K-Nearest-Neighbor Classification

Daniel Rozovsky

```
library(dplyr)
library(ggplot2)
library(class)
library(polycor)
```

Introduction

Hello fellow data scientists and data enthusiasts!

The world around us has tons of cool tendenicies and patterns, ones that we would probably never identify without data science. Luckily for us, we are in the age of machine learning and predictive modeling, where we can train a model to make predictions about the real world that can lead humans to stunning conclusions. While this task may seem daunting at first, machine learning can actually be fun as well as incredibly informative.

In this post we will tackle an example, start to finish, of a type of machine learning called **K-Nearest-Neighbor Classification**. Broadly, classification is when we take a dataset, and put a variable of interest into a certain category based on a set of predictors.

Important: before we proceed, please install the packages at the top of this post, and load them into R. The version of R as well as the version of the packages I am using are printed below.

```
# Information about the R session and packages being used

sessionInfo()
```

```
## R version 3.3.2 (2016-10-31)
## Platform: x86_64-apple-darwin13.4.0 (64-bit)
## Running under: OS X Mavericks 10.9.5
##
## 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] polycor_0.7-9 class_7.3-14  ggplot2_2.2.1 dplyr_0.7.3
##
## loaded via a namespace (and not attached):
##  [1] Rcpp_0.12.9      knitr_1.17       bindr_0.1        magrittr_1.5
##  [5] munsell_0.4.3    colorspace_1.3-2 R6_2.2.0         rlang_0.1.2
##  [9] plyr_1.8.4       stringr_1.2.0    tools_3.3.2      grid_3.3.2
## [13] gtable_0.2.0     htmltools_0.3.6  lazyeval_0.2.0   yaml_2.1.14
## [17] assertthat_0.1   rprojroot_1.2    digest_0.6.12    tibble_1.3.4
## [21] bindrcpp_0.2     glue_1.1.1       evaluate_0.10.1  rmarkdown_1.6
## [25] stringi_1.1.5    scales_0.5.0     backports_1.1.0  pkgconfig_2.0.1
```

Motivation

The first motivation for this post is to introduce a machine learning technique, as I believe this will become the cornerstone of many jobs in the data science community. The second motivation is to present and help the reader discover the nuanced approach to classification, specifically **K-Nearest-Neighbor Classification**, and to see how powerful of a predictive method it can be.

Background Information

Okay, so I've mentioned classification, but now let me define it through the classical machine learning terms. In the basic sense, classification is an example of supervised learning, which means that we know both the inputs and the outputs, and we are teaching our classifier based on known quantities within our dataset. Now, K-Nearest-Neighbor Classification is when we classify data points based on euclidean distance. Let's go through a quick easy example that will give us a flavor for K-Nearest-Neighbor Classification.

Basic Example Describing K-Nearest Neighbor Classification

For this illustration, we will use the built-in iris dataset.

```
head(iris)
```

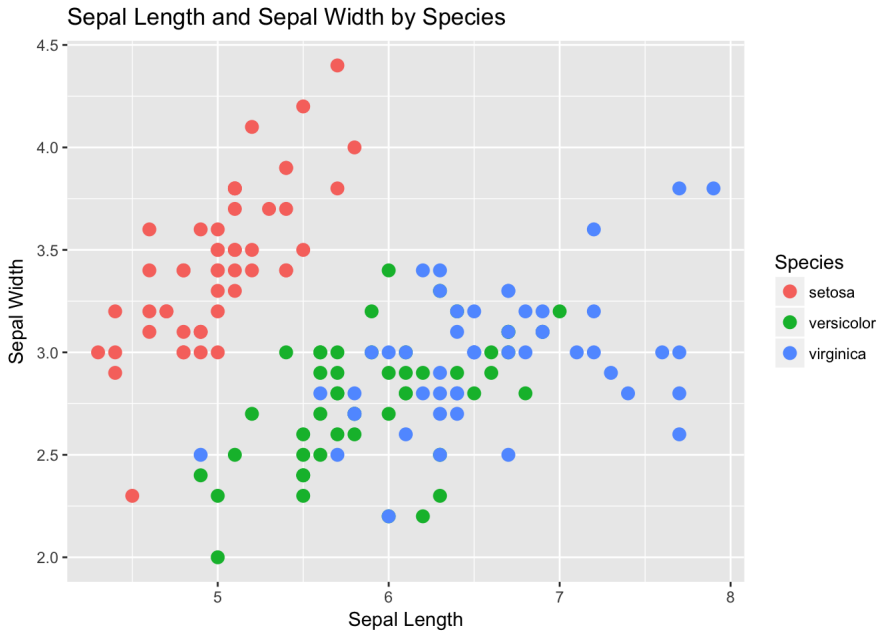
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

We see that the iris dataset has five columns, the first four being numeric descriptions (length of the sepal, width of the sepal, length of the petal, and width of the petal). The fifth column is the species and is either a setosa, versicolor, or virginica.

Let's plot two random variables in this dataset and color the plot based on the species:

```
#plotting Sepal.Length versus Sepal.Width by Species

ggplot(data = iris, aes(Sepal.Length, Sepal.Width, color = Species)) +
  geom_point(size = 3) + ggtitle('Sepal Length and Sepal Width by Species') +
  labs(x = 'Sepal Length', y =  'Sepal Width')
```



What do we notice from this plot? Well, it appears that there are three clustered areas. We can employ K-Nearest-Neighbor Classification to try and figure out if we added a new data point, which of the three clusters would it belong to (i.e. classifying it into one of the three subgroups: setosa, versicolor, or virginica).

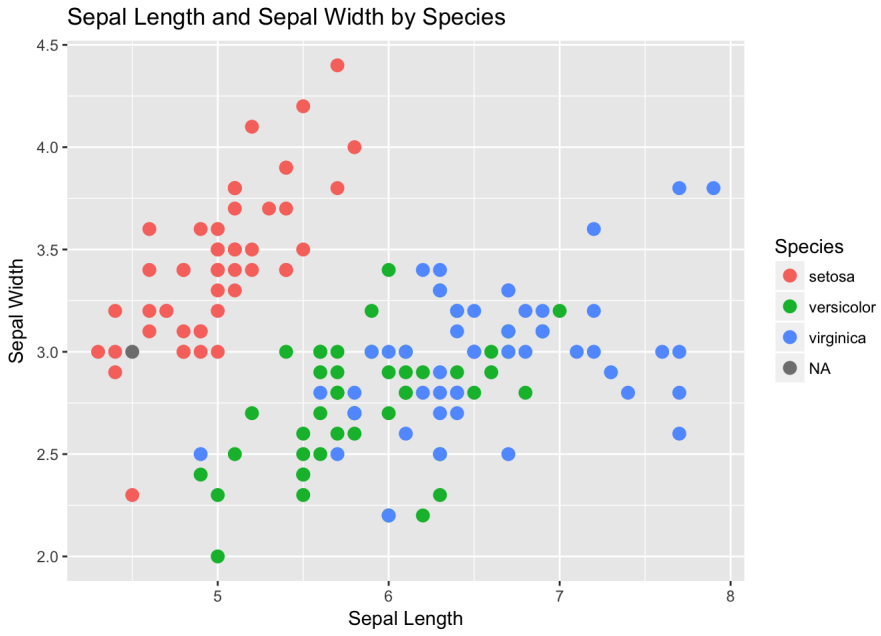
To illustrate this point, let's add another row to the iris dataset, providing values for the four numeric columns, but leaving the species as "unknown" instead of one of the three species it could be.

```
#adding a row to the iris dataset

iris_with_row <- rbind(iris, c(4.5, 3, 1.5, .4, NA))

#plotting new data

ggplot(data = iris_with_row, aes(Sepal.Length, Sepal.Width, color = Species)) +
   geom_point(size = 3) + ggtitle('Sepal Length and Sepal Width by Species') +
   labs(x = 'Sepal Length', y = 'Sepal Width')
```



Now, notice the grey point in the plot. That is the datapoint that we are trying to classify based on the rest of our data points. Now, if we didn't know any better, we could say with some degree of confidence that this `NA` data point is most likely a setosa.

Why do we say that? Well, we see that it looks like the `NA` point is surrounded by setosa data points. Also, we can see that if we purely measure the distance between points, the closest few points to the grey point are all setosas. **This process of measuring distances between points is how we obtain the `nearest neighbors` of the data point. The `K` in K-Nearest-Neighbor Classification is the number of neighbors that we need

in order to be confident in our prediction.

Clarification on distances: The way the distance is measured between points is given by the Euclidean distance formula. Below is the distance formula for two points:

$$distance = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$$

K-Nearest-Neighbor Classification

Now that we've seen a simple example of this type of classification, as well as a picture showing what we are doing, we are going to do a broader example, trying to predict many data points based on their nearest neighbors.

The dataset we will be analyzing is based on the National Basketball Association (NBA). These data can be found by going to https://github.com/ucb-stat133/stat133-fall-2017/tree/master/data, then downloading the `nba-2017-player-statistics.csv` dataset.

Our main goal will be to try to classify players by position, based on predictors such as points scored and salary earned.

Let's get familiarized with the dataset before we get started.

The NBA Players Statistics Dataset

In order to help you follow along, use the following code to download a copy of the dataset into your working directory.

```
#reading in NBA Player Statistics data

github <- 'https://raw.githubusercontent.com/ucb-stat133/stat133-fall-2017/master/'
datafile <- 'data/nba2017-player-statistics.csv'
download.file(paste0(github, datafile), destfile = 'nba2017-player-statistics.csv')
nba_data <- read.csv('nba2017-player-statistics.csv')

#now lets take a look at the dataset

head(nba_data)
```

```
##              Player Team Position Experience   Salary Rank Age GP GS  MIN
## 1        Al Horford  BOS        C          9 26540100    4  30 68 68 2193
## 2      Amir Johnson  BOS       PF         11 12000000    6  29 80 77 1608
## 3     Avery Bradley  BOS       SG          6  8269663    5  26 55 55 1835
## 4 Demetrius Jackson  BOS       PG          R  1450000   15  22  5  0   17
## 5       Gerald Green  BOS       SF          9  1410598   11  31 47  0  538
## 6     Isaiah Thomas  BOS       PG          5  6587132    1  27 76 76 2569
##   FGM  FGA Points3 Points3_atts Points2 Points2_atts FTM FTA OREB DREB AST
## 1 379  801      86          242     293          559 108 135   95  369 337
## 2 213  370      27           66     186          304  67 100  117  248 140
## 3 359  775     108          277     251          498  68  93   65  269 121
## 4   3    4       1            1       2            3   3   6    2    2   3
## 5  95  232      39          111      56          121  33  41   17   68  33
## 6 682 1473     245          646     437          827 590 649   43  162 449
##   STL BLK  TO
## 1  52  87 116
## 2  52  62  77
## 3  68  11  88
## 4   0   0   0
## 5   9   7  25
## 6  70  13 210
```

```
str(nba_data)
```

```
## 'data.frame':    441 obs. of  24 variables:
##  $ Player      : Factor w/ 441 levels "A.J. Hammons",..: 5 16 30 103 146 161 169 181 189 213 ...
##  $ Team        : Factor w/ 30 levels "ATL","BOS","BRK",..: 2 2 2 2 2 2 2 2 2 2 ...
##  $ Position    : Factor w/ 5 levels "C","PF","PG",..: 1 2 5 3 4 3 4 5 4 2 ...
##  $ Experience  : Factor w/ 19 levels "1","10","11",..: 18 3 15 19 18 14 13 11 19 15 ...
##  $ Salary      : num  26540100 12000000 8269663 1450000 1410598 ...
##  $ Rank        : int  4 6 5 15 11 1 3 13 8 10 ...
##  $ Age         : int  30 29 26 22 31 27 26 21 20 29 ...
##  $ GP          : int  68 80 55 5 47 76 72 29 78 78 ...
##  $ GS          : int  68 77 55 0 0 76 72 0 20 6 ...
##  $ MIN         : int  2193 1608 1835 17 538 2569 2335 220 1341 1232 ...
##  $ FGM         : int  379 213 359 3 95 682 333 25 192 114 ...
##  $ FGA         : int  801 370 775 4 232 1473 720 58 423 262 ...
##  $ Points3     : int  86 27 108 1 39 245 157 12 46 45 ...
##  $ Points3_atts: int  242 66 277 1 111 646 394 35 135 130 ...
##  $ Points2     : int  293 186 251 2 56 437 176 13 146 69 ...
##  $ Points2_atts: int  559 304 498 3 121 827 326 23 288 132 ...
##  $ FTM         : int  108 67 68 3 33 590 176 6 85 26 ...
##  $ FTA         : int  135 100 93 6 41 649 217 9 124 37 ...
##  $ OREB        : int  95 117 65 2 17 43 48 6 45 60 ...
##  $ DREB        : int  369 248 269 2 68 162 367 20 175 213 ...
##  $ AST         : int  337 140 121 3 33 449 155 4 64 71 ...
##  $ STL         : int  52 52 68 0 9 70 72 10 35 26 ...
##  $ BLK         : int  87 62 11 0 7 13 23 2 18 17 ...
##  $ TO          : int  116 77 88 0 25 210 79 4 68 39 ...
```

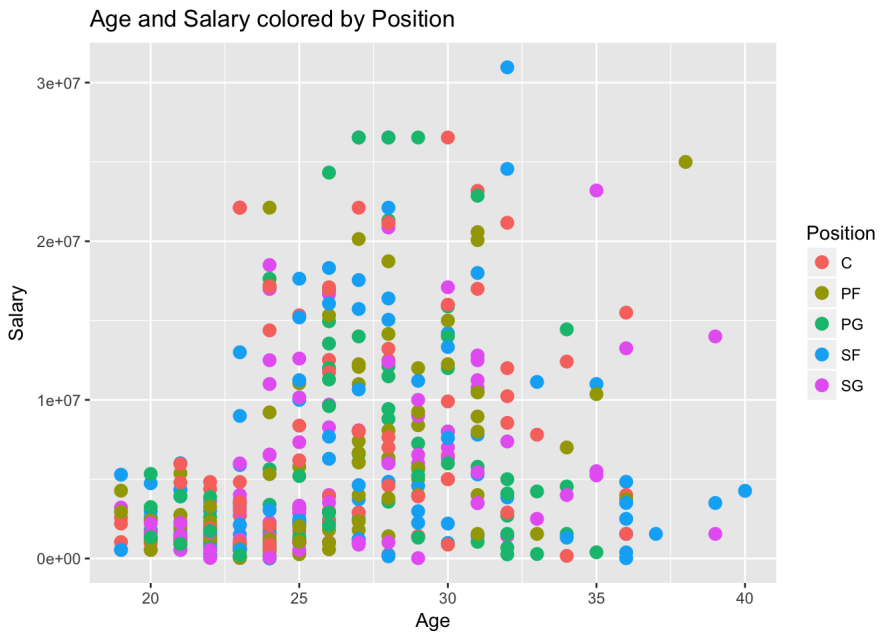
We see that we have statistics ranging from `age` to `salary` to `experience` for all 441 players in the NBA. Wouldn't it be cool if we could classify these players' positions based on some of these other statistics?

In order to start this process, let's do a bit of exploration. Let's plot some of the player statistics based on position.

Obtaining Exploratory Plots based on position

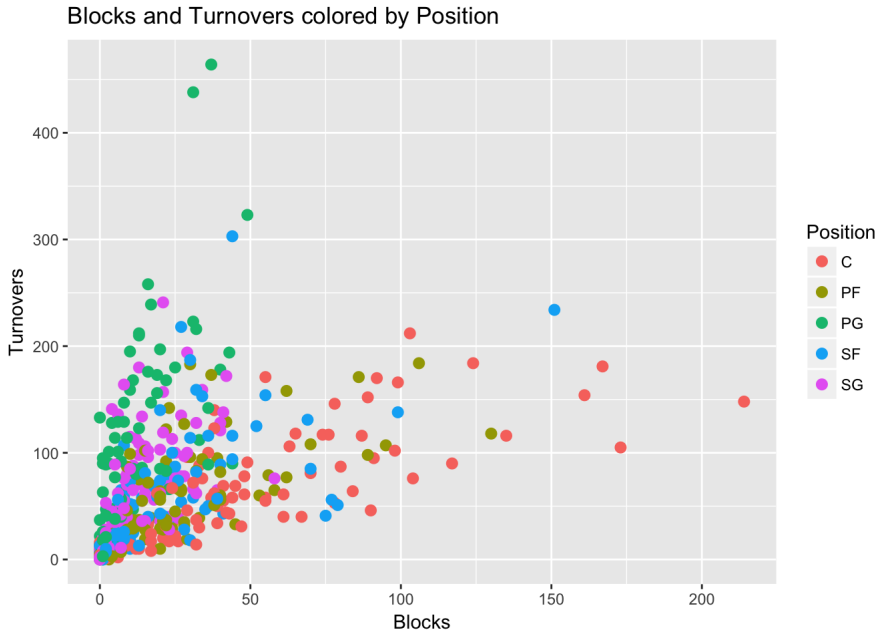
```
# exploratory plot of age and salary by position

exp_plot_1 <- ggplot(data = nba_data, aes(Age, Salary, color = Position)) +
   geom_point(size = 3) + ggtitle('Age and Salary colored by Position') +
   labs(x = 'Age', y = 'Salary')
exp_plot_1
```



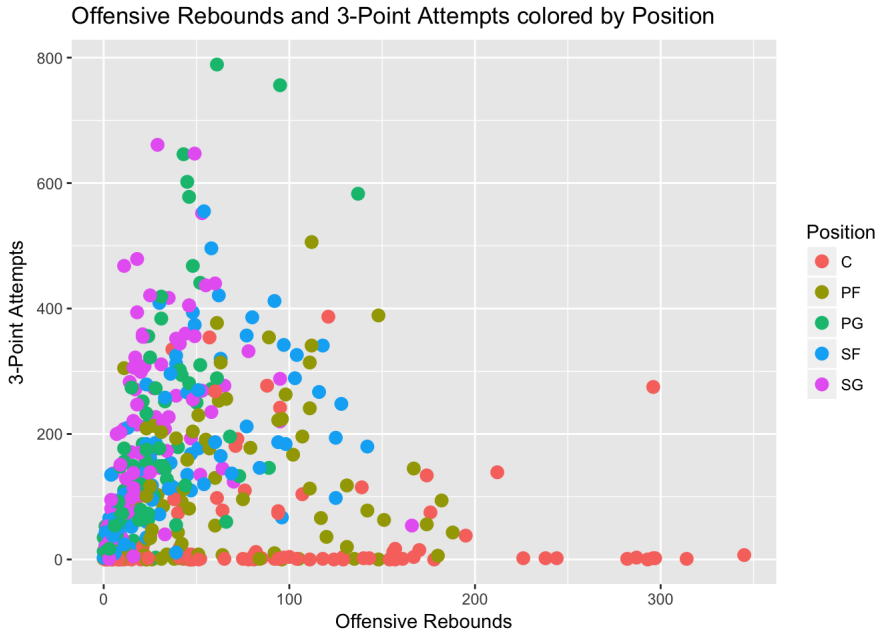

```
# exploratory plot of blocks and turnovers by position

exp_plot_2 <- ggplot(data = nba_data, aes(BLK, TO, color = Position)) +
   geom_point(size = 2.5) + ggtitle('Blocks and Turnovers colored by Position') +
   labs(x = 'Blocks', y = 'Turnovers')
exp_plot_2
```




```
# exploratory plot of Offensive Rebounds and Three Point Attempts

exp_plot_3 <- ggplot(data = nba_data, aes(OREB, Points3_atts, color = Position)) +
 geom_point(size = 3) + ggtitle('Offensive Rebounds and 3-Point Attempts colored by Position') +
labs(x = 'Offensive Rebounds', y = '3-Point Attempts')
exp_plot_3
```



We notice from these exploratory plots that data in the real world (like this basketball dataset) makes it difficult to see patterns in data simply from plotting them. Unlike the `iris` dataset, where there are clearly 3 clumps of data, the NBA dataset lends itself less to such trivial discoveries. Fortunately, this is exactly why we use K-Nearest Neighbor classifiers. We want to see if we can classify the positions of these basketball players based on the data that we have on them.

Variables to use in the test

While it is possible to use all of the data (and hence all of the variables) that we are given to make this classifier, it is sometimes counterproductive to include all of the variables as some of the variables may start to mask true relationships that we are interested in. For this reason, we will find the 5 variables with good correlations between themselves and `Position`, and use them to build our classifier.

You might ask how we will obtain a correlation with position since it is a categorical variable. Well, R has accounted for this problem, and with the help of the package `polycor`, R is able to give us the correlation.

After installing the `polycor` package, we make use of the function `hetcor` which takes in the two variables you want to find the correlation between. While `hetcor` is an impressive function that outputs many interesting details, we will be focused on its first output: the correlation between the two variables. Note that the first output gives four correlations, one for each possibe combination of the two variables. We will be interested in either of the two values (they will always be the same because the correlation between a variable x and a variable y is the same as the correlation between a variable y and a variable x) that are not equal to 1 (i.e. the variables that are different). Below is a list of the five largest correlations with the `position` variable.

```
#obtaining correlations

hetcor(nba_data$OREB, nba_data$Position)[1]
```

```
## $correlations
##                   nba_data$OREB nba_data.Position
## nba_data$OREB         1.0000000        -0.5018538
## nba_data.Position    -0.5018538         1.0000000
```

```
hetcor(nba_data$DREB, nba_data$Position)[1]
```

```
## $correlations
##                   nba_data$DREB nba_data.Position
## nba_data$DREB          1.000000         -0.248873
## nba_data.Position     -0.248873          1.000000
```

```
hetcor(nba_data$Points3, nba_data$Position)[1]
```

```
## $correlations
##                   nba_data$Points3 nba_data.Position
## nba_data$Points3         1.0000000         0.3850585
## nba_data.Position        0.3850585         1.0000000
```

```
hetcor(nba_data$BLK, nba_data$Position)[1]
```

```
## $correlations
##                   nba_data$BLK nba_data.Position
## nba_data$BLK         1.0000000        -0.4223056
## nba_data.Position   -0.4223056         1.0000000
```

```
hetcor(nba_data$FGA, nba_data$Position)[1]
```

```
## $correlations
##                   nba_data$FGA nba_data.Position
## nba_data$FGA         1.0000000         0.1205276
## nba_data.Position    0.1205276         1.0000000
```

Splitting the data into training and testing data

Now that we've identified the five variables we are going to build the classifier with, lets create a `training dataset` and a `testing dataset`. The purpose of creating a training dataset and a testing dataset is so that we can train our classifier on a large portion of the dataset (the training set always has more data than the testing set), and then, test how well our classifier does in terms of accurately predicting position in our test dataset.

```
#subsetting data into only the variables with with high correlations with Position

new_nba_data <- select(nba_data, OREB, DREB, Points3, BLK, FGA, Position)
head(new_nba_data)
```

```
##   OREB DREB Points3 BLK  FGA Position
## 1   95  369      86  87  801        C
## 2  117  248      27  62  370       PF
## 3   65  269     108  11  775       SG
## 4    2    2       1   0    4       PG
## 5   17   68      39   7  232       SF
## 6   43  162     245  13 1473       PG
```

```
#creating training data

train_nba <- new_nba_data[1:350,]
train_nba_1 <- train_nba[,1:5]

#creating testing data

test_nba <- new_nba_data[351:nrow(new_nba_data), 1:5]
```

Notice that neither the training nor the testing dataset contain the positions of the players. This is so that we can test how well we predict those positions, and then compare them to the actual results.

Building the K-Nearest-Neighbor Classifier

Now it is time to make predictions based on the K-Nearest-Neighbors. We will use the `knn` function from the package `class`, so be sure to install and load that package to follow along.

The `knn` function takes in many arguments, however we will focus on the core four:

1. train: the training set
2. test: the testing set
3. cl: the true classifications of the training set
4. k: the number of nearest neighbors that we want to consider

So here's how the process works:

`knn` takes in these four inputs, and then makes predictions. It first takes in the training dataset, which accounts for most of the rows of the nba data that we started with (with only the variables we decided to work with). Then it takes a look at all of the values in that training set (which we removed the positions from) and compares it with the correct values for the positions in the training dataset. This is how the machine learns about our data.

Next, it takes a look at the test dataset, and based on what it learned in the training dataset, it makes predictions for what it thinks the correct outputs will be for position in the test dataset.

The cool thing to note here is that the user (you!) can set the number of nearest neighbors that you want R to consider before making a choice. So you can select `k = 3`, see that it does not predict your model very well, and then switch it to compare the result. The K-Nearest-Neighbor Classifier is illustrated below.

Note: Don't forget to use `set.seed(33)` for this code chunk so that you can follow along and get the same results as shown in this post.

```
set.seed(33)
# Building KNN classifier
nba_classifier <- knn(train = train_nba_1, test = test_nba, cl = train_nba$Position, k = 3)
nba_classifier
```

```
##  [1] PG SG PF PF SF SF PF PF SF C  SG PG SG C  SG C  PG PF PG SF PF SF PF
## [24] SF SG C  C  SF C  SG PG SG SF SG SG PG SG PF PF PF PG PG PF PF PG SG
## [47] PF PG SG PF C  C  PF SF PG SF SG SF SG PG SG SG PG SG SF PF PG C  PF
## [70] SF PG SG C  PF C  PG C  C  PG PF PG C  PG SF SF PG SF PG SF PG C
## Levels: C PF PG SF SG
```

This vector is our vector of predictions for the test dataset! Let's see how accurate we were.

Checking for accuracy

```
#checking our prediction versus the correct results
correct_results <- new_nba_data[351:nrow(new_nba_data), 6]

sum(nba_classifier == correct_results)
```

```
## [1] 32
```

So we see that 32 out of the 91 values in the test dataset were correctly predicted. Perhaps we can do better by changing the number of nearest neighbors we account for.

Note: Don't forget to use `set.seed(34)` for this code chunk so that you can follow along and get the same results as shown in this post.

```
#checking different value for nearest neighbors with a different seed value
set.seed(34)

nba_classifier_better <- knn(train = train_nba_1, test = test_nba, cl = train_nba$Position, k = 7)
nba_classifier_better
```

```
##  [1] PG SG PF PF SF SG SF PF SF PF SG PG SG C  SG SF PG PF SG SF PF SF C
## [24] SF PG PF C  SF C  SG SG SF SF SG PG SG SG SG PF PF PG SG PG PF PG SF
## [47] PF SG PG SF C  C  PF SF PG SF PG SG PG PG SG SG PG SG SG PF PG C  PF
## [70] SF PG SG C  PF C  SG C  C  PG PF PG SG SG PG SF PG PF SG PG PG C
## Levels: C PF PG SF SG
```

```
#checking accuracy

sum(nba_classifier_better == correct_results)
```

```
## [1] 41
```

After checking many different cases, it is clear that the best result this classifier achieves is when it considers 7 nearest neighbors, achieving an accuracy of 41 correct out of 91.

While this result may seem disappointing at first, we can see that it is much better than just randomly guessing. Since there are five positions in basketball, each equally likely, we know that the expected value of randomly guessing correctly is 91/5 which is less than 20. So we more than doubled the number of positions correctly predicted.

Conclusion

To summarize, the purpose of this post was to show the reader a quick look into a basic machine learning technique called `K-Nearest-Neighbor Classification`. We saw a quick demonstration and pictoral representation of how the K-Nearest-Neighbor algorithm

works, and then we used it to classify basketball players' positions based on variables such as rebounding and number of blocks. While this is a simple machine learning algorithm, it is quite useful and applicable in the real world today. Difficult problems arise in businesses and large corporations that require classifying and determining which people belong in which groups. This can drastically help how businesses market to their consumers, as well as what kind of advertisements you see on a company's website. The take-away: Although this is a simple form of classification, K-Nearest-Neighbor Classification is a powerful method that can be used to predict the world around you.

References

Below are the sources that I used to help me write this post:

1. https://machinelearningmastery.com/k-nearest-neighbors-for-machine-learning/
2. https://stat.ethz.ch/R-manual/R-devel/library/class/html/knn.html
3. https://www.datacamp.com/community/tutorials/machine-learning-in-r
4. https://stackoverflow.com/questions/5559384/euclidean-distance-of-two-vectors
5. https://edumine.wordpress.com/2015/04/06/splitting-a-data-frame-into-training-and-testing-sets-in-r/
6. https://github.com/ucb-stat133/stat133-fall-2017/tree/master/data
7. https://stats.stackexchange.com/questions/129846/correlation-between-a-numeric-and-factor-in-r
8. http://www.dummies.com/programming/r/how-to-add-observations-to-a-data-frame-in-r/
9. https://en.wikipedia.org/wiki/Statistical_classification#cite_note-1
10. ps://www.analyticsvidhya.com/blog/2014/10/introduction-k-neighbours-algorithm-clustering/