Probabilistic Modeling

Pokemon case-study application

July 2022

Abstract. The project aims to implement a Bayesian Network on a Pokemon dataset, trying to understand a pattern between all the features that make a Pokemon strongest. Three different algorithms were implemented to perform this task: Hill-climbing, Incremental association Markov Blanket and Max-Min Hill Climbing. In the last part a final model is chosen and some estimations about conditional probabilities are performed.
Keywords: Bayesian Network, HC, IAMB, MMHC, Pokemon, bnlearn, probabilistic modeling.

Paola Serra ID 960547
Università degli Studi di Milano
Data Science and Economics

Contents

List of Figures

1 Research Definition

The aim of the project is to build a Bayesian Network model able to show which features influence and/or determine the strength of a Pokemon. There is no official ranking about the strenght of a Pokemon, since each Pokemon is specialized in different abilities and the value of the features can increase (eg evolution)/decrease, although the base stats are a good starting point. Base stats refer to speed, hp, attack, defense, special attack and special defense. HP stands for Hit Points and it is a value that determines how much damage a Pokemon can receive. When a Pokemon's HP is completely down to 0, the Pokemon will faint. The stylistic difference between Attack/Defense and Special Attack/Defense is that they represent different forms of combat. The firsts are used for physical moves that either use a Pokemon's body or involve some physical objects. The seconds are usually done from a distance. Besides of this features, the aim is to try to discover if other variables could be considered when choosing to capture a Pokemon (eg. sex, type, abilities ..).

1.1 Dataset description

Data were taken from a Kaggle dataset called The Complete Pokemon Dataset, available for free as zip file [1]. This dataset contains information on all 802 Pokemon from all Seven Generations of Pokemon. In particular it contains 41 variables:

- name: The English name of the Pokemon
- japanese_name: The Original Japanese name of the Pokemon
- pokedex_number: The entry number of the Pokemon in the National Pokedex
- percentage_male: The percentage of the species that are male. Blank if the Pokemon is gender less
- type1: The Primary Type of the Pokemon
- type2: The Secondary Type of the Pokemon
- classification: The Classification of the Pokemon as described by the Sun and Moon Pokedex
- height_m: Height of the Pokemon in meters
- weight_kg: The Weight of the Pokemon in kilograms
- capture_rate: Higher catch rates mean that the Pokemon is easier to catch
- basee_eggs_steps: The number of steps required to hatch an egg of the Pokemon
- abilities: A stringifield list of abilities that the Pokemon is capable of having
- experience_growth: The Experience Growth of the Pokemon
- base_happiness: Base Happiness of the Pokemon
- against_?: Eighteen features that denote the amount of damage taken against an attack of a particular type
- hp: The Base HP of the Pokemon
- attack: The Base Attack of the Pokemon
- defense: The Base Defense of the Pokemon
- sp_attack: The Base Special Attack of the Pokemon

- sp_defense: The Base Special Defense of the Pokemon
- speed: The Base Speed of the Pokemon
- generation: The numbered generation which the Pokemon was first introduced
- is_legendary: Denotes if the Pokemon is legendary.

A summary gives us the corresponding type and some statistics for all the variables. Only six variables (Name, Japanes_name,Type1, Type2, Abilities, classification) are categorical (capture_rate will be converted as numeric), all the others are numerical with Is_Legendary as a flag.

In the figures below some additional info on data are provided.

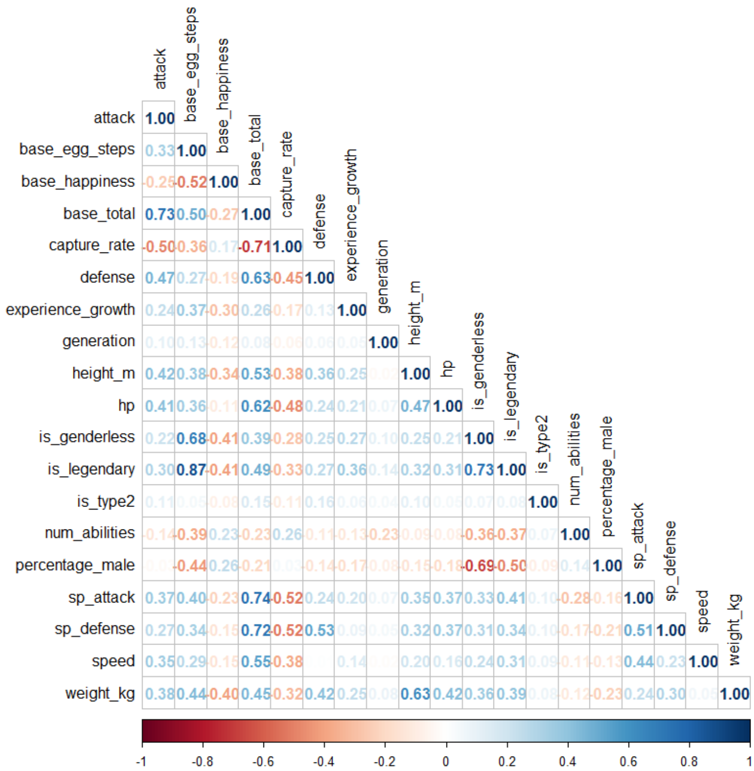

Figure 1: Correlation among variables

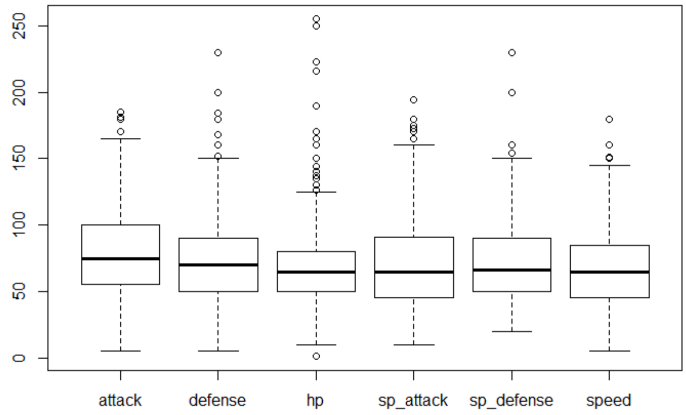

Figure 2: Box plot for Base stats

1.2 Data cleaning and Preprocessing

After an overview of the dataset, variables such as the ”English/Japanese name”, ”pokedex number” and ”classification” are discarded, since they do not add useful information for our macro research (eg. 801 names, 588 cassification’s name and pokedex number is just a sequence). To simplify our model I decide also to discard all the variables against_”something” since the variable ”type1” already encloses this ability (eg. if a pokemon is a type ”ice” it will have a weak resistance on fire and vice versa). Variables ”height_m” and ”weight_kg”contain 20 na’s values each. These values can be replaced by the mean value of all the heights and weights of pokemon for better analysis. We can see that the 98 pokemon which have null values in ”percentage_male” are actually gender less, so I decide to just fill them with a 0 and keep in mind that a very low percentage in this variable could mean both gender less or a majority-female pokemon and create a variable is_genderless to better know which one are for real. Also only some pokemon can have the type2 ability and hence a new variable is_type2 is created. Since the overall abilities were listed in a variable ”abilities” I decide to only consider the amount instead on focusing on their type. In order to learn the structure a discretization is performed in four interval(very low, low, medium, high) for the majority of the numerical variables, while flag variables and percentage_male are divided in two intervals (low and high).

2 Methodology

2.1 Learning procedure

From the base knowledge we know that the sum of speed, hp, attack, defense, special attack and special defense determine the variable "base_total" and also that if a Pokemon is legendary it needs more egg steps. Based on this a white list is built and included in our learning procedure. In order to learn the best Bayesian network a bootstrap technique on different algorithms is applied, considering a threshold as a value that represents the boundary between strong arcs and weak arcs. For each algorithm: data are re sampled by using bootstrap, then for each bootstrap sample a network is learned. It is calculated in terms of frequencies how often each possible arc appears in the network. The arcs that have a frequency greater than a threshold (t = 0.35) are included in the network in output from the specific learning procedure. In particular three algorithms were used: Hill-Climbing, Incremental association Markov Blanket and Max-Min Hill Climbing. Furthermore, for each of the bootstrapped-structure learning procedure applied, it is obtained the adjacent matrix correspondent to the structure of the network found. Then it is calculated the sum of these three matrices in order to obtain a union-matrix,where each row and column represent the variables and the cells represent the number of times an arc connected a specific variable (row) with another variable (column) in the learning procedure applied (for example: the cell corresponding to the intersection of row experience_growth and the column capture_rate contains the value 1. This means that this arc appeared in one of the three learning procedures applied). Below an example of the union matrix, showing only some rows and columns:

	attack	base_egg_steps	base_happiness	base_total	capture_rate	defense	experience_growth
attack	0	0	0	3	0	1	0
base_egg_steps	0	0	3	0	0	0	1
base_happiness	0	0	0	0	0	0	1
base_total	0	0	0	0	1	0	0
capture_rate	0	0	0	0	0	0	0
defense	0	0	0	3	0	0	0
experience_growth	0	0	0	0	1	0	0
height_m	0	0	0	0	0	0	0
hp	1	0	0	3	0	1	0

Figure 3: Union matrix

Before plotting and analyse the structures, some manipulation on the graph learned from the arcs that appear one time is necessary. No undirected graph should be present, so some test are performed in order to choose the right decision. Unfortunately only one of the six arcs has a better direction, the other is forced or dropped in order to create a cycle. All the model are represented in Figure 4:

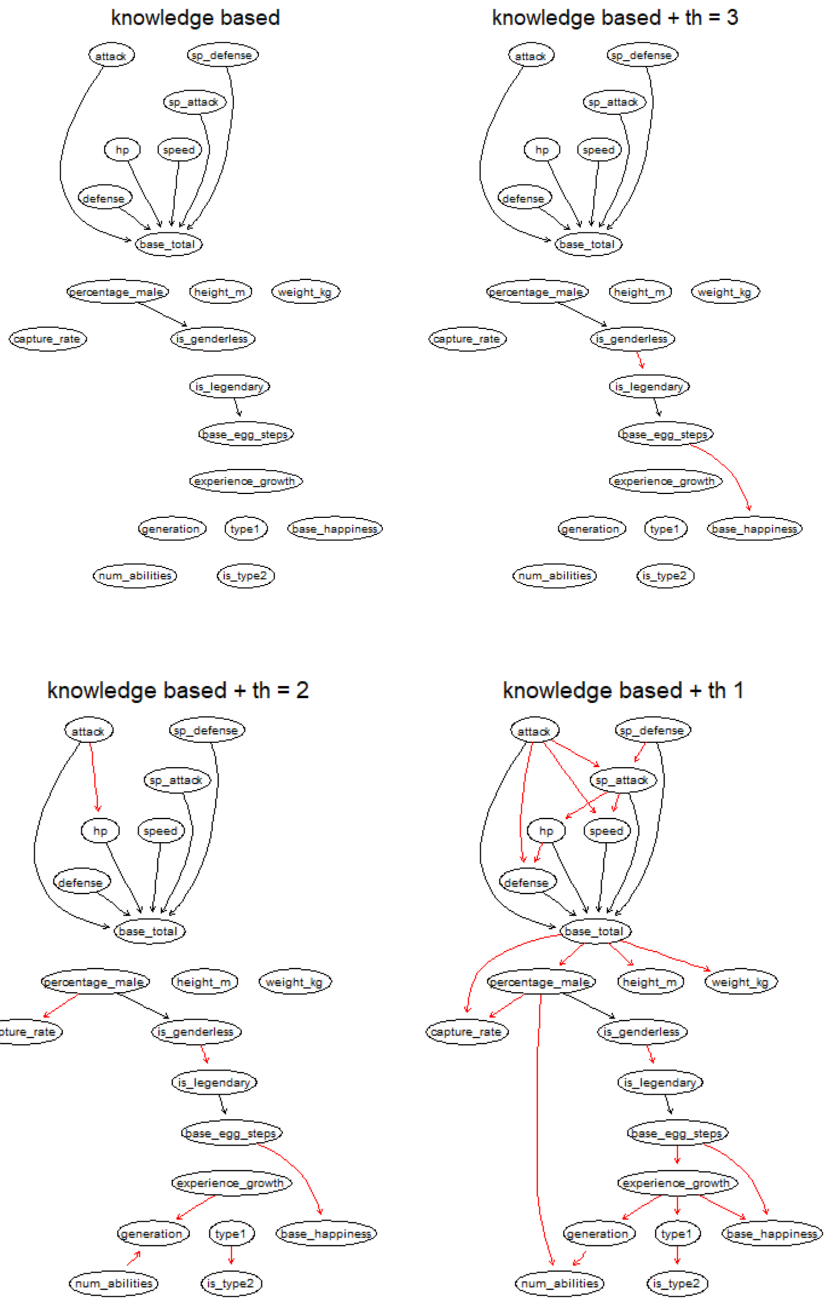

Figure 4: The structure learned for each threshold. Red arrows represent new connection between node with respect to the first model

2.2 Validation

About the goodness of fit of the models, it is possible to measure the predictive accuracy of the learned graphs by applying k-fold cross-validation, observing how the loss-function changes by varying the number of arcs included in the network. This technique consists in partitioning data in k subsets (in this case 10). Each subset is used in turn to validate the model fitted on the remaining k-1 subsets. We can compare the resulting sets of loss values by plotting them as box plots:

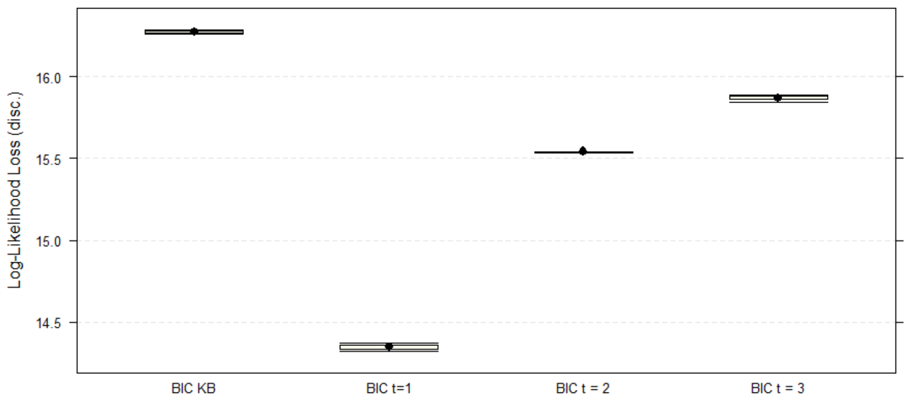

Figure 5: Box plots representing the change in the BIC loss function by varying the number of arcs included in the network.

Results show, as we expected, that by increasing the number of arcs, the accuracy of the model improves. In order to observe a path between features and analyse probabilities conditioned to other variables will be chosen the model with more arcs (threshold = 1), even if the connections could not be so strong as in the other model. Variables that has dependencies with ”base_total” are the one stated in the prior knowledge, although also we discovered that among them they are interconnected (recall that I dropped some arcs in order to not create cycles). Also, a path has found between features that lead to a Pokemon Type. It is possible to affirm that type1 $\perp\!\!\!\perp base\_total|(experience\_growth, base\_egg\_stepps, is\_legendary, is\_genderless,$ percentage_male).

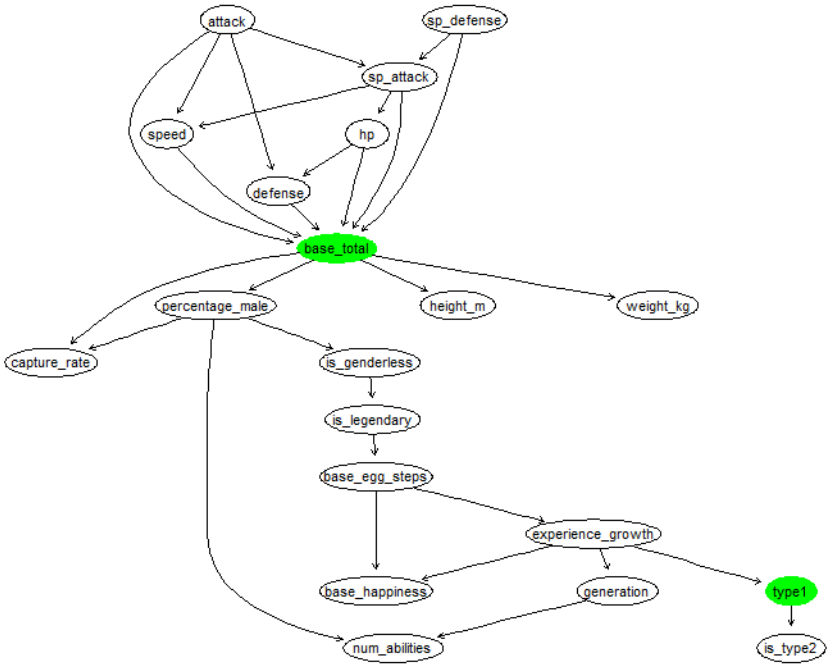

Figure 6: Bayesian Network Structure of the Final Model

2.3 Inference

Once a final model is obtained, some queries are performed on the dataset.
First let's see the probability to have a type2 Pokemon given the species of type1 (P(is_type2 = 1 | type1). Species such as Bug, rock, ground, flying, steel and dark have more probabilities to be a type2, meanwhile fairy, fighting and physic have more probabilities to not be a type2. Secondly, studying the capture rate it is confirmed that the strongest a pokemon is (based on base_total) the more difficult to hatch is. Also results show that a male Pokemon has less probabilities to be captured with respect to a female one. Thirdly, a Pokemon that belongs to the previous generation has slightly more probabilities to have high base stats. Last but non least, we tried to put together most of the variables and find the conditional probabilities. Pokemon with grass, water, steel, rock, dragon as type seem to have more probabilities to have highest score in the features.

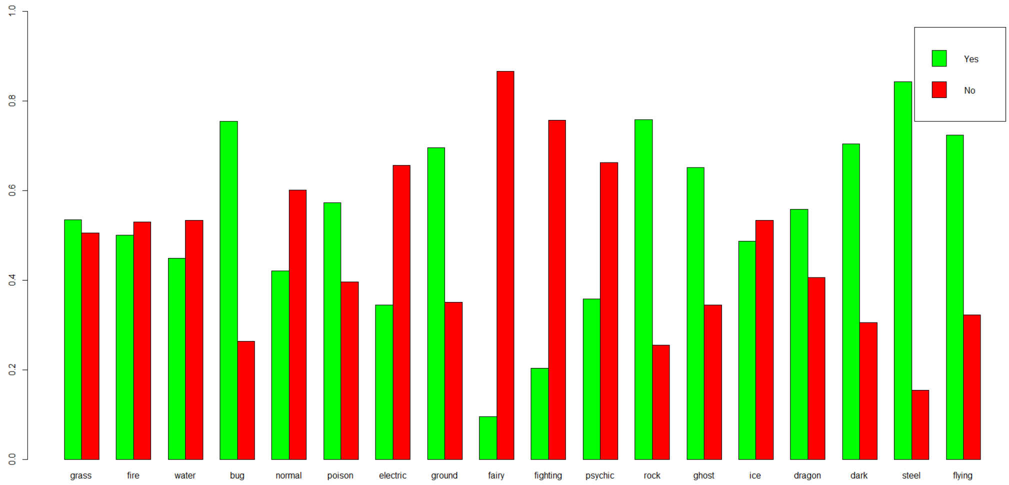

Figure 7: Probability to have a type2 given type1

3 Concluding remarks

After some pre-processing, Hill-Climbing with AIC score was selected as the best model for the pokemon dataset. As confirmed by the literature [3], the main base stats (hp, speed, attack, defense, special attack and defense) are the starting point of our Bayesian Network. Looking at them it is possibile to have an overview of the stamina of a pokemon, but we saw that other variable, such as the gender, the fact a Pokemon is legendary, the type, the capture rate and the experience_growth are also variable that need to be considered. Each Pokemon is different and has his/her main feature, strenghts and weaknesses, so could be that with some Pokemon he/she is very strong while with other will faint at the beginning of a battle. The type1 that has been founded more recurring in the last queries (grass, water, steel, rock, dragon) are also confirmed by the online network, even though no official rank exists. It is possibile also to choose a pokemon type based on the variables one is interested to consider (eg. I want a Pokemon type that is strong in one selected ability). The model could be improved by considering more features and/or trying to build a Markov network.
The code of the project is available on Github.

References

[1] https://www.kaggle.com/datasets/rounakbanik/pokemon

[2] https://bulbapedia.bulbagarden.net/wiki/Main_Page

[3] https://www.bnlearn.com

4 Appendix: R code

```
[language=R]
library(purrr)
library(bnlearn)
library(igraph)
library(corrplot)
library(gRain)
library(graph)
library(catnet)
library(gRbase)
library(Rgraphviz)


pokemon=read.csv("C:\\Users\\Paola\\Desktop\\PROBABILISTIC MODELLING\\
    PROBABILISTIC MODELLING\\pokemon.csv",
                   header = TRUE,sep = ',')

head(pokemon)

summary(pokemon)

length(unique(pokemon$classfication))
length(unique(pokemon$name))

#PREPROCESSING
sum(is.na(pokemon))
sum(is.na(pokemon$abilities))

sum(is.na(pokemon$name))
#replaced 40 values with respective mean
pokemon$weight_kg[is.na(pokemon$weight_kg)] = mean(pokemon$weight_kg,na.
    rm=TRUE)
pokemon$height_m[is.na(pokemon$height_m)] = mean(pokemon$height_m,na.rm=
    TRUE)

#98 values

pokemon$is_genderless = with(
  pokemon, ifelse(is.na(pokemon$percentage_male), 1, 0))
pokemon$percentage_male[is.na(pokemon$percentage_male)]= 0

#not null but maybe useful for our analysis
pokemon$is_type2 = with(
  pokemon, ifelse(pokemon$type2 == "", 0, 1))
#chnge the capture rate

length(pokemon$weight_kg[which(pokemon$weight_kg > 300)])    #25 pokemon
    over 300kg
pokemon[pokemon$weight_kg[which(pokemon$weight_kg > 700)]]
pokemon$name[pokemon$weight_kg > 700,]
pokemon$name[which(pokemon$weight_kg > 700)]  #[1] "Groudon"    "Giratina
    "  "Mudsdale"   "Cosmoem"    "Celesteela" "Guzzlord"  #confirmed by
    literature
pokemon$capture_rate <- sapply(pokemon$capture_rate,as.numeric)
```

```r
pokemon$capture_rate[is.na(pokemon$capture_rate)] <- 30

#create number of abilities
pokemon$num_abilities = lengths(strsplit(pokemon$abilities, ","))


pokemon$attack <- sapply(pokemon$attack,as.numeric)
pokemon$base_egg_steps <- sapply(pokemon$base_egg_steps,as.numeric)
pokemon$base_happiness <- sapply(pokemon$base_happiness,as.numeric)
pokemon$base_total<- sapply(pokemon$base_total,as.numeric)
pokemon$sp_defense<- sapply(pokemon$sp_defense,as.numeric)
pokemon$experience_growth<- sapply(pokemon$experience_growth,as.numeric)
pokemon$hp<- sapply(pokemon$hp,as.numeric)
pokemon$sp_attack <- sapply(pokemon$sp_attack,as.numeric)
pokemon$defense <- sapply(pokemon$defense,as.numeric)
pokemon$speed <- sapply(pokemon$speed,as.numeric)
pokemon$weight_kg <- sapply(pokemon$weight_kg,as.numeric)
pokemon$generation <- sapply(pokemon$generation,as.numeric)
pokemon$is_legendary <- sapply(pokemon$is_legendary,as.numeric)
pokemon$is_type2 <- sapply(pokemon$is_type2,as.numeric)
pokemon$is_genderless <- sapply(pokemon$is_genderless,as.numeric)
pokemon$num_abilities <- sapply(pokemon$num_abilities,as.numeric)


corrplot(cor(pokemon[c(20:24,26:29,32,34:36,39:44)]), method = 'number', order = 'alphabet',type = "lower",tl.col = "black")
boxplot(pokemon[c(20,26,29,34:36)])

#Discretize the data_________________________
dpokemon2 = bnlearn::discretize(pokemon[c(20:24,26:29,34:36,39,40,44)], method = "interval",
                                #dpokemon = bnlearn::discretize(npokemn, method = "interval",
                                breaks= 4) #


dpokemon2 = bnlearn::discretize(pokemon[c(20:24,26:29,34:36,39,40,44)], method = "interval",breaks= 4) #
for (i in names(dpokemon2))
  levels(dpokemon2[, i]) = c( "very low","low", "medium","high")


dpokemon1 = bnlearn::discretize(pokemon[c(32,41:43)], method = "interval",breaks= 2) #
for (i in names(dpokemon1))
  levels(dpokemon1[, i]) = c( "low","high")

dpokemon=cbind(dpokemon2, dpokemon1)

dpokemon$type1 <-as.factor(pokemon$type1)
levels(dpokemon$percentage_male)
levels(dpokemon$type1)

#LEARNING PHASE___________________________________________________________________________________

```

```r
whitelist=matrix(c(
  "attack","base_total",
  "defense","base_total",
  "sp_attack","base_total",
  "sp_defense","base_total",
  "speed","base_total",
   "hp","base_total",
   "percentage_male","is_genderless",
   "is_legendary","base_egg_steps"
   ),,2,byrow = TRUE)

colnames(whitelist)=c("from","to")
whitelist


#Bootstrap in hc model____________________________________________________________________________
set.seed(12)
str.diff_hc = boot.strength(dpokemon, R = 400, algorithm = "hc",  cpdag = TRUE,
                            algorithm.args = list(score="aic", whitelist = whitelist))

avg.diff_hc = averaged.network(str.diff_hc, threshold = 0.35)
strength.plot(avg.diff_hc, str.diff_hc, shape = "ellipse", highlight = list(arcs = whitelist, lwd = 0.35))

head(str.diff_hc)
hc_mat = amat(avg.diff_hc)
hc_mat

#Bootstrap in IAMB model__________________________________________________________________________
set.seed(12)
str.diff_iamb = boot.strength(dpokemon, R = 400, algorithm = "iamb", cpdag = TRUE,
                            algorithm.args = list(whitelist = whitelist))

avg.diff_iamb = averaged.network(str.diff_iamb, threshold = 0.35)
strength.plot(avg.diff_iamb, str.diff_iamb, shape = "ellipse", highlight = list(arcs = whitelist, lwd = 0.70))

head(str.diff_iamb)
iamb_mat = amat(avg.diff_iamb)
iamb_mat


#Bootstrap in MMHC modelL_________________________________________________________________
set.seed(12)
str.diff_rsmax2 = boot.strength(dpokemon, R = 400, algorithm = "rsmax2",
                                algorithm.args = list(whitelist = whitelist))

avg.diff_rsmax2 = averaged.network(str.diff_rsmax2, threshold = 0.35)
strength.plot(avg.diff_rsmax2, str.diff_rsmax2, shape = "ellipse",
```

```
    highlight = list(arcs = whitelist, lwd = 0.35))
rsmax2_mat = amat(avg.diff_rsmax2)
rsmax2_mat

#Union of the matrices
mat_sum = hc_mat + iamb_mat   + rsmax2_mat
mat_sum


bn_from_mat = function(adj_mat, tshl){
  adj = adj_mat
  adj[which(mat_sum < tshl)] = 0
  adj[which(mat_sum >= tshl)] = 1

  model = empty.graph(colnames(adj))
  amat(model) = adj

  return(model)
}

var = colnames(dpokemon)
e = empty.graph(var)
arcs(e) = whitelist


#model_1: threshold = 1
model_1 = bn_from_mat(mat_sum, 1)


Dmodel_1 = cextend(model_1)
#some unidirected arcs were created so try to test them e choose a
    direction

undirected.arcs(model_1)

# from               to
# [1,] "attack"          "hp"
# [2,] "capture_rate"    "percentage_male"
# [3,] "defense"         "sp_defense"
# [4,] "defense"         "speed"
# [5,] "hp"              "attack"
# [6,] "percentage_male" "capture_rate"
# [7,] "sp_defense"      "defense"
# [8,] "speed"           "defense"
# [9,] "generation"      "num_abilities"
# [10,] "num_abilities"   "generation"
#[11,] "generation"        "num_abilities"
#[12,] "num_abilities"     "generation"

choose.direction(model_1, dpokemon, arc = c("generation", "num_abilities"
    ), debug = TRUE)
choose.direction(model_1, dpokemon, criterion = "aic",arc = c("generation
    ", "num_abilities"), debug = TRUE)
choose.direction(model_1, dpokemon, criterion = "bic",arc = c("percentage
    _male", "capture_rate"), debug = TRUE)
#"sp_defense"        "defense
```

```
choose.direction(model_1, dpokemon, arc = c("base_happiness", "experience_growth"), debug = TRUE)
choose.direction(model_1, dpokemon, criterion = "bde",arc = c("base_happiness", "experience_growth"), debug = TRUE)
choose.direction(UPDATE2, dpokemon,criterion = "aic", arc = c("defense", "sp_defense"), debug = TRUE)


UPDATE = set.arc(model_1,"generation","num_abilities")
UPDATE1 = set.arc(UPDATE,"percentage_male","capture_rate")
UPDATE2 = set.arc(UPDATE1,"percentage_male","type1")
UPDATE3 = drop.arc(UPDATE2,"attack","hp")
UPDATE4 = drop.arc(UPDATE3,"defense","speed")
UPDATE5=drop.arc(UPDATE4,"defense","sp_defense")

undirected.arcs(UPDATE5)
Dmodel_1 = cextend(UPDATE5)


#model_2: threshold = 2
model_2 = bn_from_mat(mat_sum, 2)
model_2 = cextend(model_2)

#model_3: threshold = 3

model_3 = bn_from_mat(mat_sum, 3)
model_3 = cextend(model_3)

par(mfrow = c(1,2))
G_compare <-graphviz.compare(e, model_3, model_2, Dmodel_1, shape = 'ellipse',
                 main = c("knowledge based", "knowledge based + th = 3", "knowledge based + th = 2", "knowledge based + th = 1"))



cv.bic_base = bn.cv(dpokemon, bn = e, runs = 10,
                    algorithm.args = list(score = "bic"))

cv.bic_1 = bn.cv(dpokemon, bn = Dmodel_1, runs = 10,
                 algorithm.args = list(score = "bic"))

cv.bic_2 = bn.cv(dpokemon, bn = model_2, runs = 10,
                 algorithm.args = list(score = "bic"))

cv.bic_3 = bn.cv(dpokemon, bn = model_3, runs = 10,
                 algorithm.args = list(score = "bic"))


plot(cv.bic_base, cv.bic_1, cv.bic_2, cv.bic_3, xlab = c("BIC KB", "BIC t=1", "BIC t = 2", "BIC t = 3"))

losses1=c(mean(loss(cv.bic_base)),mean(loss(cv.bic_1)),mean(loss(cv.bic_2)),mean(loss(cv.bic_3)))
losses=data.frame(losses1)
rownames(losses)=c('base','1','2','3')
```

```
losses

#losses1
# losses1
# base 16.27250
# 1    14.34642
# 2    15.54045
# 3    15.87061

fit = bn.fit(Dmodel_1, dpokemon)
fit

par(mfrow = c(1,1))
gR = graphviz.plot(fit, layout = "dot", shape = "ellipse",
                   highlight = list(nodes = c("type1","base_total"), col
    = c("green"), fill = "green"))

node.attrs = nodeRenderInfo(gR)

#evidence
#The other important limitation of gRain, compared to bnlearn, is that it
     does not allow for conditional probabilities to be NaN.
type1_values = unique(dpokemon$type1)
experience_values = unique(dpokemon$experience_growth)
type2_values = unique(dpokemon$is_type2)
capturerate_values = unique(dpokemon$capture_rate)
generation_values = unique(dpokemon$generation)

query <- cpquery(fit, (percentage_male == 'low'),(base_total == 'high'),
    method = 'ls', debug = TRUE)
query


query2 <- cpquery(fit, (is_type2 == "high"),(type1 == "normal" &
    experience_growth == "very low"),method = 'ls', debug = TRUE)
query2

query3- cpquery(fit, (is_type2 == "high"),(type1 == "normal"))
#query about type

type2_h_G_type1 = c()
for (i in 1:length(type1_values)){
  value = toString(i)
  prob = cpquery(fit, (is_type2 == "high") , type1 == type1_values[i])
  type2_h_G_type1[i] = prob
}


type2_l_G_type1 = c()
for (i in 1:length(type1_values)){
  value = toString(i)
  prob = cpquery(fit, (is_type2 == "low") , type1 == type1_values[i])
  type2_l_G_type1[i] = prob
}
# > type1_values
# [1] grass    fire     water    bug      normal   poison   electric
```

```
    ground   fairy    fighting psychic  rock
# [13] ghost    ice      dragon   dark     steel    flying
# 18 Levels: bug dark dragon electric fairy fighting fire flying ghost grass ground ice normal poison ... water
# > type2_values
# [1] high low
# Levels: low high
#check
querycheck = cpquery(fit, (is_type2 == "low") , type1 == "bug")
querycheck



df <-cbind.data.frame(type2_h_G_type1,type2_l_G_type1, row.names = as.vector(type1_values))
df
g_range = range(0.0, 1.0)
barplot(t(as.matrix(df)), beside = TRUE, col = c("green", "red"),ylim = g_range,legend.text = c("Yes","No"))


##query 2

#queying about capture_rate

queryCgB=cpquery(fit, (capture_rate == "high") , (base_total== "high" ))

cpquery(fit, (capture_rate == "high") , (base_total== "very low" ))
#it is confirmed that the strongest a pokemon is the more difficult to hatch is
CP_L_G_%male_and_bt = c()
for (i in 1:length(type1_values)){
  value = toString(i)
  prob = cpquery(fit, (capture_rate == "very low") , (percentage_male == "low" ))
  type2_h_G_type1[i] = prob
}

 cpquery(fit, (capture_rate == "very low") , (percentage_male == "high" ))
# 0.8063837 seems that if a pokemon is male it has a low capture rate
 cpquery(fit, (capture_rate == "very low") , (percentage_male == "high" & base_total == "very low"))
 #[1] 0.6046512 drop quite
cpquery(fit, (capture_rate == "high") , (percentage_male == "high" ))
 cpquery(fit, (capture_rate == "very low") , (percentage_male == "low" ))
#[1] 0.4055761  genderless or more female easily to be captured
cpquery(fit, (capture_rate == "very low") , (percentage_male == "low" & base_total == "very low"))
#0.07285145

 cpquery(fit, (capture_rate == "very low") , (percentage_male == "low" & base_total == "high"))
# [1] 1

cpquery(fit, (is_legendary == "high") , (is_genderless == "high"))
```

```
 #[1] 0.6673718 confirmed


cpquery(fit, (base_total == "high") , (hp == "high" & sp_attack == "low" & sp_defense== "low"))
#[1] 0.06451613

##query about generation
generationG_bt = c()
for (i in 1:length(generation_values)){
  value = toString(i)
  prob = cpquery(fit, (generation == generation_values[i]) , (base_total == "high" ))
  generationG_bt[i] = prob
}
generationG_bt
#firsts generation seems to have an higher bt [1] 0.2960000 0.3228070 0.1887550 0.1742424
#Generation VII added the most Pok mon species whose gender is unknown, with a total of 29.

typeG_bt = c()
for (i in 1:length(type1_values)){
  value = toString(i)
  prob = cpquery(fit, (type1 == type1_values[i]) , (base_total== "high" ))
  typeG_bt[i] = prob
}
typeG_bt
#grass water  normal

typeG_bt2 = c()
for (i in 1:length(type1_values)){
  value = toString(i)
  prob = cpquery(fit, (type1 == type1_values[i]) , (base_total== "medium"))
  typeG_bt2[i] = prob
}
typeG_bt2
#grass bug normal

typeG_bt3 = c()
for (i in 1:length(type1_values)){
  value = toString(i)
  prob = cpquery(fit, (type1 == type1_values[i]) , (base_total== "low" ))
  typeG_bt2[i] = prob
}
typeG_bt3

query4 = cpquery(fit, (is_type2 == "high") , (base_total== "high" ))
query4
#[1] 0.4882812
#overall queries
typeG_ALL = c()
for (i in 1:length(type1_values)){
 value = toString(i)
```

```
prob = cpquery(fit, (type1 == type1_values[i]) , (experience_growth == 'medium' &
                      base_total== "high" & is_type2 == 'high' ))
typeG_ALL[i] = prob
}
typeG_ALL
#water   ghst rock  steel

typeG_ALL = c()
for (i in 1:length(type1_values)){
     value = toString(i)
    prob = cpquery(fit, (type1 == type1_values[i]) , (
       base_total== "medium"& base_egg_steps== 'medium'))
     typeG_ALL[i] = prob
   }
typeG_ALL

#water normal electric ground dragon ice
typeG_ALL = c()
for (i in 1:length(type1_values)){
    value = toString(i)
    prob = cpquery(fit, (type1 == type1_values[i]) , (experience_growth == 'medium' &
                         base_total== "high"& base_egg_steps== 'high'))
    typeG_ALL[i] = prob
  }
typeG_ALL

#grass steel dragon rock


typeG_ALL = c()
 for (i in 1:length(type1_values)){
    value = toString(i)
     prob = cpquery(fit, (type1 == type1_values[i]) , (experience_growth == 'medium' &
            base_total== "high" & is_legendary == 'high' ))
    typeG_ALL[i] = prob
 }
typeG_ALL

#grass water normal physic steel e dragon
```