K-Nearest Neighbors

Brooke Anderson

January 20, 2016

Load required libraries.

```
library(dplyr) ## Data wrangling
library(tidyr) ## Data wrangling
library(class) ## Includes `knn` function
library(ggplot2)
```

Read in the data.

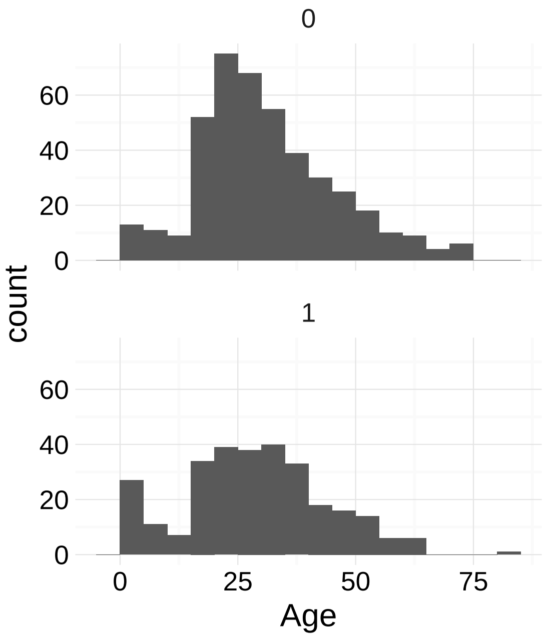
```
train <- read.csv("../data/train.csv") %>%
  mutate(Survived = factor(Survived),
         Pclass = factor(Pclass),
         Name = as.character(Name),
         Sex = factor(Sex))
test <- read.csv("../data/test.csv") %>%
  mutate(Pclass = factor(Pclass),
         Name = as.character(Name),
      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

Model with k = 1 and single continuous predictor

I started by fitting a model with a k of one, so just looking at the single nearest neighbor, and using only `Age` as a predictor. Here's the relationship between `Age` and `Survived` in the training data:

```
ggplot(train, aes(x = Age)) +
  geom_histogram(binwidth = 5) +
  facet_wrap(~ Survived, ncol = 1) +
  theme_minimal()
```

```
## Warning: Removed 177 rows containing non-finite values (stat_bin).
```



Next I fit the model. The model seemed to have some problems when the predictive variable had missing values, so I removed those from both testing and training data sets before fitting.

```
train_knn <- filter(train, !is.na(Age))
train_x <- select(train_knn, Age)
train_y <- train_knn$Survived
test_x <- filter(test, !is.na(Age)) %>%
  select(Age)
```

Next, I fit the model, with k = 1. I fit it first for the training data, to get an estimate of that accuracy. (That means I repeat `train_x` in the model statement.)

```
set.seed(1201)
train_pred <- knn(train_x, train_x, train_y, k = 1)
head(train_pred)
```

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

The `knn` function gives the predictions directly, unlike the function for Naive Bayes, which works more like a typical modeling structure for R.

Now, to assess accuracy, I'll need to get these back into the original dataframe and pick which values I want to use for predictions when `Age` is missing. I'll use "0", since the majority of people in the training dataset did not survive.

This part is a bit tricky, because you have to merge back in with the missing values and align things correctly with the passenger IDs. Normally, I would have kept `PassengerId` in to do this, but `knn` was finicky about letting me do that without modeling it.

```
x_accuracy <- data.frame(Survived = train$Survived,
                         pred = factor("0", levels = c("0", "1")))
x_accuracy$pred[!is.na(train$Age)] <- train_pred
head(cbind(x_accuracy, train$Age))
```

```
##   Survived pred train$Age
## 1        0    0        22
## 2        1    0        38
## 3        1    0        26
## 4        1    1        35
## 5        0    1        35
## 6        0    0        NA
```

```
mean(x_accuracy$Survived == x_accuracy$pred)
```

```
## [1] 0.6767677
```

Accuracy for this model in the training set is 0.67677.

Next, I fit the same model to the testing data and check out the accuracy against the Kaggle Leaderboard.

```
set.seed(1201)
test_pred <- knn(train_x, test_x, train_y, k = 1)
test_accuracy <- data.frame(PassengerId = test$PassengerId,
                            Survived = factor("0", levels = c("0", "1")))
test_accuracy$Survived[!is.na(test$Age)] <-
  as.numeric(as.character(test_pred))
write.csv(test_accuracy, file = "../predictions/knn_age_k1.csv",
          row.names = FALSE)
```

My score on Kaggle was 0.57416. This was my worst model to date.

Functions to save time

I'm realizing now that fitting different models could involve a lot of repetition, so I'll write some functions to save myself time. The following function can take different values of `k` and sets of `predictors`. Then it will fit a `knn`, generate the accuracy of the model in the training dataset, and write a file with predictions for the testing dataset to submit to Kaggle.

```
my_knn <- function(k, predictors, out_file = NULL){
  # Prep the training dataset
  train_knn <- train[ , c("Survived", predictors)]
  train_knn <- train_knn[complete.cases(train_knn), ]
  train_x <- as.data.frame(train_knn[ , predictors])
  colnames(train_x) <- predictors
  model_formula <- as.formula(paste("~",
                                    paste(predictors, collapse = " + ")))
  train_x <- model.matrix(model_formula, data = train_x)[ , -1]
  train_x <- apply(as.data.frame(train_x), 2, scale)  ## Scale predictors
  train_y <- train_knn$Survived

  # Prep the testing dataset
  test_x <- as.data.frame(test[ , predictors])
  test_x <- as.data.frame(test_x[complete.cases(test_x), ])
  colnames(test_x) <- predictors
  test_x <- model.matrix(model_formula, data = test_x)[ , -1]
  test_x <- apply(as.data.frame(test_x), 2, scale) ## Scale predictors
  # Fit knn for the training data and determine accuracy
  set.seed(1201)
  train_pred <- knn(train_x, train_x, train_y, k = k)

  x_accuracy <- data.frame(Survived = train$Survived,
                           pred = factor("0", levels = c("0", "1")))
  x_accuracy$pred[complete.cases(train[ , c("Survived", predictors)])] <-
    train_pred
  out <- mean(x_accuracy$Survived == x_accuracy$pred)

  # Format file name for output file
  if(is.null(out_file)){  ## If out_file is not specified, generate
    out_file <- paste("knn", paste(predictors, collapse = "_"), k,
                      sep = "_")
  }
  out_file <- paste0("../predictions/", out_file, ".csv")

  # Fit knn for the testing data and write out file to submit to Kaggle
  set.seed(1201)
  test_pred <- knn(train_x, test_x, train_y, k = k)
  test_accuracy <- data.frame(PassengerId = test$PassengerId,
                              Survived = factor("0", levels = c("0", "1")))
  test_accuracy$Survived[complete.cases(test[ , predictors])] <-
    as.numeric(as.character(test_pred))
  write.csv(test_accuracy, file = out_file, row.names = FALSE)

  return(paste("Accuracy in the training data is:", round(out, 5)))
}
```

Now it's much easier to test some different knn models. For example, to fit a model with k = 1 with "Fare" as the predictor:

```
my_knn(k = 1, predictors = "Fare")
```

```
## [1] "Accuracy in the training data is: 0.80696"
```

I now know the accuracy in the training dataset, and there's a new file in my `predictions` folder called "knn_Fare_1.csv" that I can submit to Kaggle. When I did, the accuracy was 0.64115. Again, with k = 1, there's a big reduction from the accuracy in testing to that in training.

I can also use it to fit multiple predictors. For example, here's a model with Fare, Age, and SibSp (Number of Siblings/Spouses Aboard):

```
my_knn(k = 5, predictors = c("Fare", "Age", "SibSp"))
```

```
## [1] "Accuracy in the training data is: 0.76431"
```

The function wrote a file to "predictions" called "knn_Fare_Age_SibSp_5.csv". When I submitted this to Kaggle, the accuracy was 0.61244.

Including categorical variables

The function `knn` seems pretty picky about including categorical variables in the model. Also, you can't `scale` categorical variables easily while they're still as a single column of factors. That's okay, though.

To make categorical variables work, you can use `model.matrix`, which can take a formula statement and returns the matrix that you'd want to put into a model, with all categorical variables converted to one or more columns of indicator variables. For example, if you were interested in fitting `Sex` and `Pclass`, here's what model matrix would do:

```r
ex <- model.matrix(~ Sex + Pclass, data = train)
head(ex, 3)
```

```
##   (Intercept) Sexmale Pclass2 Pclass3
## 1           1       1       0       1
## 2           1       0       0       0
## 3           1       0       0       1
```

```r
# Take of intercept column (always first column)
ex <- ex[ , -1]
head(ex, 3)
```

```
##   Sexmale Pclass2 Pclass3
## 1       1       0       1
## 2       0       0       0
## 3       0       0       1
```

You can scale this just like a numerical variable now.

So, for example, to fit a model with `Sex` and `Pclass` with `k = 5`, you could run the following code:

```r
# Set up choices
predictors <- c("Sex", "Pclass")
k <- 5

# Limit to complete cases with the outcome and selected predictors
train_knn <- train[ , c("Survived", predictors)]
train_knn <- train_knn[complete.cases(train_knn), ]
train_x <- as.data.frame(train_knn[ , predictors])

# Convert to the model matrix to handle categorical variables
colnames(train_x) <- predictors
(model_formula <- as.formula(paste("~",
                                paste(predictors, collapse = " + "))))
```

```
## ~Sex + Pclass
```

```r
train_x <- model.matrix(model_formula, data = train_x)[ , -1]
head(train_x, 3)
```

```
##   Sexmale Pclass2 Pclass3
## 1       1       0       1
## 2       0       0       0
## 3       0       0       1
```

```r
# Scale predictors
train_x <- apply(train_x, 2, scale)
head(train_x, 3)
```

```
##        Sexmale    Pclass2    Pclass3
## [1,]  0.737281 -0.5098652  0.9020807
## [2,] -1.354813 -0.5098652 -1.1073041
## [3,] -1.354813 -0.5098652  0.9020807
```

```r
# Create vector of outcomes
train_y <- train_knn$Survived

# Fit model
set.seed(1201)
train_pred <- knn(train_x, train_x, train_y, k = k)

# Assess accuracy in training data
# Start with merging back into the full dataset (including with missing
# observations)
x_accuracy <- data.frame(Survived = train$Survived,
                         pred = factor("0", levels = c("0", "1")))
x_accuracy$pred[complete.cases(train[ , c("Survived", predictors)])] <-
    train_pred

# Calculate accuracy
out <- mean(x_accuracy$Survived == x_accuracy$pred)
```

All of this is set up in the function I wrote, so you could just run:

```r
my_knn(k = 5, predictors = c("Sex", "Pclass"))
```

```
## [1] "Accuracy in the training data is: 0.77441"
```

When I ran this through Kaggle, the accuracy was 0.76555, which was my best score yet.

Role of `k` choice

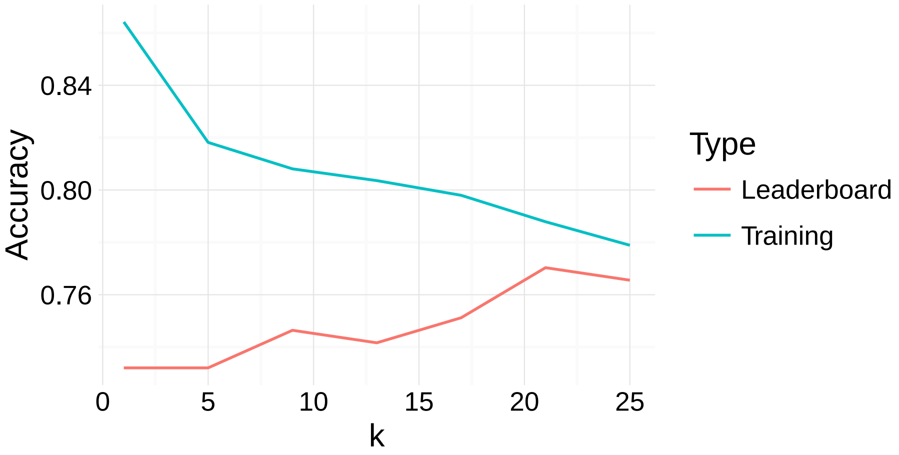
Next, I tried fitting a knn of `Sex`, `Pclass`, and `Age` using `k` values of 1, 5, 9, 13, 17, and 21:

```r
for(k in seq(1, 25, by = 4)){
  out <- my_knn(k = k, predictors = c("Sex", "Pclass", "Age"))
  cat("k =", k, out, "\n")
}
```

```
## k = 1 Accuracy in the training data is: 0.8642
## k = 5 Accuracy in the training data is: 0.81818
## k = 9 Accuracy in the training data is: 0.80808
## k = 13 Accuracy in the training data is: 0.80359
## k = 17 Accuracy in the training data is: 0.79798
## k = 21 Accuracy in the training data is: 0.78788
## k = 25 Accuracy in the training data is: 0.7789
```

Here's a graph showing how the accuracy of the model changes, both within the training and testing (Kaggle leaderboard) data, as k changes:

```
diff_ks <- data.frame(k = seq(1, 25, by = 4),
                      Training = c(0.86420, 0.81818, 0.80808,
                                   0.80359, 0.79798, 0.78788,
                                   0.77890),
                      Leaderboard = c(0.73206, 0.73206, 0.74641,
                                         0.74163, 0.75120, 0.77033,
                                         0.76555))
diff_ks <- gather(diff_ks, Type, Accuracy, -k)
ggplot(diff_ks, aes(x = k, y = Accuracy, group = Type, color = Type)) +
  geom_line() + theme_minimal()
```


Accuracy
k
Type
Leaderboard
Training