M1 Assignment 2

Lars Nielsen

14/9/2019

Contents

Description: This time you will work with Pokemon data. No data munging needed. Just old-school ML. Data You will find the dataset for this assignment under: https://github.com/SDS-AAU/M1-2019/raw/master/data/pokemon.csv

It contains data on 800 Pokemon from the 1st to the 6th generation.

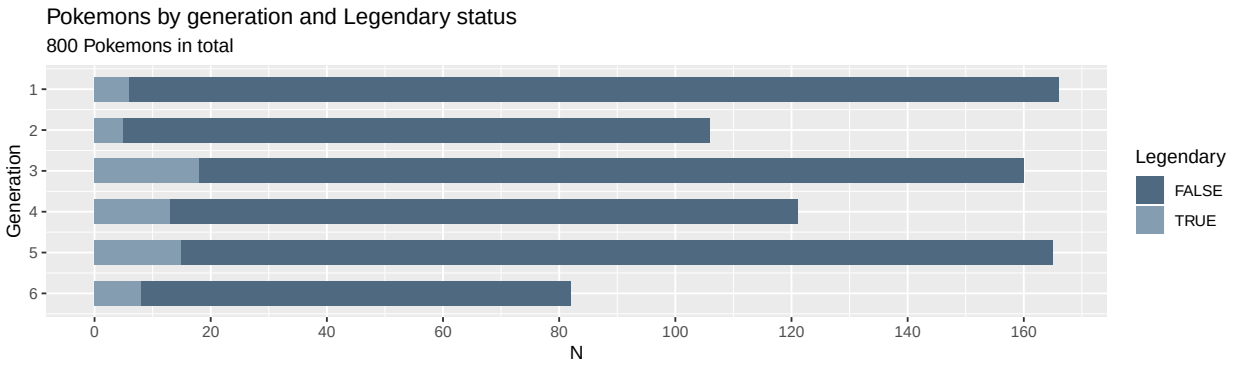
```
library(tidyverse)
library(ggforce)
library(caret)
library(yardstick)
library(knitr)
library(ggthemes)
library(ggridges)
library(kableExtra)

set.seed(15092019)
time <- Sys.time()
poke <- read_csv("https://github.com/SDS-AAU/M1-2019/raw/master/data/pokemon.csv")
```

The original .Rmd file can be found on github https://github.com/LarsHernandez/SDS-Projects-2019/tree/master/M1_assignmen_2 and i recomend using the .html file instead of the pdf.

I load the data and do a first quick visualization of pokemons by generation to get a feeling for the data:

```
poke %>%
  group_by(Generation, Legendary) %>%
  summarize(N=n()) %>%
  ggplot(aes(Generation, N, fill=Legendary)) +
  geom_col(width = 0.6) +
  scale_fill_tableau(palette = "Miller Stone", type = "regular") +
  scale_x_continuous(breaks=c(1,2,3,4,5,6), trans = "reverse") +
  scale_y_continuous(breaks=seq(0,160, by=20)) +
  coord_flip() +
  labs(title="Pokemons by generation and Legendary status",
       subtitle="800 Pokemons in total")
```



1. Unsupervised ML

A. PCA analysis

Execute a PCA analysis on all numerical variables in the dataset. Hint: Don't forget to scale them before. Use 4 components. What is the cumulative explained variance ratio?

I scale the data and use the build in `prcomp` function to calculate the prinicpal components and the eigenvalues

```
poke_scaled <- poke %>%
  select(HitPoints, Attack, Defense, SpecialAttack,
         SpecialDefense, Speed, Generation) %>%
  map_dfc(.f = scale)

PCA <- prcomp(poke_scaled, rank. = 4)
summary(PCA)
```

```
## Importance of first k=4 (out of 7) components:
##                           PC1    PC2    PC3    PC4
## Standard deviation     1.6479 1.0535 0.9932 0.8802
## Proportion of Variance 0.3879 0.1585 0.1409 0.1107
## Cumulative Proportion  0.3879 0.5465 0.6874 0.7981
```

```
## [1] "The cumulative explained variance ratio is 79.81%"
```

I also print the top loading variables for each factor to show what variation is contained within the principle components:

```
loadings <- PCA$rotation %>%
  abs() %>%
  sweep(2, colSums(.), "/") %>%
  as.data.frame %>%
  rownames_to_column("name")

loadings[,-1] <- apply(loadings[,-1], MARGIN = 2, FUN = round, digits=2)

a <- loadings %>% dplyr::select(name, PC1) %>% arrange(desc(PC1))
b <- loadings %>% dplyr::select(name, PC2) %>% arrange(desc(PC2))
c <- loadings %>% dplyr::select(name, PC3) %>% arrange(desc(PC3))
d <- loadings %>% dplyr::select(name, PC4) %>% arrange(desc(PC4))

kable(cbind(a,b,c,d)) %>% kable_styling(bootstrap_options = c("striped", "condensed"))
```

name	PC1	name	PC2	name	PC3	name	PC4
Attack	0.18	Speed	0.30	Generation	0.53	Attack	0.28
SpecialAttack	0.18	Defense	0.26	Defense	0.16	SpecialDefense	0.26
SpecialDefense	0.18	Generation	0.17	SpecialDefense	0.10	HitPoints	0.20
HitPoints	0.16	SpecialAttack	0.13	Speed	0.10	SpecialAttack	0.15
Defense	0.15	SpecialDefense	0.09	SpecialAttack	0.05	Generation	0.06
Speed	0.13	HitPoints	0.05	HitPoints	0.03	Speed	0.03
Generation	0.02	Attack	0.00	Attack	0.03	Defense	0.02

B. Clustering

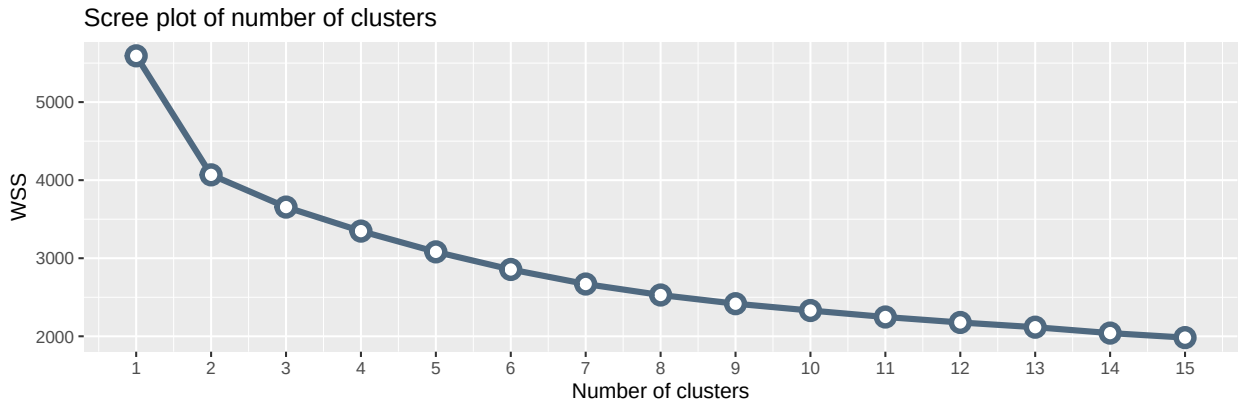
Perform a cluster analysis (either k-means or hierarchical clustering algorithm) on all numerical variables (scaled & before PCA). Apply the elbow method to determine a "pragmatic" number of clusters.

I use the kmeans algorithm to cluster the data, and i calculate the scree plot by running it through an for-loop with values from 1 to 15

```
wss <- 0
for (i in 1:15) {
  wss[i] <- kmeans(poke_scaled, centers = i, nstart=20)$tot.withinss
}

scree <- tibble(wss, N=1:15)

ggplot(scree, aes(N, wss)) +
  geom_line(size=1.5, color="#4f6980") +
  geom_point(size=3, shape=21, fill="white", color="#4f6980", stroke=2) +
  scale_x_continuous(breaks=c(1:15)) +
  labs(title="Scree plot of number of clusters", x="Number of clusters", y="WSS")
```



From this plot i find no conclusive elbow point, but conclude that 4 clusters seems approapiate, i plot the variables with the clusters in a matrix plot, and see that for a good deal of the variables there is a nice seperation, for example notice how defense in cluster 3 is seperated (most with high defense end in cluster 3), or cluster 1 in attack (where most with low attack ends up in cluster 1). This is usefull later.

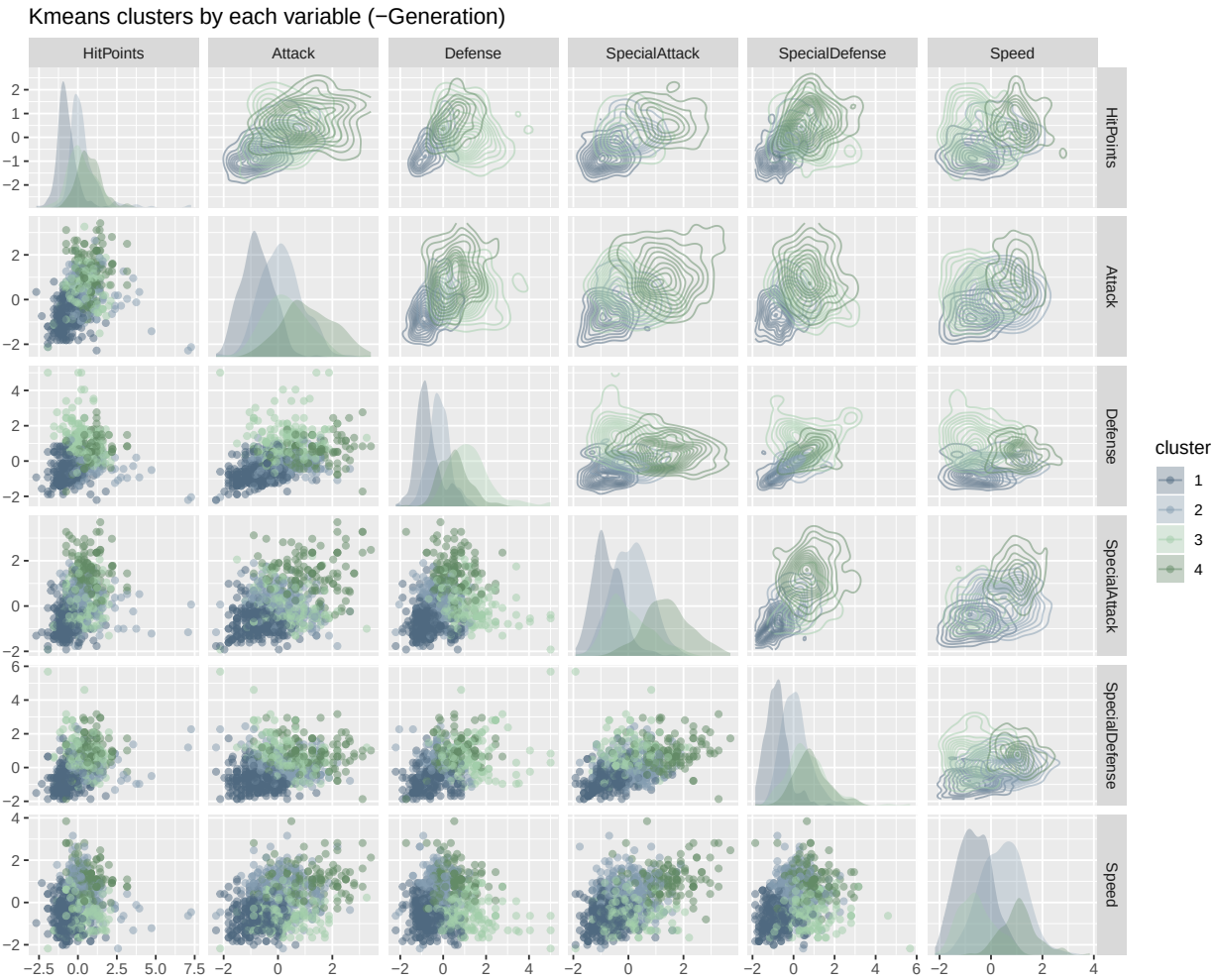
```
km <- poke_scaled %>%
  kmeans(centers = 4, nstart = 20)

poke_scaled$cluster <- as.factor(km$cluster)

ex <- km$centers %>%
```

```
  as_tibble %>%
  mutate(N=1:4) %>%
  gather(variable, value, -N)

ggplot(poke_scaled, aes(x = .panel_x, y = .panel_y, colour = cluster, fill = cluster)) +
  geom_point(alpha = 0.5, position = 'auto') +
  geom_autodensity(alpha = 0.3, colour = NA, position = 'identity') +
  geom_density2d(alpha = 0.5) +
  scale_color_tableau(palette = "Miller Stone", type = "regular") +
  scale_fill_tableau(palette = "Miller Stone", type = "regular") +
  facet_matrix(vars(-cluster, -Generation), layer.diag = 2, layer.upper = 3) +
  labs(title="Kmeans clusters by each variable (-Generation)")
```



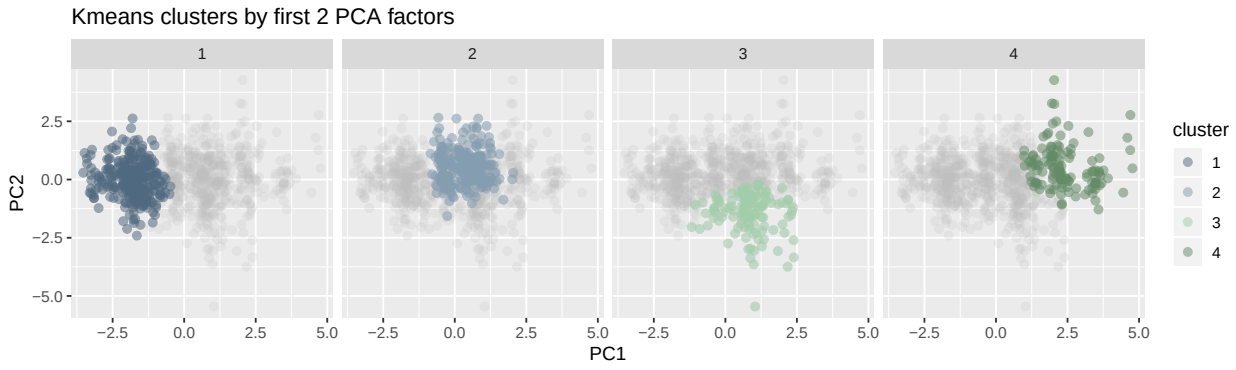
C. Visualization

Visualize the first 2 principal components and color the datapoints by cluster.

We see how the clusters also has seperated the factors from PCA nicely

```
df <- tibble(PC1 = PCA$x[,1], PC2 = PCA$x[,2], cluster=as.factor(km$cluster))
```

```
ggplot(df, aes(PC1, PC2, col = cluster)) +
  geom_point(data=df %>% select(-cluster), aes(PC1, PC2),
             inherit.aes = F, color="grey75", alpha=0.2, size=2) +
  geom_point(alpha=0.5, size=2) +
  scale_color_tableau(palette = "Miller Stone", type = "regular") +
  facet_wrap(~cluster, nrow=1) +
  labs(title="Kmeans clusters by first 2 PCA factors")
```

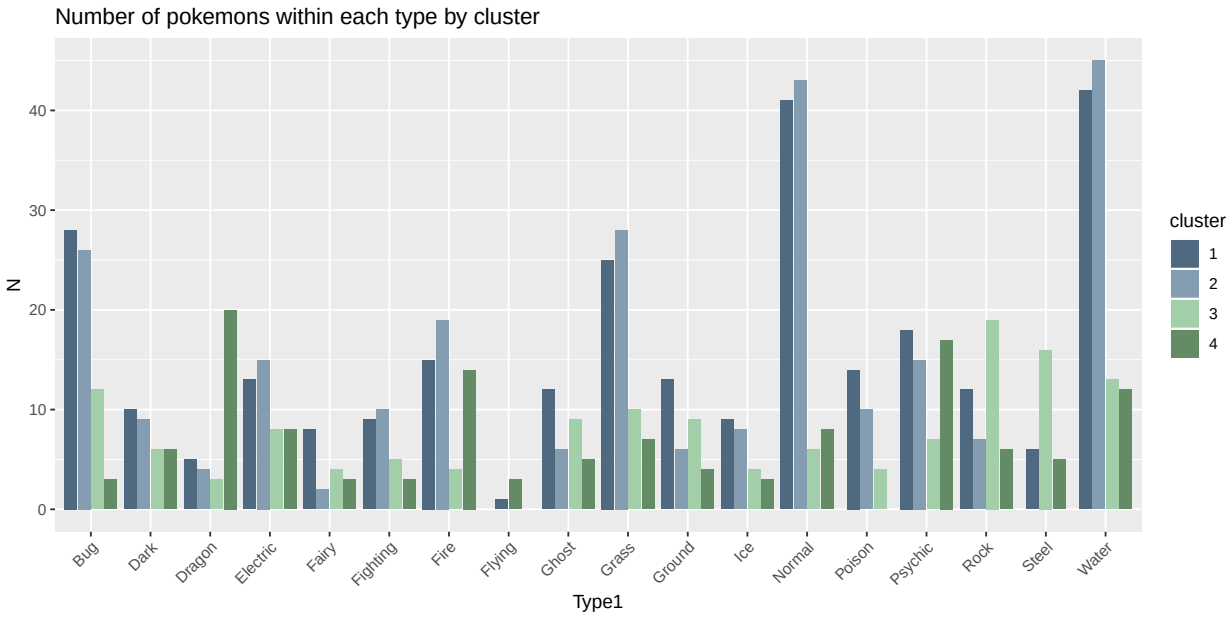


D. Inspection

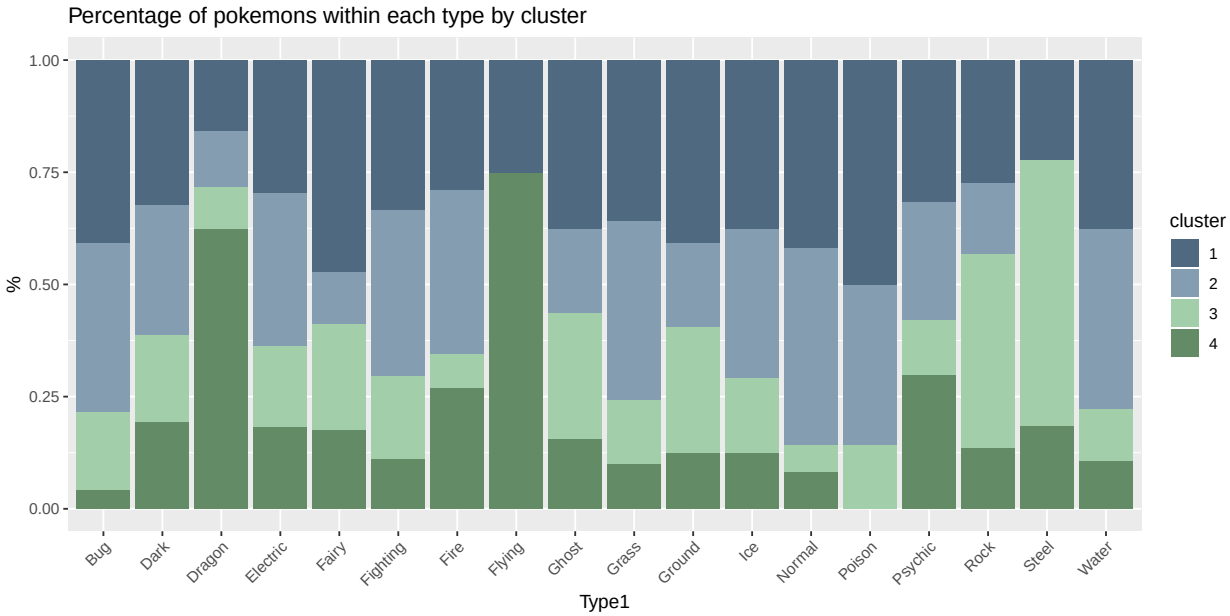
Inspect the distribution of the variable "Type1" across clusters. Does the algorithm separate the different types of pokemon?

From the two plots below we see that `rock` and `steel` fall primarliy in cluster 3, which as we saw before contained observations high in defense, also we see that cluster 1, that contained low attack, takes alot of `fairy` and `poison` but almost no `dragon`. This also makes sense as i presume that dragons have higher attack than faries. But in general the clusters are not that good at seperating the types.

```
poke_scaled %>%
  mutate(Type1 = poke$Type1,
         cluster = as.factor(km$cluster)) %>%
  group_by(Type1, cluster) %>%
  summarize(N=n()) %>%
  ggplot(aes(Type1, N, fill=cluster)) +
  geom_col(position = position_dodge2(width = 0.9, preserve = "single")) +
  scale_fill_tableau(palette = "Miller Stone", type = "regular") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title="Number of pokemons within each type by cluster")
```




```
poke_scaled %>%
  mutate(Type1 = poke$Type1,
         cluster = as.factor(km$cluster)) %>%
  group_by(Type1, cluster) %>%
  summarize(N=n()) %>%
  ggplot(aes(Type1, N, fill=cluster)) +
  geom_col(position = "fill") +
  scale_fill_tableau(palette = "Miller Stone", type = "regular") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
  labs(title="Percentage of pokemons within each type by cluster", y="%")
```



2. Supervised ML

Your task will be to predict the variable "legendary", indicating if the pokemon is a legendary one or not.

A. Preprocessing

Perform necessary ML preprocessing of your data if deemed necessary.

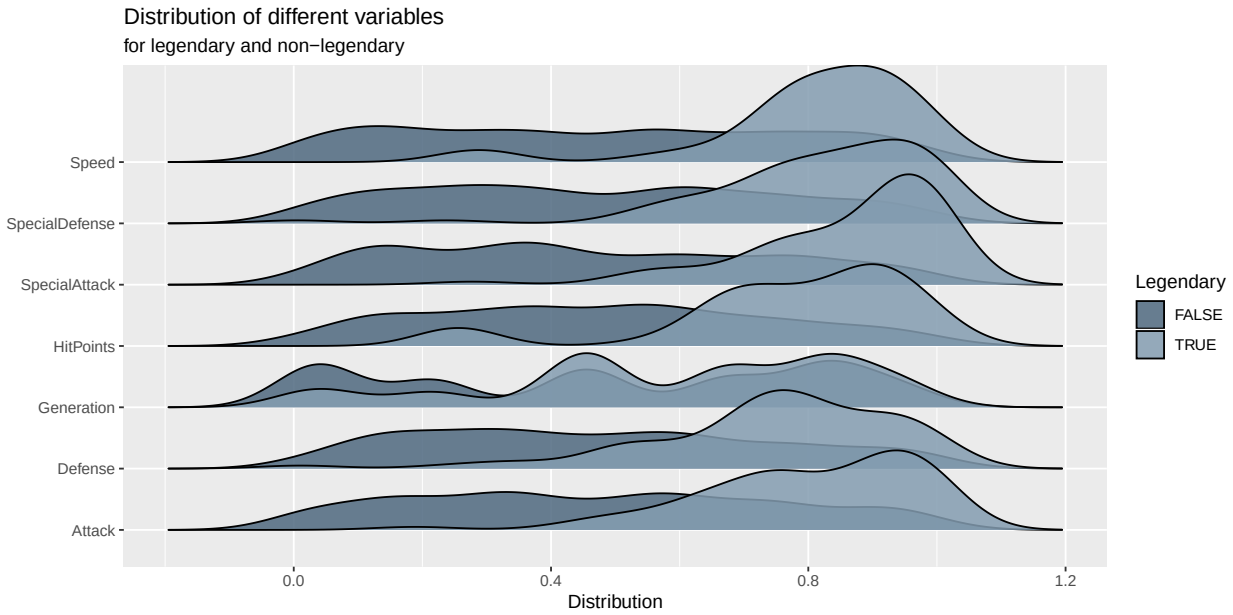
I select the relevant variables and scale them, as this is important to some of my models, especially the regularized logistic regression. Also i print the distribution of the variables by legendary status with the `ggridges` package, and i see a nice pattern that my models should be able to work with.

```
poke_ml <- poke %>%
  select(HitPoints, Attack, Defense, SpecialAttack, SpecialDefense,
         Speed, Generation, Legendary) %>%
  map_dfc(.f = scale) %>%
  mutate(Legendary = as.factor(poke$Legendary))

kable(head(poke_ml),digits = 2) %>%
  kable_styling()
```

HitPoints	Attack	Defense	SpecialAttack	SpecialDefense	Speed	Generation	Legendary
-0.9500319	-0.92432794	-0.7966553	-0.2389808	-0.2480334	-0.8010021	-1.398762	FALSE
-0.3625953	-0.52380252	-0.3476999	0.2194223	0.2909743	-0.2848371	-1.398762	FALSE
0.4206536	0.09239043	0.2936649	0.8306264	1.0096513	0.4033830	-1.398762	FALSE
0.4206536	0.64696408	1.5763945	1.5029509	1.7283282	0.4033830	-1.398762	FALSE
-1.1850065	-0.83189899	-0.9890647	-0.3917818	-0.7870411	-0.1127821	-1.398762	FALSE
-0.4409201	-0.46218322	-0.5080411	0.2194223	-0.2480334	0.4033830	-1.398762	FALSE

```
poke_ml %>%
  gather(variable, value, -Legendary) %>%
  ggplot(aes(y = as.factor(variable),
             fill =  Legendary,
             x = percent_rank(value)) ) +
  scale_fill_tableau(palette = "Miller Stone", type = "regular") +
  geom_density_ridges(alpha = 0.85) +
  labs(title = "Distribution of different variables",
       subtitle = "for legendary and non-legendary",
       fill="Legendary", y="Feature", x="Distribution") +
  theme(axis.title.y = element_blank())
```



B. Split

Split the data in a training (75%) and test (25%) dataset.

```
index    <- createDataPartition(poke_ml$Legendary, p = 0.75, list = FALSE)
training <- poke_ml[index,]
test     <- poke_ml[-index,]

paste0("The training set has ", dim(training)[1],
       " And the test set has ", dim(test)[1], " observations")
```

```
## [1] "The training set has 601 And the test set has 199 observations"
```

C. Cross-validation

Define a n-fold cross-validation workflow for your model testing.

The cross validation is 5-fold

```
cv <- trainControl(method = "cv", number = 5)
```

D. Models

Fit three separate models on your training data, where you predict the "legendary" variable. Use a: 1. Logistic regression, 2. Decision tree, and 3. another algorithm of choice

I fit 5 different models:

- Logistic regression
- Regularized Logistic Regression
- Random Forrest
- Support Vector Machine Model
- Linear Diskriminant Analysis Model

and i tune the parametres to get the highest accuracy. To be able to compare i also do a random assignment based of the probarbility of legendary in the training set.

```
fit_log <- train(Legendary ~ .,
                 data      = training,
                 trControl = cv,
                 tuneGrid  = expand.grid(alpha  = 0.5,
                                         lambda = 0),
                 method    = "glmnet",
                 family    = "binomial",
                 metric    = 'Accuracy')

fit_glm <- train(Legendary ~ .,
                 data      = training,
                 trControl = cv,
                 tuneGrid  = expand.grid(alpha  = seq(0, 1, by = 0.1),
                                         lambda = 10^seq(1, -4, by = -0.2)),
                 method    = "glmnet",
                 family    = "binomial",
                 metric    = 'Accuracy')


fit_raf <- train(Legendary ~ .,
                 data      = training,
                 trControl = cv,
                 tuneGrid  = expand.grid(.mtry = (1:7)),
                 method    = 'rf',
                 metric    = 'Accuracy')

fit_svm <- train(Legendary ~ .,
                 data      = training,
                 trControl = cv,
                 tuneGrid  = expand.grid(C = 10^seq(1, -1, by = -0.02)),
                 method    = 'svmLinear',
                 metric    = 'Accuracy')

fit_lda <- train(Legendary ~ .,
                 data      = training,
                 trControl = cv,
                 method    = "lda",
                 metric    = 'Accuracy')
```

E. Prediction

Use the fitted models to predict the "legendary" variable in your test data.

I predict the legendary status and print the confusion matrices

```
pred_log <- predict(fit_log,  test)
conf_log <- table(pred_log, test$Legendary)

pred_glm <- predict(fit_glm,  test)
conf_glm <- table(pred_glm, test$Legendary)
```

```
pred_svm <- predict(fit_svm,  test)
conf_svm <- table(pred_svm, test$Legendary)

pred_raf <- predict(fit_raf,  test)
conf_raf <- table(pred_raf, test$Legendary)

pred_lda <- predict(fit_lda,  test)
conf_lda <- table(pred_lda, test$Legendary)


random <- rbinom(n    = length(test$Legendary), size = 1,
                 prob = mean(as.logical(training$Legendary)))
conf_ran <- table(as.factor(if_else(random==1, "TRUE", "FALSE")), test$Legendary)


kable(cbind(conf_glm, conf_log, conf_svm, conf_raf, conf_lda, conf_ran)) %>%
  kable_styling() %>%
  column_spec(1, bold = T, color="black") %>%
  add_header_above(c(" "=1, "Elastic Net\nRegression" = 2,
                     "Logistic\nRegression" = 2,
                     "Support Vector\nMachines" = 2,
                     "Random\nForrest" = 2,
                     "Linear Diskriminant\nAnalysis" = 2,
                     "Random\nAssignment" = 2))
```

	Elastic Net Regression		Logistic Regression		Support Vector Machines		Random Forrest		Linear Diskriminant Analysis		Rando Assignm	
	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	TRUE	FALSE	T
FALSE	181	7	181	7	181	9	181	8	182	9	170	
TRUE	2	9	2	9	2	7	2	8	1	7	13	

F. Evaluation

Evaluate the performance of these 3 models by comparing the predicted and the true values of "legendary" in the test data. To do so, also create a confusion matrix.

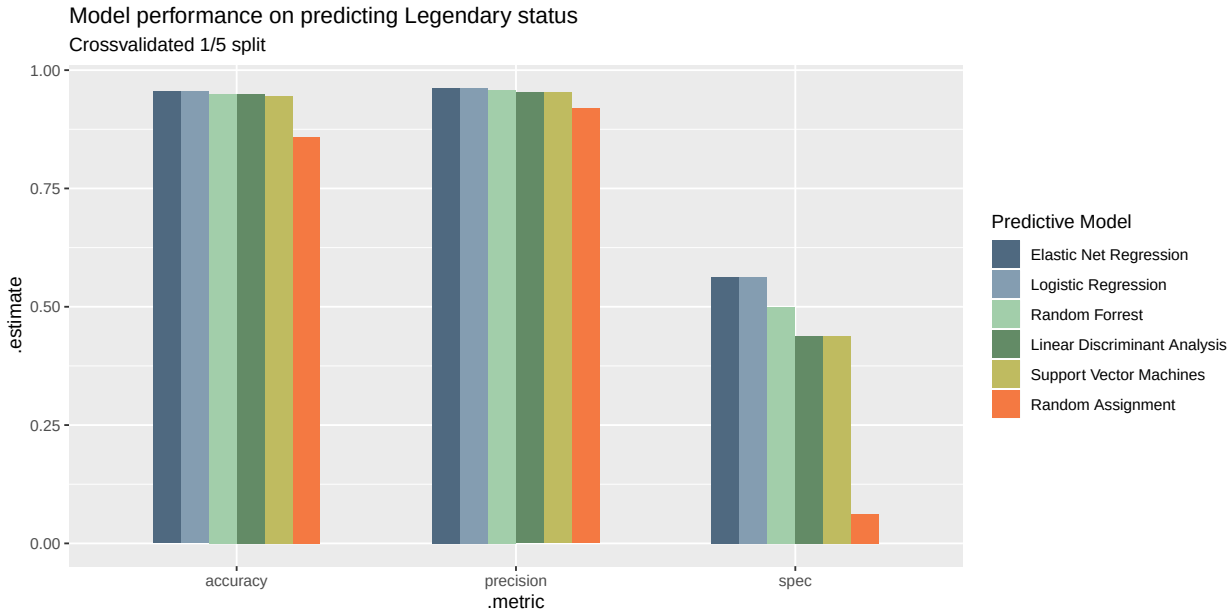
I Evaluate performance by accuracy, precision and specificity:

```
log <- rbind(spec(conf_log), precision(conf_log), accuracy(conf_log))
raf <- rbind(spec(conf_raf), precision(conf_raf), accuracy(conf_raf))
glm <- rbind(spec(conf_glm), precision(conf_glm), accuracy(conf_glm))
svm <- rbind(spec(conf_svm), precision(conf_svm), accuracy(conf_svm))
lda <- rbind(spec(conf_lda), precision(conf_lda), accuracy(conf_lda))
ran <- rbind(spec(conf_ran), precision(conf_ran), accuracy(conf_ran))

log$model <- "Logistic Regression"
raf$model <- "Random Forrest"
glm$model <- "Elastic Net Regression"
svm$model <- "Support Vector Machines"
lda$model <- "Linear Discriminant Analysis"
ran$model <- "Random Assignment"

df <- rbind(log, raf, glm, svm, lda, ran)
```

```
ggplot(df, aes(.metric, .estimate, fill = reorder(model, desc(.estimate)))) +
  geom_col(position="dodge", width = 0.6) +
  scale_fill_tableau(palette = "Miller Stone", type = "regular") +
  labs(title="Model performance on predicting Legendary status",
       subtitle="Crossvalidated 1/5 split", fill="Predictive Model")
```



I conclude that the ML models improve the prediction of Legendary status, but as the simple logistic regression performs well there is in reality no need for the black box ML algorithms. This makes sense as the data generating process is just japanese writers comming up with some values, and the legendary are assigned higher. Therefore it makes sense that this relationship can be captured by the logistic regression, and that any better performance of other models will be overfitting.

Below here i print the coeffecients from the basic logistic regression, and we see how Legendary pokemons are higher in all specs, especially speed.

```
model <- glm(Legendary ~ .,
             family = binomial(link='logit'),
             data = training)
summary(model)
```

```
##
## Call:
## glm(formula = Legendary ~ ., family = binomial(link = "logit"),
##     data = training)
##
## Deviance Residuals:
##      Min        1Q    Median        3Q       Max
## -2.33117  -0.12677  -0.02139  -0.00273   2.39759
##
## Coefficients:
##                Estimate Std. Error z value Pr(>|z|)
## (Intercept)     -7.4218     0.9446  -7.857 3.92e-15 ***
## HitPoints        0.9824     0.2848   3.449 0.000563 ***
## Attack           0.5127     0.2586   1.982 0.047447 *
```

```
## Defense          1.4629     0.3296   4.438 9.07e-06 ***
## SpecialAttack    1.1490     0.2957   3.886 0.000102 ***
## SpecialDefense   1.3553     0.2934   4.619 3.86e-06 ***
## Speed            1.6893     0.3535   4.778 1.77e-06 ***
## Generation       0.7255     0.2669   2.718 0.006565 **
## ---
## Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1
##
## (Dispersion parameter for binomial family taken to be 1)
##
##     Null deviance: 339.56  on 600  degrees of freedom
## Residual deviance: 127.65  on 593  degrees of freedom
## AIC: 143.65
##
## Number of Fisher Scoring iterations: 9
```

```
paste0("Time to knit document: ", round(difftime(Sys.time(), time,units = "mins")), " minut(s)")
```

```
## [1] "Time to knit document: 1 minut(s)"
```

Submission 18. September 12:00. Peergrade.io (link + submission details will be sent out on Monday) Please submit a PDF version of your notebook with a link to the corresponding colab notebook included. Please make sure(eg. own test in "anonymous" setting in your browser) that others can acess it.