)
```

```
## [1] 0.5384615
```

```r
#TNR
tnr <- function(y,yhat)
  sum(!y & (y == yhat) ) / # TN
  sum(!y)                 # Total Actual N

tnr(y,yhat)
```

```
## [1] 0.8367347
```

```r
#Sensitivity
sensitivity <- function(y,yhat) {
  tp <- sum(y & (y == yhat))
  fn <- sum(!y & (y != yhat))
  result <- tp / (tp + fn) # TP / TP + FN
  result
}

sensitivity(y,yhat)
```

```
## [1] 0.6363636
```

```r
#Specificity
specificity <- function(y,yhat) {
  tn <- sum(!y & (y == yhat))
  actual_f <- sum(!y)
  result <- tn / actual_f # TN / TN + FP
  result
}

specificity(y,yhat)
```

```
## [1] 0.8367347
```

```r
#Precision
precision <- function(y,yhat) {
  tp <- sum(y & (y == yhat))
  fp <- sum(y & (y != yhat))
  result <- tp / (tp + fp) # TP / TP + FP
  result
}

precision(y,yhat)
```

```
## [1] 0.5384615
```

```r
#Recall
recall <- function(y,yhat) {
  tp <- sum(y & (y == yhat))
  fn <- sum(!y & (y != yhat))
  result <- tp / (tp + fn) # TP / TP + FN
  result
}

recall(y,yhat)
```

```
## [1] 0.6363636
```

```r
#posPredValue(yhat, y, positive = "TRUE")

#ACCURACY
accuracy = function(y,yhat)
  sum(y == yhat) / # TP + TN
  length(y)                 # Total N

accuracy(y,yhat)
```

```
## [1] 0.7333333
```

```r
#ERROR RATE
error_rate = 1 - accuracy(y,yhat)
error_rate
```

```
## [1] 0.2666667
```

- What is wrong with the modeling approach used?

We are using a linear model to predict a class of versicolor instead of using linear model to predit a number. Linear models are used to predict numbers. The code above gets around this by using probabilities >.5 to mean versicolor and <.5 to mean not versicolor, but a classification model should be used instead of linear model.