Caret Package, Training and Test Sets, and Overall Accuracy

Machine Learning - Section 2.1.1

Marc Omar Haddad

17 January, 2020

The `caret` package has several useful functions for building and assessing Machine Learning methods. This section of the course focuses on how Machine Learning algorithms are evaluated.

Our first algorithm will use `height` to predict `sex` (`female` or `male`). We begin by defining the **outcome** and the **predictors**. For this example, we only have one predictor: `height`. Therefore:

```
y = heights$sex # Categorical outcome (male or female)
x = heights$height
```

A Machine Learning algorithm is ultimately evaluated by how it performs in the real world when others run the code. When developing an algorithm we usually have a data set with known outcomes. To mimic the ultimate evaluation process, we split our known data into two separate sets: The **Train Set** and the **Test Set**.

The Train Set is used to *train* our algorithm, and the Test Set is used to *evaluate* it. We evaluate our algorithm by treating the outcomes of our Test Set as *unknown*, training our algorithm with the Train Set, then running our trained algorithm on the data provided in the Test Set, and finally comparing our expected outcomes $\hat{Y}$ to actual outcomes Y.

A common approach to defining our Train and Test sets is by randomly splitting the data. The `caret` package function `createDataPartition` randomly generates indexes to split our data set. The argument `times` defines how many random samples of indexes to return. `p` defines the proportion of the index we wish to split by and `list` is an argument that indicates if we want to be returned a list or not.

```
# Var containing index vals
test_index = createDataPartition(y, times = 1, p = 0.5, list = FALSE)
# The actual splitting of the dataset by rand. gen. index vals.
train_set = heights[-test_index, ]
test_set = heights[test_index, ]
```

Using *only* the Train Set, we will develop our algorithm. Once developed, we test the algorithm (without changing it) on the Test Set. The simplest way to evaluate this particular problem is by using the **Overall Accuracy** metric by computing the proportion of cases that were correctly predicted by our algorithm in the Test Set.

We will now build two competing algorithms and determine which is "better" by comparing their overall accuracy.

First Algorithm

Our first algorithm is the simplest possible Machine Learning algorithm: Guessing.

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

We are completely ignoring our predictor `height` and simply guessing `sex`. For Machine Learning applications it is best to use **factors** to represent **categorical outcomes**. Thus, it is best practice to code categorical outcomes as factors like this:

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

As previously mentioned, **overall accuracy** is defined as the proportion of outcomes predicted correctly.

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

```
## [1] 0.524
```

Unsurprisingly, our accuracy is around **50%**. Can we do better?

Second Algorithm

Exploratory data analysis tells us that males, on average, are taller than females.

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

sex	mean(height)	sd(height)
Female	64.9	3.76
Male	69.3	3.61

For our second algorithm we will predict `male` if `height` is within 2 standard deviations from the average male.

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

mean(y == y_hat) # Proportion of correct predictions
```

```
## [1] 0.793
```

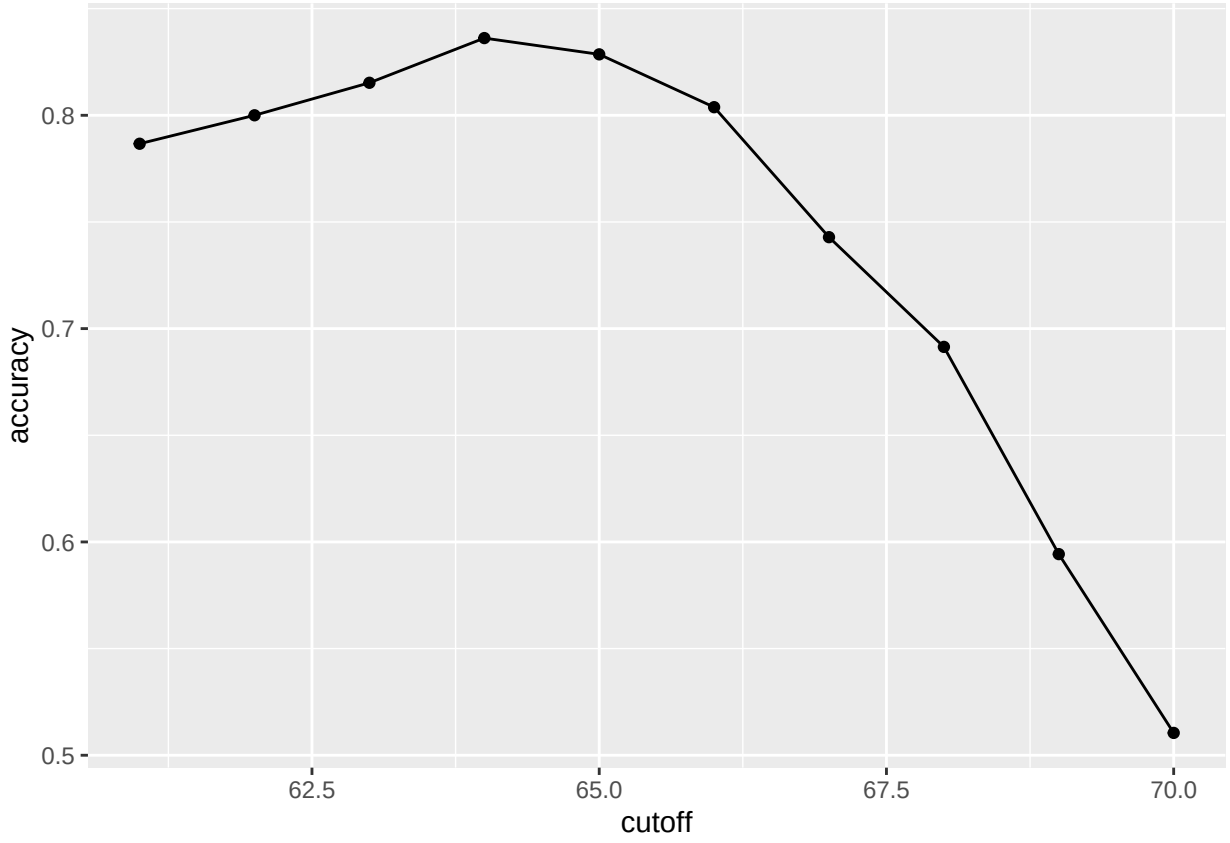
Our accuracy has significantly increased with our second algorithm. We can still do better by examining the accuracy obtained for cutoffs other than 62 inches, and picking the value that produces the best result. We will test the accuracy of 10 different cutoffs on the Train Set.

```
cutoff = seq(61, 70) # Defining sequence of cutoffs
```

```
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)
})
accuracy
```

```
##  [1] 0.787 0.800 0.815 0.836 0.829 0.804 0.743 0.691 0.594 0.510
```

We can see the relationship between the cutoffs and accuracy using a plot.




```
best_cutoff = cutoff[which.max(accuracy)]
best_cutoff
```

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

We can see that the highest accuracy is 0.836, achieved with a cutoff of 64.

We test our best cutoff on our Test Set to ensure our accuracy is not overly optimistic.

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

```
## [1] 0.817
```