Cross-Validation

Brooke Anderson

February 4, 2016

```
library(dplyr) # Data wrangling
library(tidyr) # Data wrangling
library(caret) # Machine learning
library(ggplot2) # Plotting
```

Overview of example

For the cross-validation, I'm going to check out two things about the k-nearest neighbors model using `Survived`, `Pclass`, and `Sex` as predictors. First, I'm going to see how accuracy compares with two choices:

1. Modeling `Pclass` as an ordered versus unordered categorical variable
2. Selecting that all `test` data with missing age be predicted as `0` versus that they be predicted using random draws from a binomial distribution.

Next, I'll use cross validation to tune the `k` parameter for the model.

I'll use cross-validation as a first step for both of these questions and then check the more promising models on Kaggle.

Bring in data

```
train <- read.csv("data/train.csv") %>%
         select(Survived, Pclass, Sex, Age) %>%
         mutate(Survived = factor(Survived),
         Pclass = factor(Pclass),
         Sex = factor(Sex),
         Pclass_ordered = ordered(Pclass))
test <- read.csv("data/test.csv") %>%
        select(Pclass, Sex, Age) %>%
        mutate(Pclass = factor(Pclass),
        Sex = factor(Sex),
        Pclass_ordered = ordered(Pclass))
test_ids <- read.csv("data/test.csv") %>%
            select(PassengerId)
```

I tried out the `train` function from `caret` to do the cross-validation. I specified that the method should be `knn` and that models should be assessed using "Accuracy". I let this function do the preprocessing of centering and scaling.

First, I tried out the model including `Pclass` as an unordered versus ordered variable:

```r
set.seed(16)
knn_mod_1 <- train(Survived ~ Age + Sex + Pclass + Sex:Pclass,
                   data = train,
                   method = "knn",
                   metric = "Accuracy",
                   preProc = c("center", "scale"),
                   tuneLength = 20,
                   trControl = trainControl(method = "cv", number = 10))
knn_mod_1
```

```
## k-Nearest Neighbors
##
## 891 samples
##   4 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered, scaled
## Resampling: Cross-Validated (10 fold)
##
## Summary of sample sizes: 643, 642, 642, 642, 643, 643, ...
##
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa      Accuracy SD  Kappa SD
##    5  0.8096244  0.5961046  0.04440372   0.09694037
##    7  0.8040493  0.5813418  0.04406492   0.09798562
##    9  0.8054186  0.5843500  0.05454688   0.11991976
##   11  0.8039710  0.5809159  0.05167319   0.11368505
##   13  0.7969484  0.5684542  0.05697763   0.12288625
##   15  0.7885759  0.5495193  0.05251114   0.11708055
##   17  0.7773865  0.5253970  0.04914504   0.10904015
##   19  0.7703638  0.5079860  0.04582890   0.10626895
##   21  0.7675665  0.4991597  0.04350810   0.09871277
##   23  0.7661581  0.4948247  0.04780108   0.11029075
##   25  0.7689945  0.5054643  0.04863912   0.10865668
##   27  0.7787559  0.5275012  0.04729385   0.10663247
##   29  0.7759781  0.5213143  0.05575425   0.12504713
##   31  0.7759390  0.5208149  0.05065408   0.11326833
##   33  0.7731612  0.5127020  0.06243310   0.14026905
##   35  0.7773670  0.5206036  0.04980611   0.11389593
##   37  0.7731808  0.5107320  0.05077967   0.11613654
##   39  0.7787754  0.5227161  0.04810850   0.11089025
##   41  0.7732003  0.5077229  0.04920903   0.11506620
##   43  0.7815923  0.5251476  0.05459154   0.12592170
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was k = 5.
```

```r
set.seed(16)
knn_mod_2 <- train(Survived ~ Age + Sex + Pclass_ordered + Sex:Pclass_ordered,
                   data = train,
                   method = "knn",
                   metric = "Accuracy",
                   preProc = c("center", "scale"),
                   tuneLength = 20,
                   trControl = trainControl(method = "cv", number = 10))
knn_mod_2
```

```
## k-Nearest Neighbors
##
## 891 samples
##   4 predictor
##   2 classes: '0', '1'
##
## Pre-processing: centered, scaled
## Resampling: Cross-Validated (10 fold)
##
## Summary of sample sizes: 643, 642, 642, 642, 643, 643, ...
##
## Resampling results across tuning parameters:
##
##   k   Accuracy   Kappa      Accuracy SD  Kappa SD
##    5  0.8096244  0.5965644  0.04440372   0.09685824
##    7  0.8054577  0.5845698  0.04474126   0.09942389
##    9  0.8082355  0.5903720  0.05512192   0.12109719
##   11  0.8025626  0.5781195  0.05104314   0.11244353
##   13  0.7983568  0.5712527  0.05807600   0.12506228
##   15  0.7899844  0.5523178  0.05394438   0.11985473
##   17  0.7759781  0.5226267  0.04715113   0.10512175
##   19  0.7717723  0.5107220  0.04666495   0.10797222
##   21  0.7647496  0.4931043  0.04078577   0.09310134
##   23  0.7717919  0.5069490  0.05246630   0.12002947
##   25  0.7704030  0.5077544  0.04857173   0.10815023
##   27  0.7843897  0.5397703  0.05090031   0.11477143
##   29  0.7759781  0.5208497  0.05495789   0.12319290
##   31  0.7773474  0.5239314  0.05209493   0.11656784
##   33  0.7703443  0.5061374  0.06085587   0.13689239
##   35  0.7759585  0.5173545  0.04964789   0.11334180
##   37  0.7675469  0.4986756  0.04613235   0.10642548
##   39  0.7759585  0.5167574  0.04497630   0.10424858
##   41  0.7717919  0.5049622  0.04891577   0.11437915
##   43  0.7787754  0.5190922  0.05490802   0.12627146
##
## Accuracy was used to select the optimal model using  the largest value.
## The final value used for the model was k = 5.
```

Fitting `knn` this way seems to have no problems with categorical and ordered variables. Another thing to notice is that, now, I didn't have to get rid of the observations with missing values in the training dataset before I fit the model. It turns out that the pre-processing done by `train` defaults to remove missing values as part of pre-processing.

Note from the help file for `preProcess` in `caret`:

> "Pre-processing transformation (centering, scaling etc.) can be estimated from the training data and applied to any data set with the same variables."

You can also use this pre-processing, if you want, to impute missing data by averaging across nearest neighbors for the missing values. Here's a note on that from the help file:

> "k-nearest neighbor imputation is carried out by finding the k closest samples (Euclidian distance) in the training set. Imputation via bagging fits a bagged tree model for each predictor (as a function of all the others). This method is simple, accurate and accepts missing values, but it has much higher computational cost. Imputation via medians takes the median of each predictor in the training set, and uses them to fill missing values. This method is simple, fast, and accepts missing values, but treats each predictor independently, and may be inaccurate."

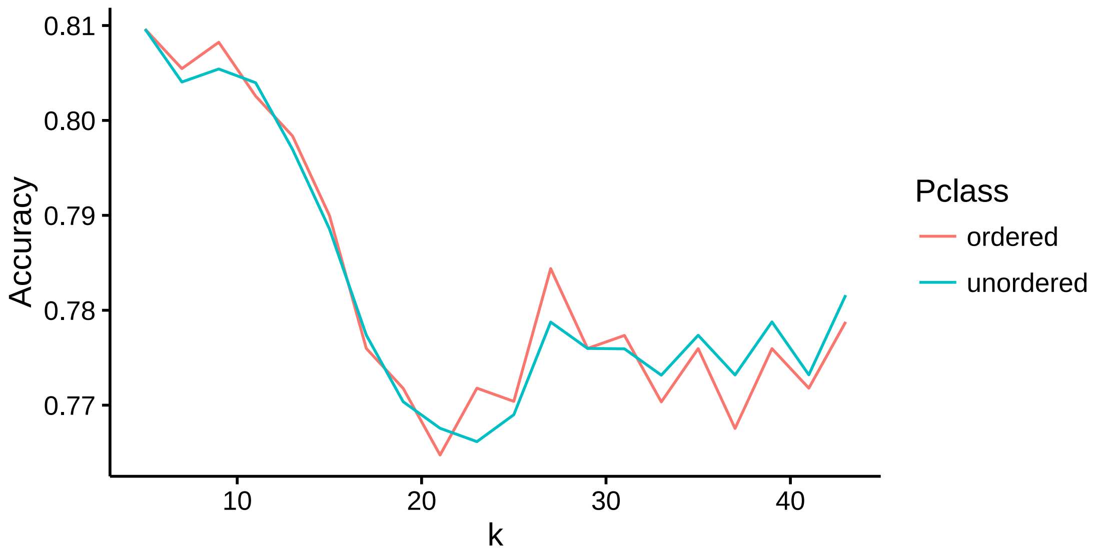
It turns out that `trainControl` has a lot of different options where you can specify which way to validate a model. These include:

- `cv`
- `repeatedcv`
- `boot`
- `boot632`
- `LOOCV`

The `number` option, when used with `method = "cv"`, gives the number of folds to use (i.e., `number = 10` would do 10-fold cross validation).

Here is a plot to compare the two methods (using `Pclass` as unordered versus ordered factor):

```r
to_plot <- data.frame(k = knn_mod_1$results$k,
                      unordered = knn_mod_1$results$Accuracy,
                      ordered = knn_mod_2$results$Accuracy) %>%
  gather(Pclass, Accuracy, -k)
ggplot(to_plot, aes(x = k, y = Accuracy, color = Pclass)) +
  geom_line() + theme_classic()
```



To get values to use for Kaggle, you can use the `predict` method for `train` objects:

```r
test_preds_1 <- predict(knn_mod_1, newdata = test)
head(test_preds_1)
```

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

Remember, since this is a method for an object (using R's object-oriented capabilities), you can get the helpfile for the method by adding `<function>.<object>`:

```r
?predict.train
```

This method is still only giving predictions for observations in the testing data where you have non-missing values for all observations:

```r
nrow(test)
```

```
## [1] 418
```

```r
length(test_preds_1)
```

```
## [1] 332
```

```r
sum(complete.cases(test))
```

```
## [1] 332
```

I'll try to predict anywhere with missing observations by taking a random draw from a binomial, with probability of surviving based on the average in the training dataset:

```r
prob_surv <- mean(train$Survived == 1)
test_preds_miss <- sample(c(0, 1), size = nrow(test),
                       replace = TRUE, prob = c(1 - prob_surv, prob_surv))

test_preds_1a <- test_preds_miss
test_preds_1a[complete.cases(test)] <- as.numeric(test_preds_1) - 1
out <- cbind(test_ids, Survived = test_preds_1a)
write.csv(out, file = "predictions/knn_unorderedPclass.csv",
          row.names = FALSE)

test_preds_2 <- predict(knn_mod_2, newdata = test)
test_preds_2a <- test_preds_miss
test_preds_2a[complete.cases(test)] <- as.numeric(test_preds_2) - 1
out <- cbind(test_ids, Survived = test_preds_2a)
write.csv(out, file = "predictions/knn_orderedPclass.csv",
          row.names = FALSE)
```

When I submitted to Kaggle, I got the following scores: 0.68421 for the one with ordered `Pclass` and 0.68421 for the one with unordered `Pclass`.

I tried this with a few different ways to predict for testing data points that were missing age. First, I predicted "0" for all new observations missing age:

```r
test_preds_1a <- rep(0, nrow(test))
test_preds_1a[complete.cases(test)] <- as.numeric(test_preds_1) - 1
out <- cbind(test_ids, Survived = test_preds_1a)
write.csv(out, file = "predictions/knn_unorderedPclass_all0.csv",
          row.names = FALSE)

test_preds_2a <- rep(0, nrow(test))
test_preds_2a[complete.cases(test)] <- as.numeric(test_preds_2) - 1
out <- cbind(test_ids, Survived = test_preds_2a)
write.csv(out, file = "predictions/knn_orderedPclass_all0.csv",
          row.names = FALSE)
```

Here, the Kaggle results were 0.74641 for the model with ordered `Pclass` and 0.74163 for the model with unordered `Pclass`.

Then, I tried:

```r
prob_surv <- mean(train$Survived == 1)
test_surv <- round(nrow(test) * prob_surv)
test_die <- round(nrow(test) * (1 - prob_surv))
test_preds_miss <- sample(c(rep(0, test_die), rep(1, test_surv)))

test_preds_1a <- test_preds_miss
test_preds_1a[complete.cases(test)] <- as.numeric(test_preds_1) - 1
out <- cbind(test_ids, Survived = test_preds_1a)
write.csv(out, file = "predictions/knn_unorderedPclass_exact.csv",
          row.names = FALSE)

test_preds_2 <- predict(knn_mod_2, newdata = test)
test_preds_2a <- test_preds_miss
test_preds_2a[complete.cases(test)] <- as.numeric(test_preds_2) - 1
out <- cbind(test_ids, Survived = test_preds_2a)
write.csv(out, file = "predictions/knn_orderedPclass_exact.csv",
          row.names = FALSE)
```

Here, the Kaggle results were 0.69378 for the model with ordered `Pclass` and 0.68421 for the model with unordered `Pclass`.

Interesting comment from one of the Kaggle forums about Kaggle accuracy being lower than CV estimates from `caret`:

> "Here are a few possibilities for why the score on the public leaderboard may be lower than the CV estimate. You're right that this can happen if the distributions of feature and outcome values in the training set doesn't reflect the test set. In a very small dataset like this, there may even be some unique cases in the test set that are not like any in the training set. There will certainly be noise in the data – cases with nearly the same feature values, but opposite outcomes. One or the other will not be predicted correctly. If one case is in the training data and the other in test, then if the classifier correctly predicts the case in train, the one in test will be mispredicted. Depending on how CV was used, it may not actually be producing a meaningful estimate of the out-of-sample error. For instance, in the R caret package, CV is used internally in the train() function to do parameter tuning. train() will return the accuracy gotten from CV, but it's safer to do your own CV outside of train(). And. . . even though the accuracy was estimated with CV, there may still be overfitting."