Classification Models For Legendary Pokemon

Alex Ryan

May 9, 2018

Introduction

This is a project that focuses on classifying Non-Legendary Pokemon from Legendary Pokemon. I will explore the data with visualizations and explanations. I will also utilize knn, naive bayes, and xgbtree. These models will attempt to distinguish between these two types of pokemon.

Data Dictionary

The Pokemon dataset was taken from Kaggle. The pokemon included come from Generations 1-6. The dataset includes attributes such as id number, name, Type.1, Type.2, total, HP, Attack, Defense, Sp.Atk, Sp.Def, Speed, Generation, and Legendary. Added features will be discussed in later sections.

Variable Name:	Variable Description:
'id'	Identification number of pokemon
'name'	Name of Pokemon
'Type.1'	Primary property trait of pokemon
'Type.2'	Secondary property trait of pokemon
'total'	Total stats of pokemon
'HP'	Total Health of pokemon
'Attack'	Total Attack of pokemon
'Defense'	Total Defense of pokemon
'Sp.Atk'	Total Special Attack of pokemon
'Sp.Def'	Total Special Defense of pokemon
'Speed'	Total Speed of pokemon
'Generation'	The generation number of pokemon
'Legendary'	A true or false value that defines whether the pokemon is a legendary or not

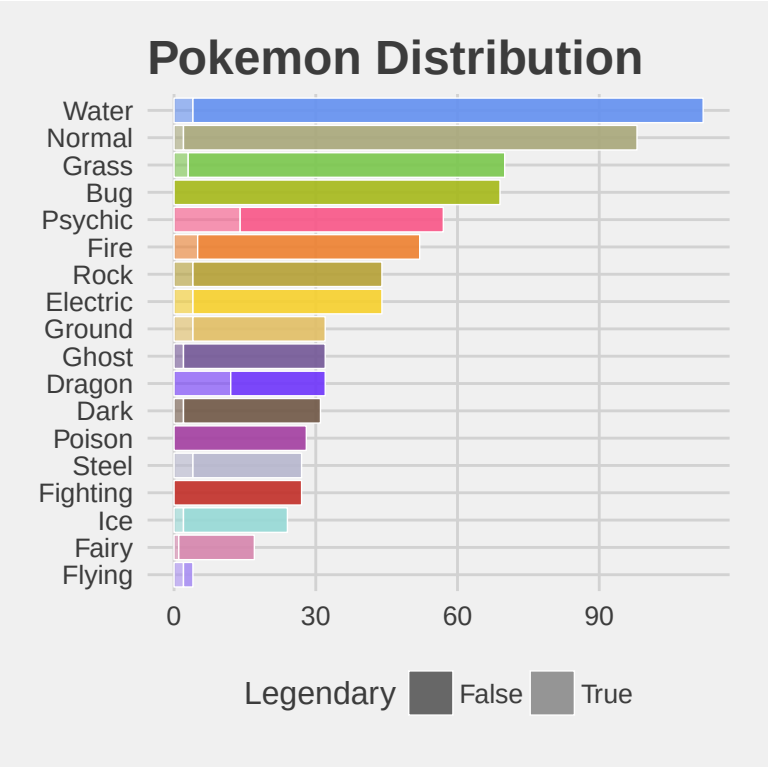
Data Loading and Packages Used

So in order to create to the visualizations and use knn, naive bayes, and xgbtrees, I loaded in all of the packages below as well as the ggplot2 and the xgboost package. I also loaded in my data.

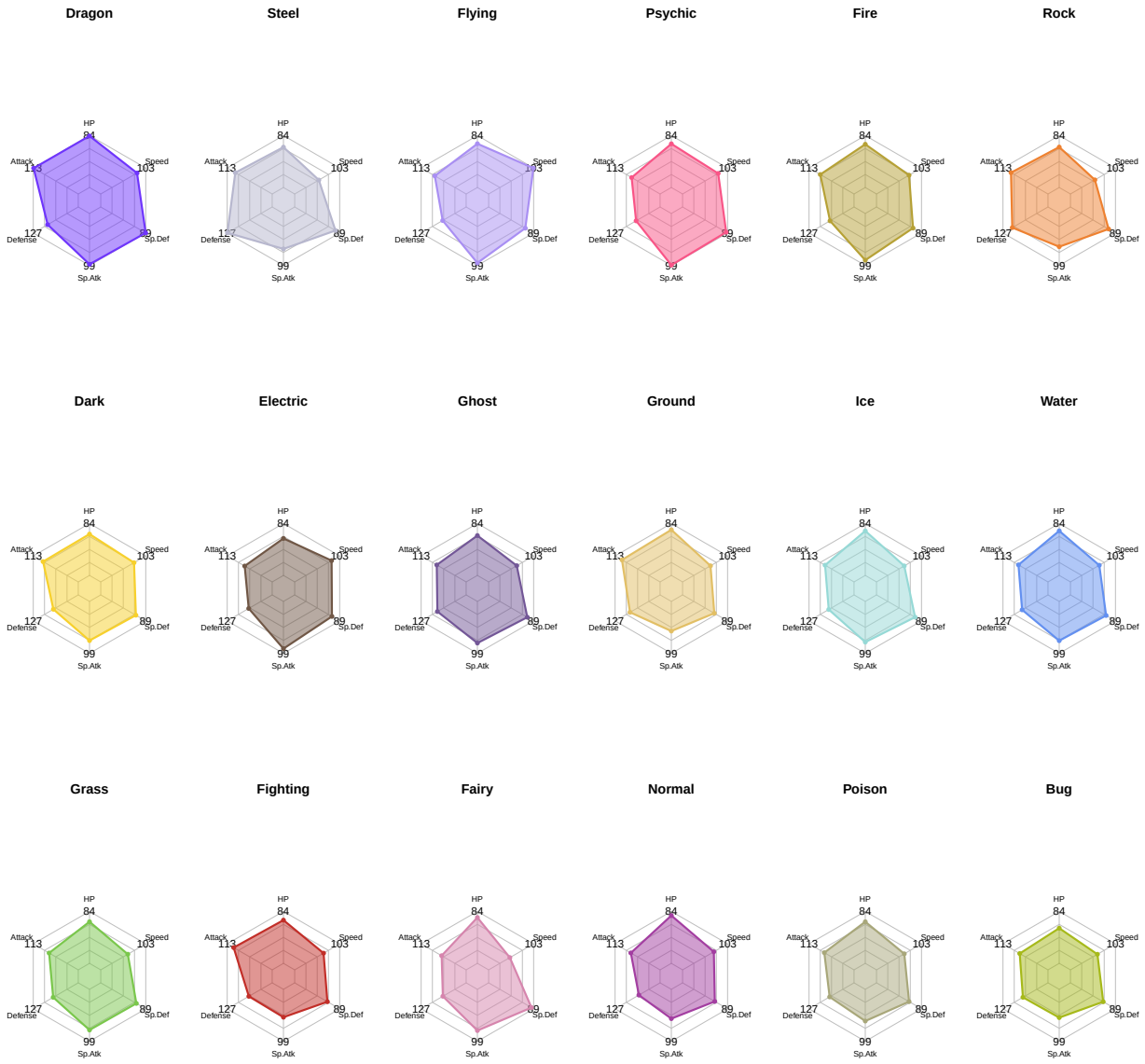
```
library(caret)
library(doSNOW)
library(magrittr)
library(dplyr)
library(ggthemes)
library(fmsb)

pokemon = read.csv("Pokemon.csv", stringsAsFactors = FALSE)
```

Exploratory Analysis with Visualizations



This graph displays the number of pokemon for each primary type and proportion of legendaries in each type. The Psychic, Dragon, and Flying Types have a higher proportion of Legendaries than the other types. So I decided to create three features that target those three types of pokemon. There is also another feature I created based on the graph below.



In this graph, we can see the average base stat distribution in comparision to the primary type's average base stat distribution. The three categories of Psychic, Dragon, and Flying all have at least average to above average stat distribution for each stat. Because each of those three primary types have a high proportion of legendaries, I decided to create a total stat strength feature to help improve legendary classification.

Features Created

This is the feature code for checking if the pokemon is a dragon, psychic, or flying. These three types all have a high proportion of legendary pokemon.

```
#Features to distinguish pokemon of these three types
pokemon$isDragon <- ifelse(pokemon$Type.1 == "Dragon", "Y", "N")
pokemon$isPsychic <- ifelse(pokemon$Type.1 == "Psychic", "Y", "N")
pokemon$isFlying <- ifelse(pokemon$Type.1 == "Flying", "Y", "N")
```

This is the feature code for checking if pokemon has higher total base stats than the median pokemon. This is because legendaries and the three high proportion types have higher base stats than the average pokemon.

```
#Features to
pokemon$OverallStrength <- ifelse(pokemon$Total > median(pokemon$Total),
                                  "Strong", "Weak")
```

Factors Used for Classification

In order to have our classification recognize our categorical data, we need to create factors for each column that we are using. This includes the features we created above.

```
#Factoring Categorical Data and Features
pokemon$Type.1 <- as.factor(pokemon$Type.1)
pokemon$Type.2 <- as.factor(pokemon$Type.2)
pokemon$Generation <- as.factor(pokemon$Generation)
pokemon$Legendary <- as.factor(pokemon$Legendary)
pokemon$OverallStrength <- as.factor(pokemon$OverallStrength)
pokemon$isDragon <- as.factor(pokemon$isDragon)
pokemon$isPsychic <- as.factor(pokemon$isPsychic)
pokemon$isFlying <- as.factor(pokemon$isFlying)
features <- c("Type.1", "Type.2", "Generation", "Legendary","OverallStrength",
              "isDragon", "isPsychic", "isFlying")
pokemon <- pokemon[, features]
```

Train and Test Partitions

The train and test splits were partitioned in a proportion of 80% of the data going to train and 20% of the data going to test. Both of the splits were roughly in same proportion of legendary and non-legendary like the original dataset.

```
# Creating Train and Test Splits
set.seed(54234)
indexes <- createDataPartition(pokemon$Legendary,
                               times = 1,
                               p = 0.8,
                               list = FALSE)
pokemon.train <- pokemon[indexes,]
pokemon.test <- pokemon[-indexes,]

# Examine the proportions of the Survived class lable across
# the datasets.
prop.table(table(pokemon$Legendary))
```

```
##
##   False    True
## 0.91875 0.08125
```

```
prop.table(table(pokemon.train$Legendary))
```

```
##
##   False    True
## 0.91875 0.08125
```

```r
prop.table(table(pokemon.test$Legendary))
```

```
##
##   False    True
## 0.91875 0.08125
```

Cross Validation and Hypertuning For XGBoost

Creating a process for cross validation to occur 10 times and be repeated 3 times, making a grand total of 30 times. I also used a tuning grid for the xgboost in order to optimize the tuning parameters.

```r
# Cross validation and tuning for model
train.control <- trainControl(method = "repeatedcv",
                              number = 10,
                              repeats = 3,
                              search = "grid")

# Tuning grid for xgboost trees
tune.grid <- expand.grid(eta = c( 0.1, 0.2, 0.3, 0.4, 0.5),
                         nrounds = c(50, 75, 100),
                         max_depth = 4:8,
                         min_child_weight = c(2.0, 2.25, 2.5),
                         colsample_bytree = c(0.5, 0.75, 1.0),
                         gamma = 0,
                         subsample = 1)
```

Creating a Cluster to Obtain Optimatal Parameters for XGBoost

I utilized doSNOW to create a cluster with 3 cores. I recommend only running 2 or less cores, if you do not have a workstation quality computer.I then inputted the parameters into the prediction and displayed the results in a confusion matrix, which compares the predictions to the actual results.

```r
#Creating Cluster
cl <- makeCluster(3, type = "SOCK")
registerDoSNOW(cl)

#Gather training model with 3 clusters to identify legendary pokemon
caret.cv <- train(Legendary ~ .,
                  data = pokemon.train,
                  method = "xgbTree",
                  tuneGrid = tune.grid,
                  trControl = train.control)
stopCluster(cl)

#Predict and output results in a confusion matrix
xgbHat = predict(caret.cv, pokemon.test)
confusionMatrix(xgbHat, pokemon.test$Legendary)
```

```
## Confusion Matrix and Statistics
##
##           Reference
## Prediction False True
##      False   137    7
```

```
##      True     10    6
##
##                Accuracy : 0.8938
##                  95% CI : (0.8353, 0.9369)
##     No Information Rate : 0.9188
##     P-Value [Acc > NIR] : 0.8998
##
##                   Kappa : 0.3561
##  Mcnemar's Test P-Value : 0.6276
##
##             Sensitivity : 0.9320
##             Specificity : 0.4615
##          Pos Pred Value : 0.9514
##          Neg Pred Value : 0.3750
##              Prevalence : 0.9187
##          Detection Rate : 0.8562
##    Detection Prevalence : 0.9000
##       Balanced Accuracy : 0.6968
##
##        'Positive' Class : False
##
```