Evauluation Metrics

W. Evan Johnson, Ph.D.
Professor, Division of Infectious Disease
Director, Center for Data Science
Rutgers University – New Jersey Medical School

7/17/2023

Evaluation metrics

Before we start describing approaches to optimize the way we build algorithms, we first need to define what we mean when we say one approach is better than another.

We use the `caret` package, which has several useful functions for building and assessing machine learning methods.

```r
library(tidyverse)
library(caret)
```

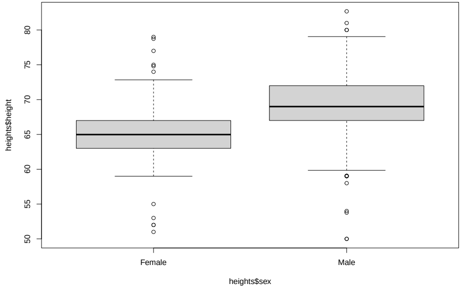
Evaluation metrics

For a first example, we use the height data in dslabs:

```r
library(dslabs)
data(heights)
```

To summarize the data, consider the following boxplot:

```r
boxplot(heights$height~heights$sex)
```



Evaluation metrics

We will start with a simple example: suppose we want to predict sex using height. We start by defining the outcome and predictors.

```
y <- heights$sex
x <- heights$height
```

Evaluation metrics

In this case, we have only one predictor, height, and `y` is clearly a categorical outcome since observed values are either `Male` or `Female`.

We know that we will not be able to predict Y very accurately based on X because male and female average heights are not that different relative to within group variability. But can we do better than guessing? To answer this question, we need a quantitative definition of better.

Training and test sets

Ultimately, a machine learning algorithm is evaluated on how it performs in the real world with completely new datasets. However, we usually have a dataset for which we know the outcomes, as we do with the heights: we know the sex of every student in our dataset. Therefore, to mimic the ultimate evaluation process, we typically split the data into two parts and act as if we don't know the outcome for one of these.

We refer to the group for which we know the outcome, and use to develop the algorithm, as the **training** set. We refer to the group for which we pretend we don't know the outcome as the **test** set.

Training and test sets

The caret package includes the function createDataPartition that helps us generates indexes for randomly splitting the data into training and test sets:

```r
set.seed(2007)
test_index <- createDataPartition(y, times = 1,
                                  p = 0.5, list = FALSE)
```

The argument times is used to define how many random samples of indexes to return, p is used to define what proportion of the data is represented by the index, and list is used to decide if we want the indexes returned as a list or not.

Training and test sets

We can use the result of the `createDataPartition` function call to define the training and test sets like this:

```
test_set <- heights[test_index, ]
train_set <- heights[-test_index, ]
```

We develop an algorithm using **only** the training set. Once we are done developing the algorithm, we will **freeze** it and evaluate it using the test set. The simplest way to evaluate the algorithm when the outcomes are categorical is by simply reporting the proportion of cases that were correctly predicted in the test set. This metric is usually referred to as **overall accuracy**.

Overall accuracy

To demonstrate the use of overall accuracy, we will build two competing algorithms and compare them.

Let's start by developing the simplest possible machine algorithm: guessing the outcome.

```
y_hat <- sample(c("Male", "Female"),
                length(test_index), replace = TRUE)
```

We are completely ignoring the predictor and simply guessing the sex.

Overall accuracy

In machine learning applications, it is useful to use factors to represent the categorical outcomes because R functions developed for machine learning, such as those in the `caret` package, require or recommend that categorical outcomes be coded as factors. So convert `y_hat` to factors using the `factor` function:

```r
y_hat <- sample(c("Male", "Female"),
                length(test_index), replace = TRUE) %>%
  factor(levels = levels(test_set$sex))
```

Overall accuracy

The *overall accuracy* is simply defined as the overall proportion that is predicted correctly:

```
mean(y_hat == test_set$sex)
```

```
## [1] 0.5104762
```

Not surprisingly, our accuracy is about 50%. We are guessing!

Overall accuracy

Can we do better? Exploratory data analysis suggests we can because, on average, males are slightly taller than females:

```
heights %>% group_by(sex) %>%
  summarize(mean(height), sd(height))
```

```
## # A tibble: 2 x 3
##   sex    `mean(height)` `sd(height)`
##   <fct>           <dbl>        <dbl>
## 1 Female           64.9         3.76
## 2 Male             69.3         3.61
```

Overall accuracy

But how do we make use of this insight? Let's try another simple approach: predict `Male` if height is within two standard deviations from the average male:

```r
y_hat <- ifelse(x > 62, "Male", "Female") %>%
  factor(levels = levels(test_set$sex))
```

The accuracy goes up from 0.50 to about 0.80:

```r
mean(y == y_hat)
```

```
## [1] 0.7933333
```

Overall accuracy

But can we do even better? In the example above, we used a cutoff of 62, but we can examine the accuracy obtained for other cutoffs and then pick the value that provides the best results.

But remember, **it is important that we optimize the cutoff using only the training set**: the test set is only for evaluation.

Although for this simplistic example it is not much of a problem, later we will learn that evaluating an algorithm on the training set can lead to **overfitting**, which often results in dangerously over-optimistic assessments.

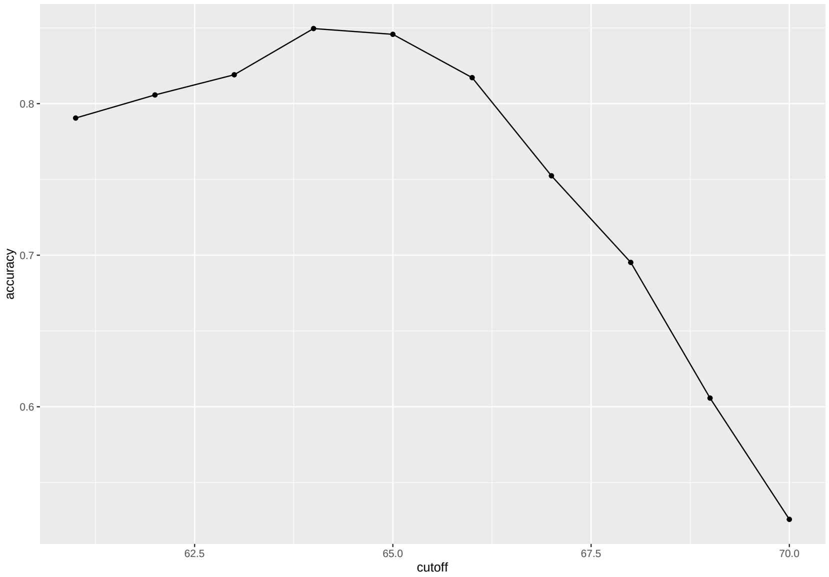
Overall accuracy

Here we examine the accuracy of 10 different cutoffs and pick the one yielding the best result:

```
cutoff <- seq(61, 70)
accuracy <- map_dbl(cutoff, function(x){
  y_hat <- ifelse(train_set$height > x, "Male", "Female") %>%
    factor(levels = levels(test_set$sex))
  mean(y_hat == train_set$sex)
})
```

We can make a plot showing the accuracy obtained on the training set for males and females:

Overall accuracy



Overall accuracy

We see that the maximum value is:

```
max(accuracy)
```

```
## [1] 0.8495238
```

which is much higher than 0.5. The cutoff resulting in this accuracy is:

```
best_cutoff <- cutoff[which.max(accuracy)]
best_cutoff
```

```
## [1] 64
```

Overall accuracy

We can now test this cutoff on our test set to make sure our accuracy is not overly optimistic:

```r
y_hat <- ifelse(test_set$height > best_cutoff,"Male","Female") %>%
  factor(levels = levels(test_set$sex))
y_hat <- factor(y_hat)
mean(y_hat == test_set$sex)
```

```
## [1] 0.8038095
```

We see that it is a bit lower than the accuracy observed for the training set, but it is still better than guessing. And by testing on a dataset that we did not train on, we know our result is not due to cherry-picking a good result.

The confusion matrix

The prediction rule we developed in the previous section predicts `Male` if the student is taller than 64 inches. Given that the average female is about 64 inches, this prediction rule seems wrong. What happened? If a student is the height of the average female, shouldn't we predict `Female`?

The confusion matrix

Generally speaking, overall accuracy can be a deceptive measure. To see this, we will start by constructing what is referred to as the *confusion matrix*, which basically tabulates each combination of prediction and actual value. We can do this in R using the function `table`:

```r
table(predicted = y_hat, actual = test_set$sex)
```

```
##            actual
## predicted Female Male
##    Female     48   32
##    Male       71  374
```

The confusion matrix

If we study this table closely, it reveals a problem. If we compute the accuracy separately for each sex, we get:

```
test_set %>%
  mutate(y_hat = y_hat) %>%
  group_by(sex) %>%
  summarize(accuracy = mean(y_hat == sex))
```

```
## # A tibble: 2 x 2
##   sex    accuracy
##   <fct>     <dbl>
## 1 Female    0.403
## 2 Male      0.921
```

The confusion matrix

There is an imbalance in the accuracy for males and females: too many females are predicted to be male. We are calling almost half of the females male! How can our overall accuracy be so high then? This is because the **prevalence** of males in this dataset is high. These heights were collected from three data sciences courses, two of which had more males enrolled:

```r
prev <- mean(y == "Male")
prev
```

```
## [1] 0.7733333
```

The confusion matrix

So when computing overall accuracy, the high percentage of mistakes made for females is outweighed by the gains in correct calls for men. **This can actually be a big problem in machine learning.** If your training data is biased in some way, you are likely to develop algorithms that are biased as well. The fact that we used a test set does not matter because it is also derived from the original biased dataset. This is one of the reasons we look at metrics other than overall accuracy when evaluating a machine learning algorithm.

The confusion matrix

There are several metrics that we can use to evaluate an algorithm in a way that prevalence does not cloud our assessment, and these can all be derived from the confusion matrix. A general improvement to using overall accuracy is to study **sensitivity** and **specificity** separately.

Sensitivity and specificity

In general, **sensitivity** is defined as the ability of an algorithm to predict a positive outcome when the actual outcome is positive: $\hat{Y} = 1$ when $Y = 1$.

Because an algorithm that calls everything positive ($\hat{Y} = 1$ no matter what) has perfect sensitivity, this metric on its own is not enough to judge an algorithm.

Sensitivity and specificity

For this reason, we also examine **specificity**, which is generally defined as the ability of an algorithm to not predict a positive $\hat{Y} = 0$ when the actual outcome is not a positive $Y = 0$.

Sensitivity and specificity

We name the four entries of the **confusion matrix**:

	Actually Positive	Actually Negative
Predicted positive	True positives (TP)	False positives (FP)
Predicted negative	False negatives (FN)	True negatives (TN)

Sensitivity and specificity

Sensitivity is typically quantified by $TP/(TP+FN)$, the proportion of actual positives (the first column $= TP+FN$) that are called positives (TP). This quantity is referred to as the **true positive rate** (TPR) or **recall**.

Sensitivity and specificity

Specificity is defined as $TN/(TN + FP)$ or the proportion of negatives (the second column $= FP + TN$) that are called negatives (TN). This quantity is also called the true negative rate (TNR).

Sensitivity and specificity

There is another way of quantifying accuracy which is $TP/(TP + FP)$ or the proportion of outcomes called positives (the first row or $TP + FP$) that are actually positives (TP). This quantity is referred to as **positive predictive value (PPV)** and also as **precision**. Note that, unlike TPR and TNR, precision depends on prevalence since higher prevalence implies you can get higher precision even when guessing.

Sensitivity and specificity

The multiple names can be confusing, so we include a table to help us remember the terms. The table includes a column that shows the definition if we think of the proportions as probabilities.

Measure of	Name 1	Name 2	Definition	Probability representation
sensitivity	TPR	Recall	$\frac{\text{TP}}{\text{TP+FN}}$	$\Pr(\hat{Y}=1 \mid Y=1)$
specificity	TNR	1-FPR	$\frac{\text{TN}}{\text{TN+FP}}$	$\Pr(\hat{Y}=0 \mid Y=0)$
Precision	PPV		$\frac{\text{TP}}{\text{TP+FP}}$	$\Pr(Y=1 \mid \hat{Y}=1)$

Here TPR is True Positive Rate, FPR is False Positive Rate, and PPV is Positive Predictive Value.

Sensitivity and specificity

The `caret` function `confusionMatrix` computes all these metrics for us once we define what category "positive" is. The function expects factors as input, and the first level is considered the positive outcome or $Y = 1$. In our example, `Female` is the first level because it comes before `Male` alphabetically. If you type this into R you will see several metrics including accuracy, sensitivity, specificity, and PPV.

```r
cm <- confusionMatrix(data = y_hat,
                      reference = test_set$sex)
```

Sensitivity and specificity

You can acceess these directly, for example, like this:

```
cm$overall["Accuracy"]
```

```
##  Accuracy
## 0.8038095
```

```
cm$byClass[c("Sensitivity","Specificity", "Prevalence")]
```

```
## Sensitivity Specificity  Prevalence
##   0.4033613   0.9211823   0.2266667
```

Sensitivity and specificity

We can see that the high overall accuracy is possible despite relatively low sensitivity. As we hinted at above, the reason this happens is because of the low prevalence (0.23): the proportion of females is low. Because prevalence is low, failing to predict actual females as females (low sensitivity) does not lower the accuracy as much as failing to predict actual males as males (low specificity).

Session Info

```r
sessionInfo()
```

```
## R version 4.2.3 (2023-03-15)
## Platform: aarch64-apple-darwin20 (64-bit)
## Running under: macOS Ventura 13.4.1
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRblas.0.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.2-arm64/Resources/lib/libRlapack.dylib
##
## locale:
## [1] en_US.UTF-8/en_US.UTF-8/en_US.UTF-8/C/en_US.UTF-8/en_US.UTF-8
##
## attached base packages:
## [1] stats     graphics  grDevices utils     datasets  methods   base
##
## other attached packages:
##  [1] dslabs_0.7.6    caret_6.0-94    lattice_0.21-8  lubridate_1.9.2
##  [5] forcats_1.0.0   stringr_1.5.0   dplyr_1.1.2     purrr_1.0.2
##  [9] readr_2.1.4     tidyr_1.3.0     tibble_3.2.1    ggplot2_3.4.3
## [13] tidyverse_2.0.0
##
## loaded via a namespace (and not attached):
##  [1] httr_1.4.7          viridisLite_0.4.2   splines_4.2.3
##  [4] foreach_1.5.2       prodlim_2023.03.31  stats4_4.2.3
##  [7] yaml_2.3.7          globals_0.16.2      ipred_0.9-14
## [10] pillar_1.9.0        glue_1.6.2          pROC_1.18.4
## [13] digest_0.6.33       rvest_1.0.3         hardhat_1.3.0
## [16] colorspace_2.1-0    recipes_1.0.7       htmltools_0.5.6
## [19] Matrix_1.5-4.1      plyr_1.8.8          timeDate_4022.108
## [22] pkgconfig_2.0.3     listenv_0.9.0       webshot_0.5.5
## [25] scales_1.2.1        svglite_2.1.1       gower_1.0.1
```