Pokemon Go

Arnab Mukherjee (MB1908)

2 November 2019

In this project we will try to analyse the Pokemon data taken from Kaggle and try to find meaningful insights from the data.

Data Loading & Description:

```
library(readr)

## Warning: package 'readr' was built under R version 3.5.3

Pokemon <- read_csv("C:/Sem-7/Mulivariate/pokemon/Pokemon.csv")

## Parsed with column specification:
## cols(
##   `#` = col_double(),
##   Name = col_character(),
##   `Type 1` = col_character(),
##   `Type 2` = col_character(),
##   Total = col_double(),
##   HP = col_double(),
##   Attack = col_double(),
##   Defense = col_double(),
##   `Sp. Atk` = col_double(),
##   `Sp. Def` = col_double(),
##   Speed = col_double(),
##   Generation = col_double(),
##   Legendary = col_logical()
## )

head(Pokemon)

## # A tibble: 6 x 13
##     `#` Name  `Type 1` `Type 2` Total    HP Attack Defense `Sp. Atk`
##   <dbl> <chr> <chr>    <chr>    <dbl> <dbl>  <dbl>   <dbl>     <dbl>
## 1     1 Bulb~ Grass    Poison     318    45     49      49        65
## 2     2 Ivys~ Grass    Poison     405    60     62      63        80
## 3     3 Venu~ Grass    Poison     525    80     82      83       100
## 4     3 Venu~ Grass    Poison     625    80    100     123       122
## 5     4 Char~ Fire     <NA>       309    39     52      43        60
## 6     5 Char~ Fire     <NA>       405    58     64      58        80
## # ... with 4 more variables: `Sp. Def` <dbl>, Speed <dbl>,
## #   Generation <dbl>, Legendary <lgl>
```

So our variables are , Type 1 and Type 2 {categorical variables denoting categories of the pokemon} Total Denoting the overall rating of the pokemon HP denoting the healing power or in other words how much damage can the pokemon take before fainting Attack, Defense, Sp.Attack ,Sp.Def are self explanatory Speed determines who will make the first shot All 8 above variables are continuous variables Generation is a categorical variable taking values from 1 to 6

Finally our response variable is Legendary again a categorical variable taking values TRUE and FALSE.

Objectives:

1. We will first try to answer the question which qualities makes a pokemon legendary & try to narrow it down to two qualities if possible.
2. Next our goal is to check if our model can predict accurately with reasonably high accuracy.

Challanges & Solutions:

{We are going to point out one problem at a time and solve that before moving into another problem} Major challanges that lies in front of us are the following,

1. Determining how we are going to replace the missing values in the data set {obviously our dataset is already small so we dont want to delete the missing value cells completely}

```
length(Pokemon$`Type 2`)- length(na.omit(Pokemon$`Type 2`))

## [1] 386

#In Percentage
100*((length(Pokemon$`Type 2`)- length(na.omit(Pokemon$`Type
2`)))/(length(Pokemon$`Type 2`)))

## [1] 48.25
```

So there are 386 missing value for Type-2 that amounts to 48.25% , so clearly deleting all the NA values is not an option. So we will replace them by the Type-1 category itself.{As that wont be giving out any wrong information}

```
for( i in 1:length(Pokemon$`Type 2`)){
  if(is.na(Pokemon$`Type 2`[i]) == 'TRUE'){Pokemon$`Type 2`[i] =
Pokemon$`Type 1`[i]}
}
head(Pokemon)

## # A tibble: 6 x 13
##     `#` Name  `Type 1` `Type 2` Total    HP Attack Defense `Sp. Atk`
##   <dbl> <chr> <chr>    <chr>    <dbl> <dbl>  <dbl>   <dbl>     <dbl>
## 1     1 Bulb~ Grass    Poison     318    45     49      49        65
## 2     2 Ivys~ Grass    Poison     405    60     62      63        80
## 3     3 Venu~ Grass    Poison     525    80     82      83       100
## 4     3 Venu~ Grass    Poison     625    80    100     123       122
```

```
## 5     4 Char~ Fire      Fire        309    39      52       43          60
## 6     5 Char~ Fire      Fire        405    58      64       58          80
## # ... with 4 more variables: `Sp. Def` <dbl>, Speed <dbl>,
## #   Generation <dbl>, Legendary <lgl>
```

2. Replacing the categorical variables with numbers and dummies

```
unique(Pokemon$`Type 1`)

##  [1] "Grass"    "Fire"     "Water"    "Bug"      "Normal"   "Poison"
##  [7] "Electric" "Ground"   "Fairy"    "Fighting" "Psychic"  "Rock"
## [13] "Ghost"    "Ice"      "Dragon"   "Dark"     "Steel"    "Flying"

unique(Pokemon$`Type 2`)

##  [1] "Poison"   "Fire"     "Flying"   "Dragon"   "Water"    "Bug"
##  [7] "Normal"   "Electric" "Ground"   "Fairy"    "Grass"    "Fighting"
## [13] "Psychic"  "Steel"    "Ice"      "Rock"     "Dark"     "Ghost"
```

So clearly for the type of pokemon there are total 18 different categories.

Dummy Variable:

A dummy variable is a numerical variable used in regression analysis to represent subgroups of the sample in your study. In research design, a dummy variable is often used to distinguish different treatment groups. In the simplest case, we would use a 0,1 dummy variable where a person is given a value of 0 if they are in the control group or a 1 if they are in the treated group. Dummy variables are useful because they enable us to use a single regression equation to represent multiple groups. This means that we don't need to write out separate equation models for each subgroup. The dummy variables act like 'switches' that turn various parameters on and off in an equation. For more see this: https://en.wikipedia.org/wiki/Dummy_variable_(statistics)

```
library("fastDummies")

## Warning: package 'fastDummies' was built under R version 3.5.3

data = Pokemon[3:13] #because we want to remove the name column it's of no
practical use
dim(data)

## [1] 800  11

data_1= fastDummies::dummy_cols(data) #changing to dummy variables
dim(data)

## [1] 800  11

head(data)

## # A tibble: 6 x 11
##   `Type 1` `Type 2` Total    HP Attack Defense `Sp. Atk` `Sp. Def` Speed
##   <chr>    <chr>    <dbl> <dbl>  <dbl>   <dbl>     <dbl>     <dbl> <dbl>
```

```
## 1 Grass     Poison      318     45      49       49          65          65     45
## 2 Grass     Poison      405     60      62       63          80          80     60
## 3 Grass     Poison      525     80      82       83         100         100     80
## 4 Grass     Poison      625     80     100      123         122         120     80
## 5 Fire      Fire        309     39      52       43          60          50     65
## 6 Fire      Fire        405     58      64       58          80          65     80
## # ... with 2 more variables: Generation <dbl>, Legendary <lgl>
```

So as we can see after using dummy Type-1 and Type-2 pokemons got classified into the dummies.

3. Now our challenge is to fit a regression model for this data

```
#Changing the types to factors
data$`Type 1` = as.factor(data$`Type 1`)
data$`Type 2` = as.factor(data$`Type 2`)

#Breaking the Data into test and train set & Running Logistic Regression
smp_size <- floor(0.85 * nrow(data))

#set.seed(123)
f=function(){
train_ind <- sample(seq_len(nrow(data)), size = smp_size)

train <- data[train_ind, ]
test <- data[-train_ind, ]

for(i in c(4,6))
{
ll=unique(train[,i])
for(j in seq(1,nrow(test),1))
{
if(!(test[j,i] %in% ll))
test[j,i]=0
}
}
logitMod <- glm(train$`Legendary` ~., family=binomial(link="logit"), data =
train)

predictedY <- predict(logitMod, test, type="response")
fitted.results <- ifelse(predictedY > 0.5,1,0)
fitted_train=predict(logitMod, train, type="response")
fitted_train=ifelse(fitted_train > 0.5,1,0)

misClasificError <- mean(fitted.results != test$`Legendary`)
mis_train=mean(fitted_train != train$`Legendary`)
print(paste('Accuracy_Test',1-misClasificError, 'Accuracy_Train',1-
mis_train))
print(summary(logitMod))
}
```

```r
f()
```

```
## Warning: glm.fit: fitted probabilities numerically 0 or 1 occurred

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## Warning in predict.lm(object, newdata, se.fit, scale = 1, type =
## ifelse(type == : prediction from a rank-deficient fit may be misleading

## [1] "Accuracy_Test 0.858333333333333 Accuracy_Train 0.972058823529412"
##
## Call:
## glm(formula = train$Legendary ~ ., family = binomial(link = "logit"),
##     data = train)
##
## Deviance Residuals:
##     Min       1Q   Median       3Q      Max
## -3.2331  -0.0142  -0.0002   0.0000   2.0784
##
## Coefficients: (1 not defined because of singularities)
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -7.262e+01  5.050e+03  -0.014 0.988526
## `Type 1`Dark      2.121e+01  2.361e+03   0.009 0.992830
## `Type 1`Dragon    1.804e+01  2.361e+03   0.008 0.993901
## `Type 1`Electric  2.242e+01  2.361e+03   0.009 0.992422
## `Type 1`Fairy     1.686e+01  2.361e+03   0.007 0.994303
## `Type 1`Fighting -5.890e-01  4.606e+03   0.000 0.999898
## `Type 1`Fire      1.914e+01  2.361e+03   0.008 0.993529
## `Type 1`Flying    3.195e+01  2.371e+03   0.013 0.989248
## `Type 1`Ghost     1.965e+01  2.361e+03   0.008 0.993358
## `Type 1`Grass     2.042e+01  2.361e+03   0.009 0.993100
## `Type 1`Ground    2.115e+01  2.361e+03   0.009 0.992851
## `Type 1`Ice       2.125e+01  2.361e+03   0.009 0.992816
## `Type 1`Normal    1.499e+01  2.361e+03   0.006 0.994934
## `Type 1`Poison    7.180e+00  4.824e+03   0.001 0.998812
## `Type 1`Psychic   2.015e+01  2.361e+03   0.009 0.993190
## `Type 1`Rock      2.023e+01  2.361e+03   0.009 0.993163
## `Type 1`Steel     1.946e+01  2.361e+03   0.008 0.993422
## `Type 1`Water     1.760e+01  2.361e+03   0.007 0.994052
## `Type 2`Dark      5.889e+00  4.464e+03   0.001 0.998947
## `Type 2`Dragon    5.173e+00  4.464e+03   0.001 0.999075
## `Type 2`Electric  7.136e+00  4.464e+03   0.002 0.998725
## `Type 2`Fairy     6.883e+00  4.464e+03   0.002 0.998770
## `Type 2`Fighting  8.822e+00  4.464e+03   0.002 0.998423
## `Type 2`Fire      1.079e+01  4.464e+03   0.002 0.998071
## `Type 2`Flying    7.894e+00  4.464e+03   0.002 0.998589
## `Type 2`Ghost     8.191e+00  4.464e+03   0.002 0.998536
## `Type 2`Grass     7.630e+00  4.464e+03   0.002 0.998636
```

```
## `Type 2`Ground    6.224e+00  4.464e+03   0.001 0.998888
## `Type 2`Ice       7.838e+00  4.464e+03   0.002 0.998599
## `Type 2`Normal    8.637e+00  4.464e+03   0.002 0.998456
## `Type 2`Poison   -7.514e+00  5.417e+03  -0.001 0.998893
## `Type 2`Psychic   9.662e+00  4.464e+03   0.002 0.998273
## `Type 2`Rock      1.353e+01  4.464e+03   0.003 0.997582
## `Type 2`Steel     1.049e+01  4.464e+03   0.002 0.998124
## `Type 2`Water     8.889e+00  4.464e+03   0.002 0.998411
## Total             9.327e-02  2.298e-02   4.059 4.93e-05 ***
## HP               -1.563e-02  2.217e-02  -0.705 0.480973
## Attack           -4.549e-02  2.316e-02  -1.964 0.049478 *
## Defense          -5.000e-02  2.184e-02  -2.289 0.022068 *
## `Sp. Atk`        -3.631e-02  2.602e-02  -1.395 0.162893
## `Sp. Def`        -7.496e-03  2.258e-02  -0.332 0.739851
## Speed                    NA         NA      NA       NA
## Generation        1.139e+00  3.192e-01   3.567 0.000361 ***
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 391.693  on 679  degrees of freedom
## Residual deviance:  89.075  on 638  degrees of freedom
## AIC: 173.08
##
## Number of Fisher Scoring iterations: 20
```

Our regular GLM is giving error because of rank defficiency so we move to to bayes-GLM

Bayes-GLM

```
library("arm")

## Warning: package 'arm' was built under R version 3.5.3

## Loading required package: MASS

## Warning: package 'MASS' was built under R version 3.5.3

## Loading required package: Matrix

## Loading required package: lme4

## Warning: package 'lme4' was built under R version 3.5.3

##
## arm (Version 1.10-1, built: 2018-4-12)

## Working directory is C:/Sem-7/Mulivariate

smp_size <- floor(0.85 * nrow(data))

#set.seed(123)
```

```r
g=function(){
train_ind <- sample(seq_len(nrow(data)), size = smp_size)

train <- data[train_ind, ]
test <- data[-train_ind, ]

for(i in c(4,6))
{
ll=unique(train[,i])
for(j in seq(1,nrow(test),1))
{
if(!(test[j,i] %in% ll))
test[j,i]=0
}
}
logitMod <- bayesglm(train$`Legendary` ~., family=binomial(link="logit"),
data = train)

predictedY <- predict(logitMod, test, type="response")
fitted.results <- ifelse(predictedY > 0.5,1,0)
fitted_train=predict(logitMod, train, type="response")
fitted_train=ifelse(fitted_train > 0.5,1,0)

misClasificError <- mean(fitted.results != test$`Legendary`)
mis_train=mean(fitted_train != train$`Legendary`)
print(paste('Accuracy_Test',1-misClasificError, 'Accuracy_Train',1-
mis_train))
print(summary(logitMod))
}

g()

## [1] "Accuracy_Test 0.933333333333333 Accuracy_Train 0.972058823529412"
##
## Call:
## bayesglm(formula = train$Legendary ~ ., family = binomial(link = "logit"),
##     data = train)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.93223  -0.08993  -0.01013  -0.00131   2.06088
##
## Coefficients:
##                    Estimate Std. Error z value Pr(>|z|)
## (Intercept)      -2.963e+01  3.976e+00  -7.454 9.08e-14 ***
## `Type 1`Dark     -9.959e-01  1.551e+00  -0.642  0.52071
## `Type 1`Dragon    3.375e-02  9.013e-01   0.037  0.97013
## `Type 1`Electric  8.216e-01  1.092e+00   0.752  0.45202
## `Type 1`Fairy    -6.272e-01  1.541e+00  -0.407  0.68399
## `Type 1`Fighting -1.180e+00  1.719e+00  -0.687  0.49221
```

```
## `Type 1`Fire      7.624e-01  8.633e-01   0.883  0.37721
## `Type 1`Flying    1.263e+00  1.312e+00   0.963  0.33540
## `Type 1`Ghost     8.497e-02  1.362e+00   0.062  0.95027
## `Type 1`Grass     6.648e-01  9.680e-01   0.687  0.49228
## `Type 1`Ground    7.834e-01  1.207e+00   0.649  0.51641
## `Type 1`Ice       1.878e+00  1.398e+00   1.343  0.17924
## `Type 1`Normal   -2.071e+00  1.309e+00  -1.583  0.11346
## `Type 1`Poison   -2.097e-01  2.119e+00  -0.099  0.92117
## `Type 1`Psychic   7.343e-01  8.580e-01   0.856  0.39209
## `Type 1`Rock      1.035e+00  9.600e-01   1.078  0.28118
## `Type 1`Steel     9.722e-01  1.034e+00   0.940  0.34722
## `Type 1`Water    -4.862e-01  1.010e+00  -0.481  0.63034
## `Type 2`Dark     -2.539e+00  1.521e+00  -1.669  0.09510 .
## `Type 2`Dragon   -9.477e-01  1.090e+00  -0.869  0.38467
## `Type 2`Electric -5.132e-01  1.466e+00  -0.350  0.72632
## `Type 2`Fairy    -5.932e-01  1.189e+00  -0.499  0.61774
## `Type 2`Fighting -2.533e-01  9.091e-01  -0.279  0.78050
## `Type 2`Fire      6.596e-01  1.144e+00   0.577  0.56412
## `Type 2`Flying    9.127e-01  8.202e-01   1.113  0.26585
## `Type 2`Ghost     4.191e-02  1.361e+00   0.031  0.97542
## `Type 2`Grass    -1.909e-01  1.161e+00  -0.164  0.86946
## `Type 2`Ground   -9.639e-02  1.132e+00  -0.085  0.93217
## `Type 2`Ice      -7.871e-01  1.261e+00  -0.624  0.53258
## `Type 2`Normal    7.257e-02  1.337e+00   0.054  0.95671
## `Type 2`Poison   -8.742e-01  1.730e+00  -0.505  0.61326
## `Type 2`Psychic   9.913e-01  8.516e-01   1.164  0.24441
## `Type 2`Rock      2.044e+00  1.473e+00   1.387  0.16533
## `Type 2`Steel     2.736e-01  1.008e+00   0.271  0.78612
## `Type 2`Water     1.344e+00  1.128e+00   1.192  0.23327
## Total             3.410e-02  1.282e-02   2.659  0.00783 **
## HP                1.737e-02  1.689e-02   1.028  0.30374
## Attack            2.273e-04  1.451e-02   0.016  0.98750
## Defense           6.578e-04  1.505e-02   0.044  0.96513
## `Sp. Atk`         3.475e-03  1.484e-02   0.234  0.81488
## `Sp. Def`         1.854e-02  1.614e-02   1.149  0.25069
## Speed             2.137e-02  1.650e-02   1.295  0.19536
## Generation        6.015e-01  1.893e-01   3.178  0.00148 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 386.89  on 679  degrees of freedom
## Residual deviance: 100.26  on 637  degrees of freedom
## AIC: 186.26
##
## Number of Fisher Scoring iterations: 26
```

Support Vector Machines:

a.Linear Kernel

```
library("caTools")

## Warning: package 'caTools' was built under R version 3.5.3

library("e1071")

## Warning: package 'e1071' was built under R version 3.5.3

smp_size <- floor(0.85 * nrow(data))

#set.seed(123)
g=function(){
train_ind <- sample(seq_len(nrow(data)), size = smp_size)

train <- data[train_ind, ]
test <- data[-train_ind, ]

for(i in c(4,6))
{
ll=unique(train[,i])
for(j in seq(1,nrow(test),1))
{
if(!(test[j,i] %in% ll))
test[j,i]=0
}
}
linear_classifier = svm(train$`Legendary` ~., data = train,type = 'C-
classification',
                 kernel = 'linear')

predictedY <- predict(linear_classifier, test, type="response")

fitted_train=predict(linear_classifier, train, type="response")


misClasificError <- mean(predictedY != test$`Legendary`)
mis_train=mean(fitted_train != train$`Legendary`)
print(paste('Accuracy_Test',1-misClasificError, 'Accuracy_Train',1-
mis_train))
print(summary(linear_classifier))



}
g()
```

```
## [1] "Accuracy_Test 0.891666666666667 Accuracy_Train 0.972058823529412"
##
## Call:
## svm(formula = train$Legendary ~ ., data = train, type = "C-
classification",
##     kernel = "linear")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  linear
##        cost:  1
##
## Number of Support Vectors:  72
##
##  ( 42 30 )
##
##
## Number of Classes:  2
##
## Levels:
##  FALSE TRUE
```

b. Polynomial Kernel:

```
library("caTools")
library("e1071")
smp_size <- floor(0.85 * nrow(data))

#set.seed(123)
g=function(){
train_ind <- sample(seq_len(nrow(data)), size = smp_size)

train <- data[train_ind, ]
test <- data[-train_ind, ]

for(i in c(4,6))
{
ll=unique(train[,i])
for(j in seq(1,nrow(test),1))
{
if(!(test[j,i] %in% ll))
test[j,i]=0
}
}
poly_classifier = svm(train$`Legendary` ~., data = train,type = 'C-
classification',
                 kernel = 'polynomial')

predictedY <- predict(poly_classifier, test, type="response")
```

```r
fitted_train=predict(poly_classifier, train, type="response")


misClasificError <- mean(predictedY != test$`Legendary`)
mis_train=mean(fitted_train != train$`Legendary`)
print(paste('Accuracy_Test',1-misClasificError, 'Accuracy_Train',1-
mis_train))
print(summary(poly_classifier))



}
g()
```

```
## [1] "Accuracy_Test 0.941666666666667 Accuracy_Train 0.972058823529412"
##
## Call:
## svm(formula = train$Legendary ~ ., data = train, type = "C-
classification",
##     kernel = "polynomial")
##
##
## Parameters:
##    SVM-Type:  C-classification
##  SVM-Kernel:  polynomial
##        cost:  1
##      degree:  3
##      coef.0:  0
##
## Number of Support Vectors:  164
##
##  ( 114 50 )
##
##
## Number of Classes:  2
##
## Levels:
##  FALSE TRUE
```

Final Analysis & Remarks:

So first analyze what we did in this project,

1. We selected the pokemon data
2. Modified the data (i.e. imputed missing values and taken care of categorical variables with dummies to ease our computation)
3. Executed GLM and Bayes-GLM
4. Then we went on to use two different SVM classifier methods (both of those were mentioned in class)

Now let's see the results we got,

Name of the method	Training Accuracy	Testing Accuracy
GLM	97.2	85.833
Bayes-GLM	97.2	93.33
SVM-linear	97.2	89.16
SVM-polynomial	97.2	94.166

As it is already clear from the warning we got while executing the analysis this data set is rank deficient so basic GLM model won't do much good in terms of interpretability, so we fall back to the Bayes-GLM model and we can see that **total and the generation** are the most important features in determining if the pokemon is legendary

And in case of prediction as we can see SVM with polynomial kernel out-performs every other method. Although it should be noted that all of the models gives very high accuracy.