Data Mining & Statistical Learning

— Homework 6 —

Pablo González

Abstract

The following report aims to deepen our understanding of some well known statistical learning methods. For this purpose, different supervised algorithms such as ensemble learners were implemented and, to assess them quantitatively, we measured their performance when classifying whether a Pokémon is legendary.

Introduction.

Pokémon, also known as Pocket Monsters in Japan, is a Japanese media franchise founded by Nintendo, Game Freak, and Creatures. According to them, Pokémon are fictional creatures of all shapes and sizes who live in the wild or alongside humans. There are currently more than 800 creatures, with different characteristics, that inhabit the Pokémon universe and some of them are classified as legendary.

On the other hand, Datamining & Statistical learning refers to a framework of statistical models used by data scientists for both inference and prediction, thus including also the ability of predicting the class of a given data point. For example, whether a Pokémon is legendary or not.

Problem Statement and Data Sources

For this assignment, we will be working with a version of the Pokémon data set. This data set has been filtered in order not to contain linear combinations, proxies of our target variable, not-randomly missing values or columns containing either names or identification numbers. Our processed data is composed of 801 observations and 29 variables coming from different pokémon generations across years. For each row, the first binary variable contains whether the pokékon in question is legendary.

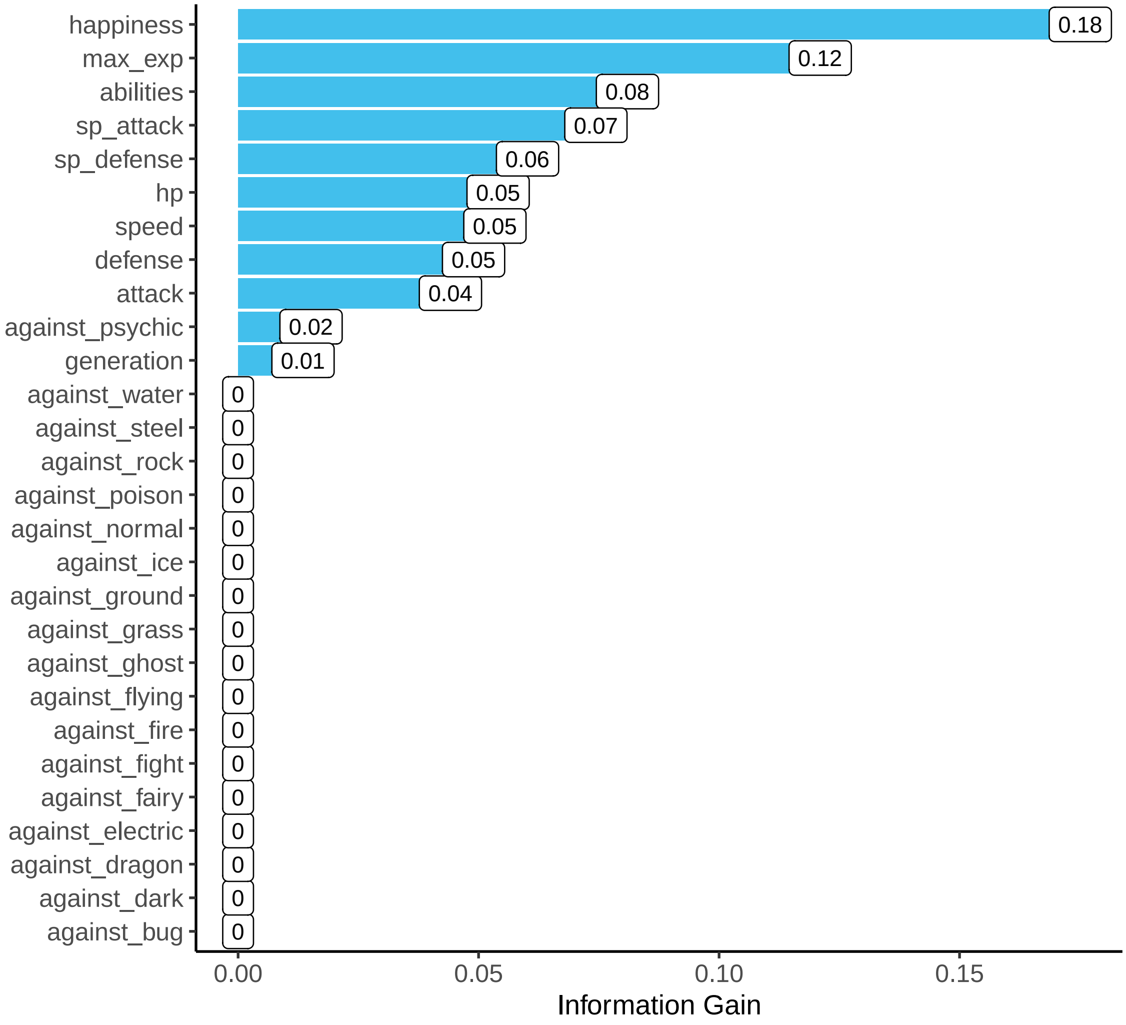
The objective of our analysis is to predict whether a given Pokémon is or not legendary based on 28 attributes such as their health points, attack, defense or speed. In order to achieve our goal, we determine which variables are more likely useful for this classification problem and quantitatively compare the performance of 6 different statistical methods in terms of both error rate (1-accuracy) and area under the ROC curve (AUC). Since our aim is of predictive nature, our decisions regarding variable selection will be taken accordingly. Additionally, no coefficients will be presented in this report, as a significant amount of the employed models do not allow this type of interpretation.

Exploratory Data Analysis

Our data is composed of 801 observations (pokémon) and 29 variables. For each row, the first variable contains a binary variable denoting whether the pokémon is legendary or not and the remaining 28 variables represent certain characteristics that may be insightful to predict this characteristics.

In order to determine which features would be useful for our analysis, as shown in Figure 2, we calculated the information gain of each variable through entropy and determined the best potential predictors for our classification problem.

Figure 1: Predictors by information gain

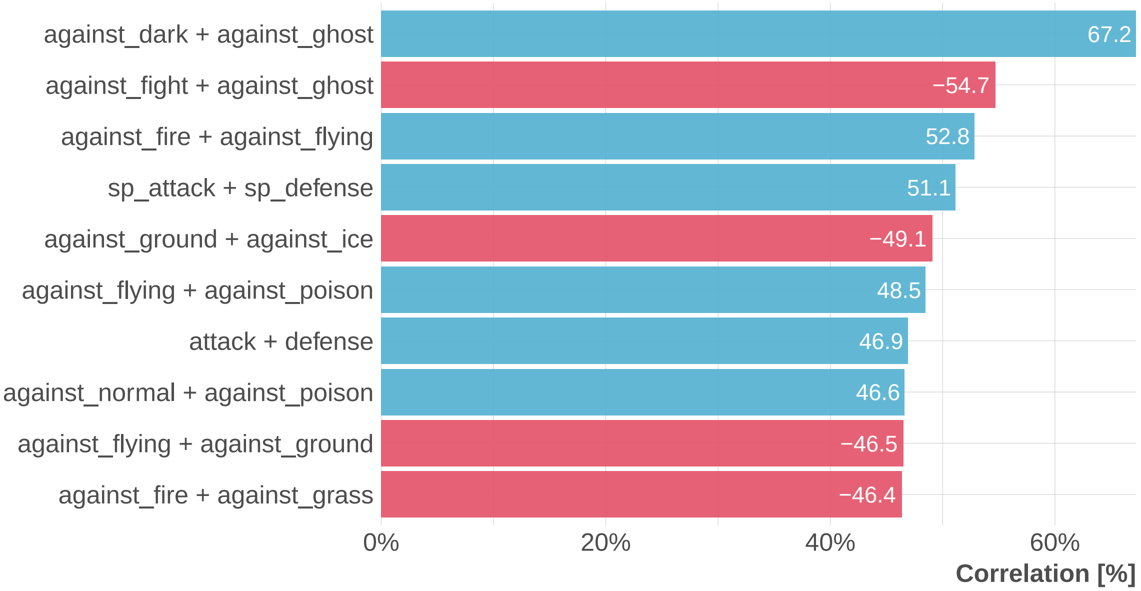


As Figure 2 shows, *generation*, as well as all *against_something* variablese have almost no predictive power, whilst *base_egg_steps* can be considered a strong predictor. Additionally, Figure 3 shows the correlation between the 10 most correlated numeric variables.

Figure 3

Ranked Cross−Correlations

10 most relevant



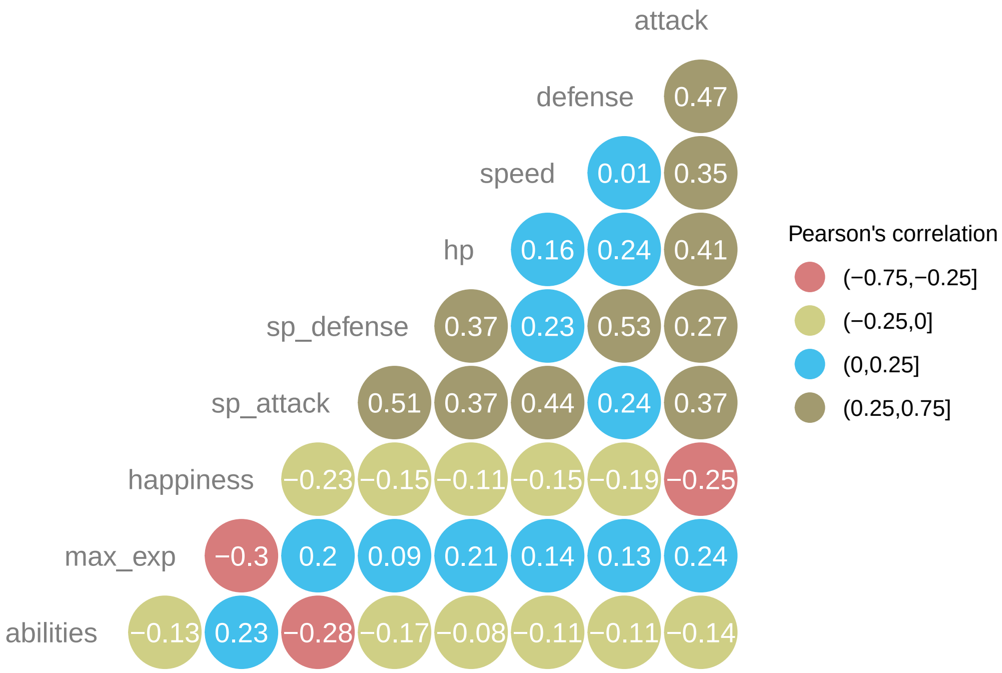
Correlations with p−value < 0.05

Analyzing Figure 2 and 3, and taking our previous conclusions regarding these variables, we can notice that both *generation* and all special resistances presents a weak predictive power and some also present a high correlation with each other. Thus, since removing it would not have a negative effect in our prediction, nor generate omitted variable bias, we decided not to consider them as a predictor.

In fact, whilst it is important to acknowledge that variables such as *sp_attack* and *sp_defense* have strong correlations between each other, thus violating the non-collinearity assumption of the logistic regression, as we know, multicollinearity does not affect the accuracy of predictive models and, as previously stated, removing an important predictor can generate omitted variable bias, which does affect our predictive goal.

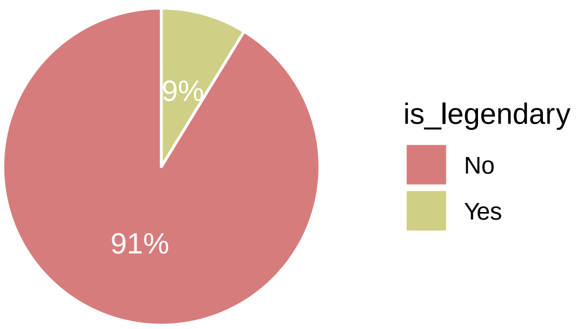
Figure 4: Predictors

Correlation Matrix



After having determined our predictors, we would like to further analyze both our dependent and independent variables. To illustrate, Figure 4 shows the distribution of both of our categorical variables. It is important to note that the variable *is_legendary* is not equally distributed.

Distribution of legendaries



Additionally, Table 1 shows some basic summary statistics for our numeric predictors.

Table 1: Summary statistics

	Min	**Mean**	**Median**	**Max**	**Var**	**Kurtosis**
abilities	1	2.479	3.000	6	0.735	3.013
max-exp	0	43.750	38.462	100	237.444	2.813
happiness	0	65.362	70.000	140	384.119	5.870
sp-attack	10	71.306	65.000	194	1046.770	0.394
sp-defense	20	70.911	66.000	230	780.783	1.498
hp	1	68.959	65.000	255	706.285	8.247
speed	5	66.335	65.000	180	835.653	-0.133
defense	5	73.009	70.000	230	946.741	2.546
attack	5	77.858	75.000	185	1034.190	0.056

Regarding the Kurtosis statistic, it is important to note that this value measures the heaviness of the tails of a distribution relative to a normal distribution, with a Kurtosis value of 3. This means that variables with a higher Kurtosis are more prone to presenting outliers and, in contrast, variables with a negative kurtosis tend to have a flatter distribution.

Finally, we decided to split the original data set into disjoint training and testing data sets, so that we can better evaluate and compare different models. By doing so, we will train our models using 90% from our original data, thus allowing for a better model generation, and keep the remaining 10% for testing the accuracy of these models.

Proposed Methodology

As previously stated, in this report we will compare the performance of 6 different statistical methods for our classification problem. Thus, in order to achieve this goal, we run the following models:

1. Linear discriminant analysis (LDA): This supervised method is commonly used for classification problems. Similar to principal components analysis (PCA), LDA is a dimensionality reduction technique, however, unlike PCA, LDA is also a supervised algorithm that allows for pattern classification by constructing a lower dimensional space that maximizes the variance between classes while minimizing the variance within each class.

2. Naive Bayes (NB): This probabilistic classifier assumes independence of the predictors and uses the Bayes rule to compute the conditional posterior probabilities of our response variable. For this analysis we use a gaussian naive bayes classifier, which implies that we also assume a conditional gaussian likelihood for the predictors.

3. Logistic Regression (LR): This classifier is an extension of the linear regression where the target variable is of categorical nature and is therefore predicted using the logit of the odds as the response variable. This method is commonly used for binary classification problems, as it makes no assumptions regarding the distribution of the target classes. However, it can become unstable when there is a clear separation between classes.

4. K-Nearest Neighbors (KNN): This is a supervised method that can be used to solve both regression problems (as in Homework 1) and classification problems such as predicting *is_legendary*. For this classifier, we estimated the best value for its K-parameter and checked our result by computing several values of K. Thus, when presenting our results, we will only report the resulting classifier for a K value of 7, however, Appendix A explains how we estimated the best hyper-parameter for this data set and shows the cross-validated testing error results for several values of K as well.

5. Random Forest (RF): Similar to the previous method, Random Forest can also be used both for regression and classification problems. In this case, this ensemble classifier operates by constructing 300 decision trees and taking the majority vote of these trees for prediction. As we know, ensemble learners generally outperform all their constituents, however, data characteristics can still affect their performance.

6. Boosting (BST): Boosting is also an ensemble learner. This supervised algorithm is characterized by being able to convert several weak learners to a strong one, thus allowing for bias an variance reduction. The optimal number of boosting iterations is estimated through cross-validations.

It is important to remark that since LDA assumes the predictors to come from a multivariate normal distribution, categorical variables cannot be used by this model. However, since all our predictors are numeric, this will not represent a problem in this occasion.

In the following section, we will analyze the results of the above presented models. It is important to note that no information regarding coefficients was given due to the fact that a significant amount of our models do not generate this type of values and also because our aim is of a predictive nature and we have already determined that our independent variables have predictive power over *is_legendary*.

Analysis and Results

As previously stated, in this section we will discuss the performance of our 6 models using the testing error as our evaluation measure. In addition, we will also discuss the performance of the same 6 models using the area under the roc curve (AUC) and compare both results in order to reach a better understanding of the resulting models.

Table 2 reports the testing error for each model. It is important to note that a better performing model should have a smaller testing error.

Table 2: Testing errors

LDA	NB	LR	KNN	RF	BST
0.0125	0.025	0.0125	0.0125	0	0.025

When analyzing these results, we can notice that the best model for this testing data set is given by the Random Forest classifier with a testing error of 0%, followed by the KNN, LDA and Logistic Regression models with a testing error of 1.25%.

Additionally, Table 3 reports the AUC for each model. As we know, this value indicates how well a model is capable of correctly distinguishing between the given classes and, therefore, a better performance is indicated by a larger value. When analyzing these results, we can notice that the Random Forest classifier is once again the best performing algorithm and that the Boosting classifier does perform better with respect to the presented baseline algorithms.

Table 3: AUC

LDA	NB	LR	KNN	RF	BST
0.992	0.992	0.99733	0.99867	1	0.99733

Normally, this analysis should be sufficient if we one had a large data set. However, since our data set is relatively small, we decided to use Monte Carlo cross-validations to further assess the robustness of each method. Table 4 reports the summary statistics for our testing error after performing a hundred repetitions.

Table 4: Summary statistics for the testing error

	Min	**Mean**	**Median**	**Max**	**Var**	**Kurtosis**
LDA	0	0.03225	0.0375	0.0625	0.00032	-0.92635
NB	0	0.04512	0.0375	0.1125	0.00056	-0.49883
LR	0	0.03687	0.0375	0.1000	0.00044	-0.01295
KNN	0	0.02538	0.0250	0.0875	0.00034	0.48739
RF	0	0.02163	0.0250	0.0750	0.00027	0.47586
BST	0	0.02975	0.0250	0.1000	0.00032	1.45768

Analyzing these results, we can notice that our testing errors have now increased but the random forest classifier is still the better performing algorithm in terms of testing error. Additionally, we can also notice that both ensemble learners are the ones with the least amounnt of variance, which can be explained by the fact that they combine multiple models to make a decission.

Additionally, when analyzing the summary statistics for the area under the curve (AUC) given by Table 5, we can notice that the random forest classifier does also significantly outperform every other classifier, as it has a greater minimum and mean value, whilst also possessing a smaller variance.

Table 5: Summary statistics for the AUC

	Min	**Mean**	**Median**	**Max**	**Var**	**Kurtosis**
LDA	0.93506	0.98561	0.98952	1	0.00020	0.95484
NB	0.80000	0.95702	0.96000	1	0.00123	3.38225
LR	0.88158	0.97876	0.98716	1	0.00071	3.60879
KNN	0.85526	0.97478	0.99346	1	0.00160	1.11431
RF	0.97511	0.99504	0.99702	1	0.00003	0.99484
BST	0.96053	0.99321	0.99571	1	0.00007	2.92868

Finally, taking these results into account, in the following section we would like to give a general summary of our work and conclude this report with some general remarks (conclusions) regarding the performance of our 6 different classifiers.

Conclusions

To confirm our previous analysis of difference, and since we do not know whether our data is normally distributed, we conduct both a T-test and a Wilcox-test over the results presented in Table 5. In this case, it is noteworthy that using both tests will allow us to prove that the performance of the KNN method is different to the compared model's performance if at least one of these tests rejects the null hypothesis of them having a similar performance.

Table 6: A testing error comparison of Random Forest vs other methods

	LDA	NB	LR	KNN	BST
T-test	0	0	0	0.020	0
W-test	0	0	0	0.024	0

As Table 6 shows, the results regarding our Random Forest classifier overperforming every other classifier in terms of testing error are statistically significant, since we can reject the null hypothesis of them having a similar testing error, to the 95% level. Furthermore, Table 7 shows that our results regarding the same classifier are also highly significant

Table 7: An AUC comparison of Random Forest vs other methods

	LDA	NB	LR	KNN	BST
T-test	0	0	0	0	0.003
W-test	0	0	0	0	0.005

The previous tests prove that our conclusions regarding the best classifiers in terms of both testing error and AUC were statistically significant. Thus, regarding our recommended model to predict *is_legendary*, we argue that the Random Forest classifier is the best model for our declared objectives.

Finally, regarding future improvements, we realize that our models are far from perfect and much work can still be done, specially in terms of outlier detection and variable selection. Thus, it would be interesting to see future studies in these subjects, as well as studies regarding selection of interaction variables.

References

- This report was done using “The Complete Pokemon Dataset” from Kaggle’s repository. Available from:https://www.kaggle.com/rounakbanik/pokemon.
- Addional information regarding the assumptions for the Naive Bayes estimator can be found at: https://cran.r-project.org/web/packages/e1071/e1071.pdf (page #34)
- Statistics were done using R 4.0.3 (R Core Team, 2020).

Appendix

A. Estimation of the best K value.

With the aim of obtaining a robust estimator for the K hyper-parameter of our KNN classifier, we computed the testing errors for the first 50 values of K, each for the same data subset one hundred times and, for each repetition, considered only the results that were one $\epsilon = 0.001$ times away from its minimum value, obtaining a list with all the different values of K that met such condition during the repetitions. Later, using that list, we computed how many times did each of those values appear as one of the best performing hyper-parameters, left the most common 5% and computed some summary statistics, as shown by Table 8.

Table 8: Best k value estimate

Mode	**Mean**	**Median**	**Variance**
7	7.006	7	2.646

Our criteria is based on the idea that the best performing hyper-paramether would be the mode of our list. However, in order to ascertain these results, we conducted another 300 simulations for the first 10 values of K and focused on their mean testing error, noticing that, in fact, the value of K=7 seems to perform the best on this data set, given the presented statistics.

Table 9: Testing errors for the first 10 values of K

K	**min**	**mean**	**median**	**max**	**var**	**kurtosis**
1	0	0.03625	0.0375	0.0875	0.00034	-0.43442
2	0	0.03525	0.0375	0.0875	0.00038	-0.08189
3	0	0.03062	0.0250	0.0875	0.00029	0.22207
4	0	0.03087	0.0250	0.0750	0.00029	0.14630
5	0	0.03000	0.0250	0.0750	0.00028	-0.31571
6	0	0.02687	0.0250	0.0750	0.00027	-0.02884
7	0	0.02600	0.0250	0.0625	0.00018	-0.55720
8	0	0.02700	0.0250	0.0875	0.00031	0.17870
9	0	0.02650	0.0250	0.1000	0.00032	2.43503
10	0	0.02550	0.0250	0.0750	0.00028	0.42279

B. R Code

```
knitr::opts_chunk$set(echo = FALSE, message = FALSE, warning = FALSE, fig.pos = 'H')
#Necessary libraries.
if (!require('reshape2')) install.packages('reshape2'); library('reshape2')
if (!require('knitr')) install.packages('knitr'); library('knitr')
if (!require('kableExtra')) install.packages('kableExtra'); library('kableExtra')
if (!require('GGally')) install.packages('GGally'); library('GGally')
if (!require('moments')) install.packages('moments'); library('moments')
if (!require('ggpubr')) install.packages('ggpubr'); library('ggpubr')
if (!require('ggstance')) install.packages('ggstance'); library('ggstance')
if (!require('MASS')) install.packages('MASS'); library('MASS')
if (!require('e1071')) install.packages('e1071'); library('e1071')
if (!require('tidyverse')) install.packages('tidyverse'); library('tidyverse')
if (!require('rminer')) install.packages('rminer'); library('rminer')
if (!require('class')) install.packages('class'); library('class')
if (!require('randomForest')) install.packages('randomForest');
if (!require('gbm')) install.packages('gbm'); library('gbm')
if (!require('FSelectorRcpp')) install.packages('FSelectorRcpp')
library('FSelectorRcpp'); library('randomForest')
if (!require('devtools')) install.packages('devtools'); library('devtools')
if (!require('lares')) install_github("laresbernardo/lares"); library('lares')
# Functions

statistics <- function(data, digits = 3, title = "Summary statistics"){
  data <- data %>% select_if(is.numeric)
  n = seq(1,6*length(data),1)
  table <- data %>% setNames(gsub("_","-",names(.))) %>%
    summarise_all(list(~min(., na.rm = T), ~mean(., na.rm = T),
                       ~median(., na.rm = T),~max(., na.rm = T),
                       ~var(., na.rm = T), ~kurtosis(., na.rm = T))) %>%
  pivot_longer(n,names_to = "measure", values_to = "value") %>%
  separate(measure, c("var", "statistic"), sep = "\\_") %>%
  pivot_wider(names_from = var, values_from = value) %>%
  column_to_rownames('statistic') %>%    t(.) %>% data.frame() %>%
  kable(caption = title, digits = digits, align = "l",
        col.names = c("Min", "Mean", "Median", "Max", "Var", "Kurtosis") ) %>%
    kable_styling("striped", full_width = T,
      latex_options = c("HOLD_position")) %>% row_spec (row = 0, bold = T)
  return(table)
}

shuffle <- function(data){

  row.names(data) <- NULL
  n = dim(data)[1]
  n1 = round(n/10)
  flag <- sort(sample(1:n, n1))
  x_train <- data[-flag,-1]
  y_train <- data[-flag,1]
  x_test <- data[flag,-1]
  y_test <- data[flag,1]

  return( list(x_train, y_train, x_test, y_test) )
}

pie <- function(df, title, mycolors) {
  clase <- eval(parse(text = paste("df$",colnames(df)[1],sep = "")))
  circulo <- df %>% ggplot(aes(x = "", y = frec, fill = as.character(clase))) +
  geom_bar(width = 1, stat = "identity", color = "white", alpha = 0.8) +
  scale_fill_manual(values = mycolors) +
  coord_polar("y", start=0)


  design <- circulo + geom_text(aes(label = paste0(round(frec*100), "%")),
            position = position_stack(vjust = 0.5), color = "#ffffff") +
  labs(x = NULL, y = NULL, title = title, fill = colnames(df)[1]) +
  theme_classic() + theme(axis.line = element_blank(),
          axis.text = element_blank(),
          axis.ticks = element_blank(),
          plot.title = element_text(hjust = 0.5))

return(design)
}

get_errors <- function(data){

  temp <- shuffle(data)
  x_train = temp[[1]]; y_train = temp[[2]];
  x_test = temp[[3]]; y_test = temp[[4]];

  # Naive Bayes
  mod1 <- naiveBayes( x_train, y_train)
  pred1 <- predict(mod1, x_test)
  pred.prob1 <- predict(mod1,x_test,type="raw")
  te1 <- mean( pred1 != y_test)
  roc1 <- mmetric(y_test, pred.prob1, "AUC")

  # LDA
  mod2 <- lda(y_train ~., data = x_train)
  pred2 <- predict(mod2,x_test)$class
  te2 <- mean(pred2 != y_test)
  pred.prob2=predict(mod2,x_test, type = 'response')$posterior
  roc2 <- mmetric(y_test, pred.prob2, "AUC")


  # Logistic
  mod3 <- glm(y_train ~., data = x_train, family = binomial)
  pred3 <- predict(mod3, x_test, type = "response")
  te3 <- mean(round(pred3,0) != y_test)
  pred.prob3 <- data.frame(a = 1-pred3, b = pred3)
  colnames(pred.prob3) <- c(0,1)
```

```r
roc3 <- mmetric(y_test, pred.prob3, "AUC")
#Alternatively: Better for this case, but not learned in this course
#mod3 <- nnet::multinom(y_train ~ ., data=x_train, maxit=400, MaxNWts = 4000)
#pred3 <- predict(mod3, x_test)

# KNN
pred4 <- knn(x_train %>% scale(), x_test %>% scale(), y_train, k=7)
te4 <- mean(pred4 != y_test)
temp4 <- knn(x_train %>% scale(), x_test %>% scale(), y_train, k=7, prob = T)
a = c()
for (i in 1:length(temp4)){
  prob = attr(temp4,'prob')[i]
  prob = ifelse(temp4[i] == 1, prob, 1-prob)
  a = c(a, prob)
}
te4 <- mean(pred4 != y_test)
pred.prob4 = cbind( 1-a,a)
roc4 <- mmetric(y_test, pred.prob4, "AUC")

# Random Forest
mod5 <- randomForest(y_train ~ ., data=x_train, ntree=300, boos=T)
pred5 <- predict(mod5, newdata = x_test)
pred.prob5 <- predict(mod5, newdata = x_test, type = "prob")
te5 <- mean(pred5 != y_test)
roc5 <- mmetric(y_test, pred.prob5, "AUC")


# Boosting
numytr <- as.numeric(y_train)-1 # gbm causes a fatal error with factors
mod6 <- gbm(numytr ~ .,data=cbind(numytr,x_train), distribution = 'bernoulli',
            n.trees = 1000, shrinkage = 0.01, cv.folds = 10)
perf_gbm1 = gbm.perf(mod6, method="cv") #640 iterations are optimal
pred6 <- predict(mod6,newdata = x_test, n.trees=perf_gbm1,
                 type="response")
  te6 <- mean(round(pred6,0) !=  y_test)
  pred.prob6 <- data.frame(a = 1-pred6, b = pred6)
  colnames(pred.prob6) <- c(0,1)
  roc6 <- mmetric(y_test, pred.prob6, "AUC")

  #Return training errors
  TE  <- data.frame( t( c(te1,te2,te3,te4,te5, te6) ) )
  ROC <- data.frame( t( c(roc1,roc2,roc3,roc4,roc5, roc6) ))

  return(list(TE,ROC))

}

get_k<- function(data, reps = 100, rate = 0){
  temp = c()
  for (j in 1:reps)
    {
    tempdata <- shuffle(data)
    x_train = tempdata[[1]]; y_train = tempdata[[2]];
    x_test = tempdata[[3]]; y_test = tempdata[[4]]

    knb <- data.frame(k = NA, value = NA)
    for (i in 1:50){
      ypred2.test <- knn(x_train %>% scale(), x_test %>% scale(), y_train, k=i)
      knb <- rbind(knb, c(i, mean(ypred2.test != y_test)) )
      }
      knb <- knb %>% filter(value <= min(knb$value,na.rm = 1)*(1+rate) )

    temp = c(temp, unlist( knb$k))
  }
  n <- ceiling(length( unique(temp) )/20) #(Top 5%)
  temp2 <- data.frame(temp) %>% group_by(temp) %>% count(sort = TRUE) %>%
           head(n) %>% select(temp) %>% as_vector()
  temp <- temp[temp %in% temp2]
  best <- data.frame(
    Mode =unique(temp)[which.max(tabulate(match(temp, unique(temp))))],
    Mean = mean(temp), Median = median(temp), Variance = var(temp)
            )
  return(best)
}

knnerrors <- function(data, k = 1:5, reps = 100, seed = 7406){
  kvalues = c()
  for (i in k){
    set.seed(seed)
    aux = c()
    for (j in 1:reps){
      temp <- shuffle(data)
      x_train = temp[[1]]; y_train = temp[[2]];
      x_test = temp[[3]]; y_test = temp[[4]]

      pred <- knn(x_train %>% scale(), x_test %>% scale(), y_train, k=i)
      aux <- c(aux, mean(pred != y_test))
    }
    row <- data.frame(aux) %>%
      summarise_all(list(~min(., na.rm = T), ~mean(., na.rm = T),
                         ~median(., na.rm = T), ~max(., na.rm = T),
                         ~var(., na.rm = T), ~kurtosis(., na.rm = T)))
    kvalues <- rbind(kvalues, row)
  }
  kvalues = cbind(data.frame(K = k),data.frame(kvalues))
  return( data.frame(kvalues) )
}
#Dataset
pokemon <- read.csv("pokemon.csv") %>% relocate(is_legendary, .before = 1)

coab <- function(vector){return(length(unlist(strsplit(vector,split='[,]'))))}
```

```r
pokemon$abilities <- sapply(pokemon$abilities, coab,USE.NAMES = 0)
pokemon$experience_growth <- 100*rescale01(pokemon$experience_growth)

colSums(is.na(pokemon))


# Pre-processing
# We remove variables that are linear transformations of others, as well as
# variables that contain too many missing observations or not at random.
# Variables that are ids are also removed.
pokemon <- pokemon %>%  select_if(is.numeric) %>%
  select(-pokedex_number, -base_total, -base_egg_steps,
         -height_m, -percentage_male, -weight_kg) %>%
  mutate(is_legendary = as.factor(is_legendary),
         generation = as.factor(generation)) %>%
  rename(max_exp = experience_growth, happiness = base_happiness)

# Graph colors:
my_colors <- c("indianred","#C3C367","#13AFE8",
               "lightgoldenrod4", "mediumslateblue")
dim(pokemon) #The dataset contains 1375 observations with 257 variables.
sum(is.na(pokemon)) #There is no missing data
rownames_to_column(information_gain( is_legendary ~ ., data = pokemon %>%
  mutate_if(is.numeric, scale) )) %>% rename(variable = attributes) %>%
  ggplot(mapping = aes(x = importance, y = fct_reorder(as.factor(variable),
  importance), label = round(importance,2) )) +
  geom_colh(fill = my_colors[3], alpha = 0.8) + geom_label(size=3) +
  theme_classic() +theme(axis.text = element_text(size = 10)) +
  labs(title    = "Figure 1: Predictors by information gain",
  x = "Information Gain", y = "")
corr_cross(pokemon %>% select(-is_legendary, -generation),
  max_pvalue = 0.05, top = 10) %>%
  annotate_figure(top = text_grob("Figure 3", color = "black", face = "bold",
                                    size = 15, vjust = 1,hjust = 4))
```

```
# Predictors selection
pokemon <- pokemon %>% select(is_legendary, abilities, max_exp,
                              happiness, sp_attack, sp_defense, hp, speed,
                              defense, attack)
pokemon %>% select_if(is.numeric) %>%
  ggcorr(label = T, label_color = "#ffffff", label_size = 4, label_round = 2,
         hjust = 0.9, size = 4, color = "grey50", geom = "blank",
         layout.exp = 1, nbreaks = 4) +
  geom_point(size = 13, aes(color = cut(coefficient,
             breaks = c(-0.75,-0.25,0,0.25,0.75) )), alpha = 0.8) +
  geom_text(aes(label=round(coefficient,2) ),hjust=0.5,
            vjust=0.5, color = 'white') +
  scale_color_manual( values = my_colors ) +
  labs(title = "Figure 4: Predictors", subtitle = "Correlation Matrix",
       color = "Pearson's correlation") +
  guides(colour = guide_legend(override.aes = list(size=5))) +
  theme(plot.title = element_text(hjust = 1),
        plot.subtitle = element_text(hjust = 1))
pokemon %>% mutate(is_legendary = ifelse(is_legendary == 1, "Yes","No")) %>%
  group_by(is_legendary) %>% summarise(n = n()) %>%
  mutate(frec = n/sum(n)) %>% pie('Distribution of legendaries', my_colors)

pokemon %>% statistics(digits = 3)
header <- c('LDA', 'NB','LR', 'KNN', 'RF','BST')

# ---- Models without cross validation
set.seed(7406) # Set the seed for randomization
analysis <- get_errors(pokemon)
te <- analysis[[1]]; roc <- analysis[[2]]
te %>% kable(caption = "Testing errors", digits = 5,
             align = "l", col.names = header) %>%
  kable_styling("striped", full_width = T, latex_options = c("hold_position"))
roc %>% kable(caption = "AUC", digits = 5, align = "l", col.names = header) %>%
```

```r
  kable_styling("striped", full_width = T, latex_options = c("hold_position"))
#CV
errors <- NULL; rocs <- NULL
set.seed(7406)
for (b in 1:100)
  {
  stats <- get_errors(pokemon)
  errors <- rbind(errors, stats[[1]])
  rocs <- rbind(rocs, stats[[2]])
  }
rownames(errors) = NULL; colnames(errors) = header
data.frame(errors) %>%
  statistics(title = 'Summary statistics for the testing error',digits = 5)
rownames(rocs) = NULL; colnames(rocs) = header
data.frame(rocs) %>% statistics('Summary statistics for the AUC', digits = 5)
T1=t.test(errors[,5],errors[,6],paired=T)
T2=t.test(errors[,5],errors[,4],paired=T)
T3=t.test(errors[,5],errors[,3],paired=T)
T4=t.test(errors[,5],errors[,2],paired=T)
T5=t.test(errors[,5],errors[,1],paired=T)

W1=wilcox.test(errors[,5],errors[,6],paired=T)
W2=wilcox.test(errors[,5],errors[,4],paired=T)
W3=wilcox.test(errors[,5],errors[,3],paired=T)
W4=wilcox.test(errors[,5],errors[,2],paired=T)
W5=wilcox.test(errors[,5],errors[,1],paired=T)

table <- data.frame(
  rbind( c(T5$p.value,T4$p.value,T3$p.value,T2$p.value,T1$p.value),
        c(W5$p.value,W4$p.value,W3$p.value,W2$p.value,W1$p.value)))
rownames(table) <- c('T-test','W-test')
table %>%
  kable(caption="A testing error comparison of Random Forest vs other methods",
  digits = 3, align = "l", col.names = header[-5]) %>%
  kable_styling("striped", full_width = T, latex_options = c("hold_position"))
T1=t.test(rocs[,5],rocs[,6],paired=T)
T2=t.test(rocs[,5],rocs[,4],paired=T)
T3=t.test(rocs[,5],rocs[,3],paired=T)
T4=t.test(rocs[,5],rocs[,2],paired=T)
T5=t.test(rocs[,5],rocs[,1],paired=T)

W1=wilcox.test(rocs[,5],rocs[,6],paired=T)
W2=wilcox.test(rocs[,5],rocs[,4],paired=T)
W3=wilcox.test(rocs[,5],rocs[,3],paired=T)
W4=wilcox.test(rocs[,5],rocs[,2],paired=T)
W5=wilcox.test(rocs[,5],rocs[,1],paired=T)

table <- data.frame(
  rbind( c(T5$p.value,T4$p.value,T3$p.value,T2$p.value,T1$p.value),
         c(W5$p.value,W4$p.value,W3$p.value,W2$p.value,W1$p.value)))
rownames(table) <- c('T-test','W-test')
table %>%
  kable(caption = "An AUC comparison of Random Forest vs other methods",
        digits = 3, align = "l", col.names = header[-5]) %>%
  kable_styling("striped", full_width = T, latex_options = c("hold_position"))
set.seed(7406) # Set the seed for randomization
best <- get_k(pokemon, reps = 300, rate = 0.001)

best %>% kable(caption = "Best k value estimate", digits = 3, align = "l") %>%
  kable_styling("striped",full_width=T, latex_options = c("HOLD_position")) %>%
  row_spec (row = 0, bold = T)

knnerrors(pokemon, k = 1:10, seed = 6047) %>%
  kable(caption = "Testing errors for the first 10 values of K",
    digits = 5, align = "l") %>% kable_styling("striped", full_width = T,
    latex_options = c("HOLD_position")) %>% row_spec (row = 0, bold = T)
```