K-NN con Pokemón

Este análisis presenta el algoritmo de aprendizaje automático K-Nearest Neighbor (K-NN) utilizando el conocido conjunto de datos de Pokémon. Al final de este documento, debes tener una comprensión de lo siguiente:

1. Qué es el algoritmo de aprendizaje automático de K-NN
2. Cómo programar el algoritmo en R
3. Un poco más sobre Pokémon

¿Qué son los Pokémon?

Los Pokémon se usan para luchar entre sí y vienen con una variedad de estadísticas de batalla diferentes como: ataque, defensa, velocidad, ataque especial, defensa especial y puntos de golpe. Los Pokémon también se pueden clasificar en una variedad de tipos diferentes (como: fuego, agua, hierba, eléctrico, hielo, fantasma, oscuro, hada, tierra, etc.). Cuanto más fuerte es un Pokémon, más probable es que gane en la batalla. Hay muchos matices en Pokémon (y esto es solo un resumen básico), pero espero que los detalles específicos de Pokémon no obstaculicen tu comprensión de KNN (el punto central de este documento).

Pregunta de interés?

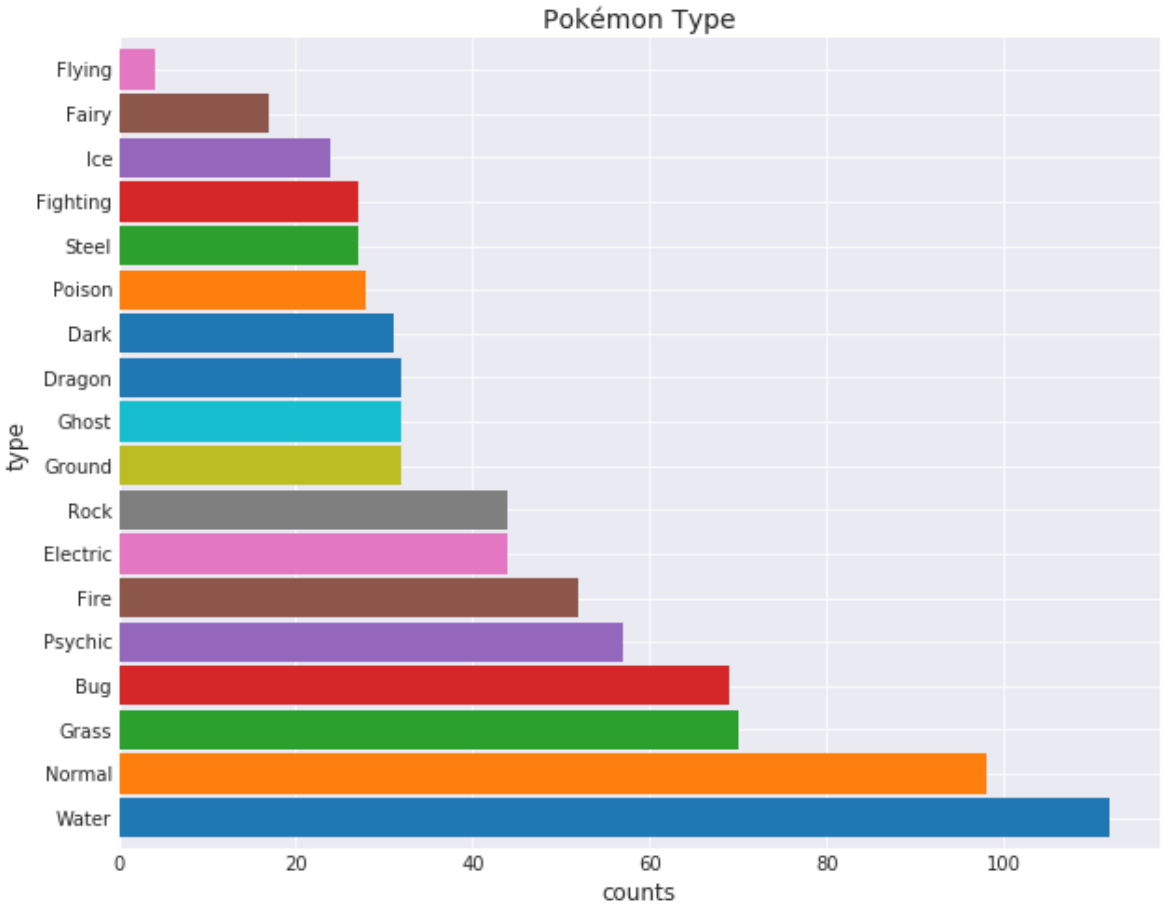
¿Los Pokémon con estadísticas de batalla similares (ataque, defensa, velocidad, etc.) tienden a congregarse por tipo (fuego, agua, hierba, etc.) y, de ser así, podemos adivinar el tipo de Pokémon basándonos solo en sus estadísticas de batalla?

K-vecinos más cercanos (KNN) – por ejemplo

A los efectos de este documento, definimos K-NN como un popular algoritmo de aprendizaje automático supervisado para clasificar datos. En lenguaje coloquial, el algoritmo utiliza la similitud entre los puntos de datos para determinar la pertenencia al grupo de un nuevo punto de datos.

Un ejemplo: digamos que tienes 5 Pokémon y supongamos que en el reino de Pokémon solo hay dos tipos: fuego y agua. Resulta (a partir de nuestros datos de muestra) que todos nuestros Pokémon de tipo fuego (los primeros tres: growlithe, flareon y ponyta) son rojos y los Pokémon de agua restantes (poliwag, vaporeon) son azules.

Estos cinco Pokémon ahora se convertirán en nuestros datos de entrenamiento. Esto significa que los aceptamos como son y sabemos que su clasificación es verdadera (los que son fuego son en realidad fuego y los que son agua son agua). Si ahora te mostrara un nuevo Pokémon (vulpix), llamémoslo los datos de prueba, ¿podrías averiguar a qué grupo pertenece? Agreguemos vulpix a nuestro gráfico de líneas de Pokémon según su coloración:



Intuitivamente, podría clasificar a Vulpix como fuego porque es rojo. ¡Esa es una gran lógica! Ahora, te desafío a modificar un poco tu forma de pensar: ¿qué pasaría si decidieras clasificar este vulpix como fuego porque es más similar a su vecino más cercano en el gráfico de líneas (tal vez eso fue lo que hiciste?). En este caso, vulpix está más cerca de growlithe, flareon y ponyta y más alejado de poliwag y vaporeon. Debido a que sus vecinos son fuego, vulpix ahora es fuego. Este es el pensamiento básico detrás de K-NN: qué tan cerca estoy de mis vecinos y lo que sea que estén clasificados es como estoy clasificado.

Parece sencillo, ¿verdad? Ahora, ¿qué pasa si se te pide que clasifiques ahora los siguientes Pokémon (tentacool):
Esto es complicado, porque tentacool aquí es rojo y azul; ¿qué hacemos? Bueno, parece que hay un poco más de azul en nuestro Pokémon que de rojo, así que hagamos un gráfico de tentacool y muévalos un poco hacia el lado más azul de nuestro gráfico:

Así que ahora pensemos: ¿quiénes son los vecinos más cercanos a tentacool? Bueno, sí solo miramos a un vecino, entonces es poliwag y si miramos a dos vecinos, entonces es poliwag y ponyta. Extender a tres vecinos luego introduce vaporeon y cuatro vecinos trae flareon. Una imagen de esto se puede ver aquí con un vecino (rojo), dos vecinos (verde) y tres vecinos (azul).

En este caso, la forma en que clasificamos nuestro tentacool determinará cuántos vecinos encuestaremos. Si elegimos un vecino, diremos que este pokemon es agua. Si elegimos dos, entonces estamos divididos 50/50 y lanzamos una moneda (¡y dejamos que el azar decida!). Si se seleccionan tres vecinos, entonces tenemos 2 votos para el agua y un voto para el fuego. Digamos que elegimos 3 vecinos; en este caso, clasificaremos tentacool como agua.

Nota: Llegados a este punto te estarás preguntando… ¿qué pasó con nuestro vulpix (el primer pokemon que clasificamos)? ¿Por qué no los incluimos en nuestro nuevo gráfico para ayudar a clasificar tentacool? Esto habla de un tema más amplio de datos de entrenamiento y prueba, pero por ahora, comprende que en realidad no conocemos la clasificación de vulpix. Hicimos una conjetura educada sobre su clasificación. La clasificación de vulpix (para nuestro ejemplo) no se conoce al 100% y como tal, no se incluirán como datos de entrenamiento para tomar decisiones sobre futuros pokemon (como tentacool).

En este ejemplo básico, vimos que si primero trazamos nuestros datos de entrenamiento en función de un atributo (color), podríamos adivinar el tipo de un pokemon desconocido (datos de prueba). Obviamente, el color por sí solo puede no ser un gran indicador del tipo de Pokémon (aunque lo hace sorprendentemente bien), por lo que también podría valer la pena analizar otros factores: velocidad, ataque, defensa, puntos de vida, etc. Lo que vamos a averiguar en el siguiente análisis es si podemos usar dos o más atributos de un pokemon junto con K-NN para ayudar a determinar el tipo de pokemon.

Es posible que tengas preguntas sobre lo siguiente (que se abordarán a lo largo del documento):

1. ¿Qué determina qué tan cerca están los vecinos?
2. ¿Cómo elijo los atributos para agrupar Pokémon en función de?
3. ¿Cuántos vecinos elijo?

¡Está bien! Estas son preguntas que espero cubrir a lo largo de este documento.

Análisis

Para este análisis, necesitaremos las siguientes bibliotecas y estableceremos la semilla:

```
> library("dplyr")
> library("class")
> library("ggplot2")
> library("GGally")
> library("caret")
> set.seed(12345)
```

Nota: No olvides instalarla antes de utilizarla!

Primero, importaremos el conjunto de datos:

```
> pokemon_data <- read.csv("pokemon.csv")
```

Hay un montón de columnas en este conjunto de datos; para que podamos centrar nuestra comprensión en K-NN, solo seleccionemos algunas de las columnas (name, attack, defense, sp_attack, sp_defense, speed, hp, type1, type2). Estas columnas seleccionadas son simplemente los atributos discutidos anteriormente. Podemos obtener una vista previa del marco de datos de la siguiente manera:

```
> reduced_pokemon_data <- pokemon_data %>%
+ select(name, attack, defense, sp_attack, sp_defense, speed, hp, type1, type2)
```

```
> head(reduced_pokemon_data)
        name attack defense sp_attack sp_defense speed hp type1  type2
1  Bulbasaur     49      49        65         65    45 45 grass poison
2    Ivysaur     62      63        80         80    60 60 grass poison
3   Venusaur    100     123       122        120    80 80 grass poison
4 Charmander     52      43        60         50    65 39  fire
5 Charmeleon     64      58        80         65    80 58  fire
6  Charizard    104      78       159        115   100 78  fire flying
```

Por ahora, estas son las columnas con las que trabajaremos. En nuestro conjunto de datos, hay 801 Pokémon únicos y 18 tipos diferentes. Para simplificar un poco el análisis, solo veremos 5 tipos diferentes de Pokémon: bicho, dragón, lucha, eléctrico y normal. Reduciré aún más el conjunto de datos para mirar solo a los Pokémon con un solo tipo (nota: los Pokémon pueden tener hasta dos tipos: por ejemplo: fuego/psíquico, tierra/dragón, etc.):

```
> final_dataset <- reduced_pokemon_data %>%
+ filter( type2 == "" & type1 %in% c( "bug", "dragon", "fighting", "electric", "normal"))
```

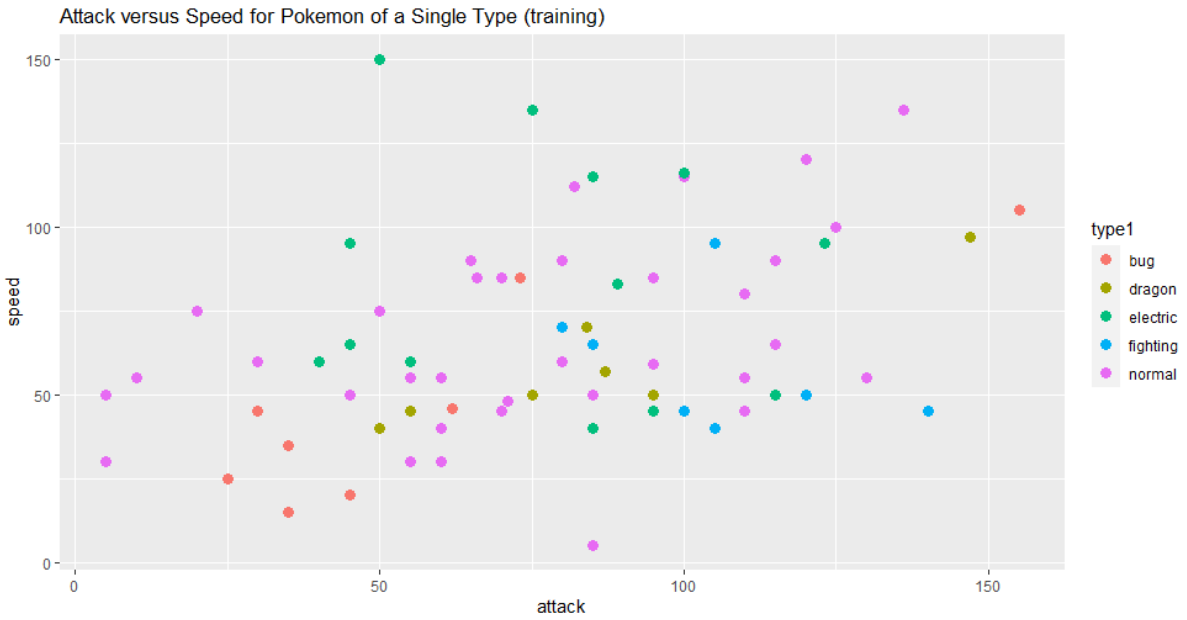
Estos ajustes reducen drásticamente nuestro conjunto de datos de 801 instancias a solo 139. Por desgracia, recordaré al lector que el propósito de este documento es explicar K-NN, no proporcionar un análisis riguroso del conjunto de datos de Pokémon. Dicho esto, eliminemos nuestra columna de tipo 2 (ya que todos los Pokémon en nuestros datos son de un solo tipo) y factoricemos nuestra columna de tipo 1. Luego, dividiremos el conjunto de datos en un conjunto de entrenamiento, validación (que se tratará a continuación) y prueba:

```
> final_dataset <- final_dataset %>%
+  select(-c(type2)) %>%
+  mutate_at(vars(type1), ~(factor(type1)))
> random_rows <- sample(1:nrow(final_dataset), nrow(final_dataset) * .75)
> training_data <- final_dataset[random_rows, ]
> # Renumber the rows
> row.names(training_data) <- 1:nrow(training_data)
> testing_data <- final_dataset[-random_rows, ]
> random_rows_validation <- sample(1:nrow(training_data), nrow(training_data) * .30)
> validation_data <- training_data[random_rows_validation, ]
> training_data <- training_data[-random_rows_validation, ]
```

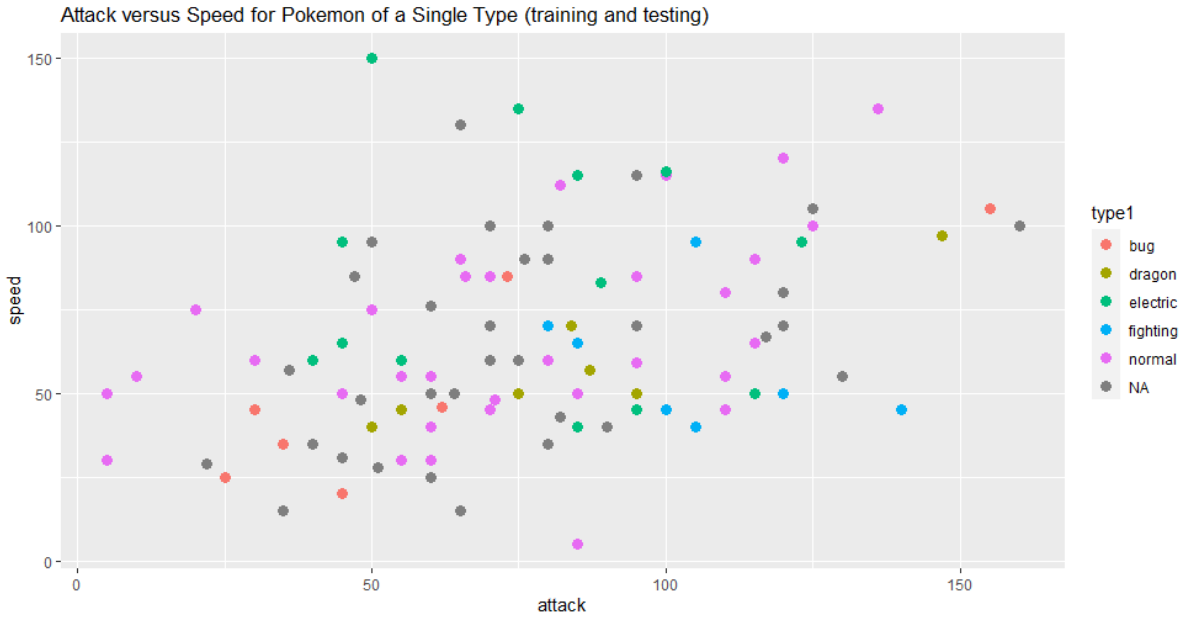
En concreto para este primer análisis nos interesa fijarnos en los stats de ataque y velocidad de nuestros pokemon; un plot puede tener el siguiente aspecto:

```
> ggplot(data = training_data, aes(x = attack, y = speed, col = type1)) +
+  geom_point(size = 3) +
+  ggtitle("Attack versus Speed for Pokemon of a Single Type (training)")
```

En el plot, podemos ver algunos clústeres donde parece que se están reuniendo grupos de Pokémon (por ejemplo, los Pokémon bichos tienden a congregarse con un ataque más bajo y una velocidad más baja, y los Pokémon eléctricos tienden a tener un ataque bajo y una velocidad alta). Incluyamos nuestros Pokémon de prueba (los que queremos clasificar en los distintos grupos):




```
> combined_training_test <- rbind(training_data, testing_data)
> combined_training_test$type1 <- factor(c(as.character(training_data$type1), rep(NA, nrow(testing_data))))
> ggplot(data = combined_training_test, aes(x = attack, y = speed, col = type1)) +
+  geom_point(size = 3) +
+  ggtitle("Attack versus Speed for Pokemon of a Single Type (training and testing)")
```



Para recordarle al lector nuestro objetivo aquí: nos gustaría clasificar cada uno de estos puntos grises en uno de los 5 tipos de Pokémon etiquetados. El algoritmo hará esto

encontrando los k-vecinos más cercanos a cada uno de los puntos de datos grises y clasificándolos según el método de votación descrito anteriormente.

Para responder a nuestra primera pregunta anterior, para determinar qué puntos están más cerca de nuestros datos de prueba desconocidos, el algoritmo K-NN simplemente usará la distancia euclidiana para determinar qué puntos de datos de entrenamiento están más cerca.

A continuación, abordaremos la segunda pregunta: "¿Cómo elijo los atributos en mi modelo?" Bueno, a decir verdad, K-NN puede tomar una cantidad infinita de atributos, ¡no solo dos! Para poder mostrar el plot más fácilmente, decidí mirar solo dos atributos. Para elegir qué atributos seleccioné, observé una serie de diagramas de dispersión y determiné qué dos atributos (visualmente) produjeron las mejores agrupaciones de datos. Encontré este paquete agradable (GGally) que ayuda a visualizar pares de nuestras columnas de una manera agradable. Echemos un vistazo a una serie de diagramas de dispersión para comparaciones de variables por pares:

```
> pairs(final_dataset[, 2:8],col = training_data$type1,lower.panel=NULL)
```

Como puedes ver arriba, el diagrama de dispersión de velocidad/ataque parecía mostrar la mejor separación entre puntos. Técnicamente, podrías usar cualquier número de combinaciones de estos puntos y usar el algoritmo K-NN. Al final de este análisis, mostraré los resultados al usar todos los atributos. Pero primero, permítanme subdividir nuestros datos de entrenamiento para mostrar solo el algoritmo K-NN con solo la velocidad y el ataque utilizados:

```
> training_data_speed_attack <- training_data %>%
```

```
+  select(c(speed,attack))
> testing_data_speed_attack <- testing_data %>%
+  select(c(speed,attack))
```

Genial, ahora que tenemos nuestros atributos seleccionados, programaremos el algoritmo. En lo que respecta a los algoritmos de aprendizaje automático, K-NN es bastante simple en términos de llamada de función:

```
> knn_attack_speed <- knn(train = training_data_speed_attack, test = testing_data_speed_attack, cl = training_data$type1, k = 5)
```

Esto es lo que significan los diversos parámetros de nuestra llamada de función:

train: una matriz o marco de datos de solo los atributos para los datos de entrenamiento
test: una matriz o marco de datos de solo los atributos para los datos de prueba
cl: una lista de las clasificaciones para los datos de entrenamiento (recuerda, estamos tratando de predecir los tipos de Pokémon para nuestros datos de prueba)
k: La cantidad de vecinos a los que nos gustaría "consultar" para determinar el tipo de los nuevos datos de prueba de Pokémon (explicaré por qué elegí 5 a continuación)
knn_attack_speed: Lo que decidimos llamar a nuestra llamada de función. Esta función solo devuelve las etiquetas de clase predichas para nuestros datos de prueba.

Ahora que nuestro algoritmo se ha ejecutado, una de las mejores formas de verificar cómo se desempeñó nuestro algoritmo es construir una matriz de confusión. La matriz de confusión es un buen visual que nos permitirá ver cómo nuestras etiquetas de clase predichas se comparan con nuestras etiquetas de clase reales. La matriz de confusión es la siguiente:

```
> confusionMatrix(knn_attack_speed,testing_data$type1 )
Confusion Matrix and Statistics

          Reference
Prediction bug dragon electric fighting normal
  bug        2      0        1        0      1
  dragon     0      0        0        1      0
  electric   0      0        1        0      1
  fighting   1      0        0        1      1
  normal     2      2        3        7     11

Overall Statistics

               Accuracy : 0.4286
                 95% CI : (0.2632, 0.6065)
    No Information Rate : 0.4
    P-Value [Acc > NIR] : 0.4272

                  Kappa : 0.1422

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: bug Class: dragon Class: electric Class: fighting Class: normal
Sensitivity             0.40000       0.00000         0.20000         0.11111        0.7857
Specificity             0.93333       0.96970         0.96667         0.92308        0.3333
Pos Pred Value          0.50000       0.00000         0.50000         0.33333        0.4400
Neg Pred Value          0.90323       0.94118         0.87879         0.75000        0.7000
Prevalence              0.14286       0.05714         0.14286         0.25714        0.4000
Detection Rate          0.05714       0.00000         0.02857         0.02857        0.3143
Detection Prevalence    0.11429       0.02857         0.05714         0.08571        0.7143
Balanced Accuracy       0.66667       0.48485         0.58333         0.51709        0.5595
```

La matriz de confusión se puede leer de la siguiente manera: cuando la clase real del pokemon era error (5 instancias), K-NN predijo: error (2 veces), lucha (1 vez) y normal (2 veces).

¡En general, a partir de la salida, podemos ver que nuestro modelo funcionó bien! Con una precisión del 42.86 %, podemos decir sin duda que nuestro modelo funcionó mejor que simplemente lanzar un dado de 5 caras (lo que habría dado como resultado una precisión del 20.00 % (100 %/5 clases)). Sin embargo, antes de continuar, hablemos de cómo determinamos qué debería ser K. Para hacer esto, usamos la validación cruzada (que se usará posteriormente). Recordarás que dividimos nuestro conjunto de datos en un conjunto de entrenamiento, prueba y validación: aquí es donde entra en juego el conjunto de validación. Usaremos el conjunto de validación para seleccionar adecuadamente nuestro nivel de K para nuestros datos de entrenamiento que luego evaluaremos en nuestros datos de prueba.

```
> trControl <- trainControl(method  = "cv",
+  number  = 10)
> fit <- train(type1 ~  speed + attack,
+  method     = "knn",
+  tuneGrid   = expand.grid(k = 1:5),
+  trControl  = trControl,
+  metric     = "Accuracy",
+  data       = validation_data)
> fit
k-Nearest Neighbors

31 samples
 2 predictor
 5 classes: 'bug', 'dragon', 'electric', 'fighting', 'normal'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 28, 29, 29, 27, 26, 28, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  1  0.3350000  -0.01774147
  2  0.4433333   0.13273810
  3  0.3516667   0.05808913
  4  0.4566667   0.19089390
  5  0.5066667   0.24827264

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 5.
```

Como podemos ver, el número óptimo de pliegues (folds) en nuestro conjunto de validación fue 5 (basado en la medición de la precisión).

Análisis - Parte 2

Solo por diversión y curiosidad, ¿qué pasaría si decidiéramos usar todos nuestros atributos del Pokémon en lugar de solo mirar la velocidad y el ataque? ¿Cómo cambiaría eso nuestro análisis? Primero determinamos nuestro k óptimo:

```
> # Determine the optimal K
> fit_all_attributes <- train(type1 ~  speed + attack + defense + hp  + sp_attack + sp_defense,
+  method     = "knn",
+  tuneGrid   = expand.grid(k = 1:5),
+  trControl  = trControl,
+  metric     = "Accuracy",
+  data       = validation_data)
> fit_all_attributes

k-Nearest Neighbors

31 samples
 6 predictor
 5 classes: 'bug', 'dragon', 'electric', 'fighting', 'normal'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 29, 29, 28, 27, 27, 28, ...
Resampling results across tuning parameters:

  k  Accuracy   Kappa
  1  0.4833333  0.1915584
  2  0.5000000  0.2445887
  3  0.5416667  0.2743590
  4  0.5583333  0.2772727
  5  0.5333333  0.2300000

Accuracy was used to select the optimal model using the largest value.
The final value used for the model was k = 4.
```

Luego ejecutamos el modelo:

```
> # Perform the KNN
> training_data_all <- training_data %>%
+  select(c(speed,attack, defense, hp, sp_attack, sp_defense))
> testing_data_all <- testing_data %>%
+  select(c(speed,attack, defense, hp, sp_attack, sp_defense))
> knn_all <- knn(train = training_data_all, test = testing_data_all, cl = training_data$type1, k = 4)
```

Finalmente, se muestra la matriz de confusión:

```
> confusionMatrix(knn_all,testing_data$type1 )
Confusion Matrix and Statistics

          Reference
Prediction bug dragon electric fighting normal
  bug        2      0        0        0      0
  dragon     2      1        0        0      0
  electric   0      0        3        0      0
  fighting   1      1        0        1      0
  normal     0      0        2        8     14

Overall Statistics

               Accuracy : 0.6
                 95% CI : (0.4211, 0.7613)
    No Information Rate : 0.4
    P-Value [Acc > NIR] : 0.01326

                  Kappa : 0.4103

 Mcnemar's Test P-Value : NA

Statistics by Class:

                     Class: bug Class: dragon Class: electric Class: fighting Class: normal
Sensitivity             0.40000       0.50000         0.60000         0.11111        1.0000
Specificity             1.00000       0.93939         1.00000         0.92308        0.5238
Pos Pred Value          1.00000       0.33333         1.00000         0.33333        0.5833
Neg Pred Value          0.90909       0.96875         0.93750         0.75000        1.0000
Prevalence              0.14286       0.05714         0.14286         0.25714        0.4000
Detection Rate          0.05714       0.02857         0.08571         0.02857        0.4000
Detection Prevalence    0.05714       0.08571         0.08571         0.08571        0.6857
Balanced Accuracy       0.70000       0.71970         0.80000         0.51709        0.7619
```

¡Y así, hemos aumentado nuestra precisión al 60%! ¡Eso es bastante bueno para una clasificación de 5 clases! Como podemos ver, incluir más atributos ayuda con la clasificación de nuestros Pokémon (aunque te aconsejo que mires nuestra clasificación de Pokémon de lucha: nuestro modelo clasificó 8 de 9 Pokémon de lucha como de tipo normal, ¡eso es un problema!).

Idealmente, nos gustaría que nuestro modelo tuviera una precisión del 100 % en nuestro conjunto de pruebas; sin embargo, esto rara vez (¿o nunca?) es el caso. Ahora, con nuestro modelo de entrenamiento, podemos salir a la naturaleza y clasificar un nuevo tipo de Pokémon en función de sus atributos (sabiendo que hemos logrado una precisión del 60 % en los datos de prueba). Bastante genial, ¿verdad?

Conclusión

En este documento, analizamos el algoritmo K-NN utilizando el conjunto de datos de Pokémon. Cubrimos lo siguiente en este documento:

1. Que pokemon son
2. Qué es KNN

3. Cómo funciona el algoritmo K-NN
4. Cómo determinar atributos (o características) para medir K-NN
5. Dividir los datos en: conjuntos de entrenamiento, prueba y validación
6. Cómo elegir la mejor K

Hay mucha más información para desglosar en este ejemplo presentado (otras métricas de "éxito" como: precisión, recuperación, valor predictivo positivo, etc., tamaño de muestra, filtrado de datos, uso de más etiquetas de clase, etc.) sin embargo, esto sólo es una introducción suave al algoritmo K-NN.