In my last post It scraped some character statistics from the mobile game Star Wars: Galaxy of Heroes. In this post, We’ll be aiming to try out k-means clustering in order to see if it comes out with an intuitive result, and to learn how to integrate this kind of analysis into a tidy workflow using broom.

First We’ll load the required packages and set some plot preferences.

library(tidyverse)

## ── Attaching packages ─────────────────────────────────────────────────────────────────────────── tidyverse 1.2.1 ──

## ✔ ggplot2 3.2.0 ✔ purrr 0.3.2

## ✔ tibble 2.1.3 ✔ dplyr 0.8.1

## ✔ tidyr 0.8.3 ✔ stringr 1.4.0

## ✔ readr 1.3.1 ✔ forcats 0.4.0

## ── Conflicts ────────────────────────────────────────────────────────────────────────────── tidyverse\_conflicts() ──

## ✖ dplyr::filter() masks stats::filter()

## ✖ dplyr::lag() masks stats::lag()

library(broom)

theme\_set(theme\_bw())

Here is the data I had previously scraped. It contains 176 Star Wars characters, each with 16 performance attributes.

swgoh\_stats <- read\_csv("content/post/data/unsupervised-learning/swgoh\_stats.csv")

## Parsed with column specification:

## cols(

## `Character Name` = col\_character(),

## Power = col\_double(),

## Speed = col\_double(),

## Health = col\_double(),

## `Max Ability` = col\_double(),

## `Physical Dmg` = col\_double(),

## `Physical Crit` = col\_double(),

## `Special Dmg` = col\_double(),

## `Special Crit` = col\_double(),

## `Armor Pen` = col\_double(),

## `Resistance Pen` = col\_double(),

## Potency = col\_double(),

## Protection = col\_double(),

## Armor = col\_double(),

## Resistance = col\_double(),

## Tenacity = col\_double(),

## `Health Steal` = col\_double()

## )

glimpse(swgoh\_stats)

## Observations: 176

## Variables: 17

## $ `Character Name` "Aayla Secura", "Admiral Ackbar", "Ahsoka Tano"…

## $ Power 18972, 18972, 21378, 21378, 21378, 24358, 25143…

## $ Speed 145, 139, 125, 168, 110, 124, 167, 165, 131, 13…

## $ Health 32108, 35392, 27138, 30636, 47459, 35381, 34870…

## $ `Max Ability` 11375, 6020, 11223, 10728, 4485, 7633, 11982, 4…

## $ `Physical Dmg` 3602, 3286, 3657, 3617, 2056, 2970, 3804, 4207,…

## $ `Physical Crit` 1022, 333, 1110, 1079, 470, 734, 869, 812, 575,…

## $ `Special Dmg` 2356, 3718, 1469, 1770, 3366, 2772, 1559, 1363,…

## $ `Special Crit` 0, 60, 30, 40, 75, 115, 0, 60, 0, 90, 225, 565,…

## $ `Armor Pen` 184, 75, 329, 164, 5, 144, 307, 517, 81, 35, 5,…

## $ `Resistance Pen` 15, 37, 0, 0, 60, 197, 0, 5, 5, 5, 60, 185, 0, …

## $ Potency 0.41, 0.36, 0.23, 0.38, 0.07, 0.43, 0.33, 0.72,…

## $ Protection 41388, 39690, 34900, 39500, 54640, 36710, 44131…

## $ Armor 309, 454, 295, 301, 488, 289, 278, 217, 508, 41…

## $ Resistance 116, 423, 110, 152, 471, 124, 104, 76, 431, 253…

## $ Tenacity 0.50, 0.60, 0.33, 0.54, 0.56, 0.42, 0.38, 0.29,…

## $ `Health Steal` 0.15, 0.15, 0.35, 0.15, 0.30, 0.55, 0.20, 0.05,…

Some things to note straight off the bat. This algorithm can only deal with numeric data as it calculates distances from centroids. We also have to provide it with a number of clusters (centers) which requires either some a priori knowledge of the data, or some experimentation (or both).

There is also a stochastic element to the algorithm as the first step assigns each point to a random cluster. The algorithm can be repeated several times (with the nstart parameter) and the best performing run used. This means that if reproducibility is needed, a random number seed must be set. I’ll be setting a new seed every time I generate some analysis with specific attributions I’d like to have reproduced.

First, let’s have a look at a single kmeans() model using an arbitrary 3 clusters:

set.seed(1)

test\_km <- kmeans(swgoh\_stats[,-1], centers = 3, nstart = 50)

test\_km

## K-means clustering with 3 clusters of sizes 81, 51, 44

##

## Cluster means:

## Power Speed Health Max Ability Physical Dmg Physical Crit

## 1 21025.85 149.7654 34730.53 8365.827 3255.432 730.3086

## 2 21115.14 158.9608 32173.96 9040.569 3320.804 802.4118

## 3 21369.43 141.5455 42204.59 7059.500 3200.682 566.2045

## Special Dmg Special Crit Armor Pen Resistance Pen Potency Protection

## 1 2652.531 97.55556 149.7160 40.86420 0.4234568 41969.47

## 2 2673.176 121.09804 169.5098 54.33333 0.4323529 34069.16

## 3 2194.000 45.79545 105.5000 11.31818 0.4095455 51363.80

## Armor Resistance Tenacity Health Steal

## 1 366.1728 238.2222 0.4413580 0.1709877

## 2 311.6471 209.3333 0.4249020 0.1568627

## 3 517.8409 363.3864 0.4929545 0.1977273

##

## Clustering vector:

## [1] 1 1 2 1 3 2 1 2 1 1 3 1 1 1 1 1 2 1 3 3 1 1 1 1 1 1 2 3 1 3 2 3 1 3 1

## [36] 2 1 2 1 3 2 3 1 1 2 2 3 1 1 2 1 1 2 3 1 1 2 1 1 2 2 3 1 1 1 3 2 3 2 1

## [71] 3 3 1 3 2 2 1 2 2 2 1 2 3 1 2 3 3 2 2 3 1 1 1 1 2 1 2 1 1 2 1 3 2 3 3

## [106] 1 1 1 3 2 1 2 2 2 1 1 1 1 3 2 3 2 3 1 1 3 3 2 3 1 1 2 2 1 2 1 1 1 2 2

## [141] 1 2 2 3 2 1 3 3 3 2 1 3 1 3 1 3 1 1 3 2 3 1 1 1 1 3 1 1 1 3 2 1 2 2 3

## [176] 1

##

## Within cluster sum of squares by cluster:

## [1] 2788849852 2446171455 2763654327

## (between\_SS / total\_SS = 54.9 %)

##

## Available components:

##

## [1] "cluster" "centers" "totss" "withinss"

## [5] "tot.withinss" "betweenss" "size" "iter"

## [9] "ifault"

Printing the model object gives us some useful information, including cluster sizes, cluster means (centroids), a vector of cluster attributions for each character, and some information about the performance of the clustering. For k-means, the objective is to maximise the between-cluster sum of squares (variance) and minimise the within-cluster sum of squares, i.e. have tight clusters that are well separated. The total sum of squares is the total of these two figures, and so ideally we’d like to have as much of the variance as possible coming from the between-cluster variance instead of the within-cluster variance. Hence, 54.9% is a measure of cluster performance. If every point was its own cluster, it would be 100%.

Looking at the summary, we get some slightly different information, which indicates that integrating this model into a tidy workflow may be tricky as it is essentially a list of different sized elements.

summary(test\_km)

## Length Class Mode

## cluster 176 -none- numeric

## centers 48 -none- numeric

## totss 1 -none- numeric

## withinss 3 -none- numeric

## tot.withinss 1 -none- numeric

## betweenss 1 -none- numeric

## size 3 -none- numeric

## iter 1 -none- numeric

## ifault 1 -none- numeric

This is where the broom package comes in, which is installed with the tidyverse. This package is used for turning messy model objects like this into tidy tibbles and has three main functions:

The glance() function gives a single row of top-level metrics. This is useful for extracting performance measures.

glance(test\_km)

## # A tibble: 1 x 4

## totss tot.withinss betweenss iter

##

## 1 17741410664. 7998675634. 9742735030. 3

The tidy() function gives a row for each cluster. I can’t imagine I’d use this one too often for k-means.

tidy(test\_km)

## # A tibble: 3 x 19

## x1 x2 x3 x4 x5 x6 x7 x8 x9 x10 x11

##

## 1 21026. 150. 34731. 8366. 3255. 730. 2653. 97.6 150. 40.9 0.423

## 2 21115. 159. 32174. 9041. 3321. 802. 2673. 121. 170. 54.3 0.432

## 3 21369. 142. 42205. 7060. 3201. 566. 2194 45.8 106. 11.3 0.410

## # … with 8 more variables: x12 , x13 , x14 , x15 ,

## # x16 , size , withinss , cluster

The augment() function gives a row for each observation. You not only have to provide the model object, but also the original data (exactly as it was originally used) and it augments it with cluster attribution. This is where the main results lie.

augment(test\_km, swgoh\_stats[,-1])

## # A tibble: 176 x 17

## Power Speed Health Max.Ability Physical.Dmg Physical.Crit Special.Dmg

##

## 1 18972 145 32108 11375 3602 1022 2356

## 2 18972 139 35392 6020 3286 333 3718

## 3 21378 125 27138 11223 3657 1110 1469

## 4 21378 168 30636 10728 3617 1079 1770

## 5 21378 110 47459 4485 2056 470 3366

## 6 24358 124 35381 7633 2970 734 2772

## 7 25143 167 34870 11982 3804 869 1559

## 8 21378 165 20693 4205 4207 812 1363

## 9 21378 131 36543 4020 3126 575 2226

## 10 21378 136 37121 4215 3332 904 2487

## # … with 166 more rows, and 10 more variables: Special.Crit ,

## # Armor.Pen , Resistance.Pen , Potency ,

## # Protection , Armor , Resistance , Tenacity ,

## # Health.Steal , .cluster

The question then is how to implement these into a tidy workflow, especially when you may want to experiment with many different models. After reading about it online and some experimentation, I think I have it figured out. The trick seems to involve 3 steps:

1. Create a tibble with a list column of models;
2. Use one of the broom functions above to create another list column, dependent on the question being asked;
3. Unnest the broom-derived list column to surface the appropriate model parameters in the tibble.

As an example, the first thing to do here is to find an appropriate number of clusters to use. Therefore I create a tibble with a row for a number of clusters from 1 to 20, and map each of these numbers to the kmeans algorithm to create a column of models. Since I’m interested in how model performance changes with the number of clusters, I map each model to the glance() function and unnest it. I can then generate a scree plot to try to find an appropriate elbow in the curve. Instead of using the total within-cluster sum of squares, I normalise it as it I can interpret it more usefully.

tibble(clusters = 1:20) %>%

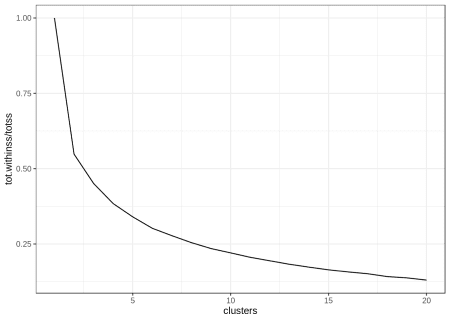
mutate(mod = map(clusters, ~ kmeans(swgoh\_stats[,-1], centers = .x, nstart = 50))) %>%

mutate(glanced = map(mod, glance)) %>%

unnest(glanced) %>%

ggplot(aes(x=clusters, y = tot.withinss/totss)) +

geom\_line()



The scree plot says that when there is once cluster performance is at its worst possible, however it’s difficult to choose a number of clusters as there’s no obvious knee (apart from the one at clusters = 2, which has performance of around 43%). I’ll initially go with 10 as that seems to give around 80% performance.

Now I’ve chosen a number of clusters, I’ll look at how characters are clustered for each attribute. This time I use the augment() function to get cluster attributions. Due to the size of the data, I also only use a random 50% of the data from each cluster:

set.seed(123)

kmeans(swgoh\_stats[,-1], centers = 10, nstart = 50) %>%

augment(swgoh\_stats[,-1]) %>%

bind\_cols(swgoh\_stats[,1]) %>%

mutate(`Character Name` = fct\_reorder(`Character Name`, paste0(.cluster))) %>%

group\_by(.cluster) %>%

sample\_frac(0.5) %>%

ungroup() %>%

gather(attribute, value, -`Character Name`, -.cluster) %>%

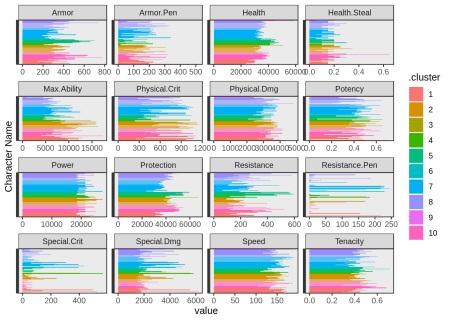
ggplot(aes(x = `Character Name`, y = value, fill = .cluster)) +

geom\_col() +

coord\_flip() +

facet\_wrap(~attribute, drop = TRUE, scales = "free") +

theme(axis.text.y = element\_blank())



In this plot, I’m surprised to see that it’s difficult to distinguish between clusters, except for health and protection. Since these have by far the biggest values and I don’t want to cluster by these alone, it’s clear I have to scale the values. One thing I learned is that the scale() function is used for scaling and centering, and it does both by default. It also returns a matrix, so I have to turn it back into a tibble.

Let’s redo the scree plot.

swgoh\_scaled <- scale(swgoh\_stats[,-1]) %>% as\_tibble()

tibble(clusters = 1:20) %>%

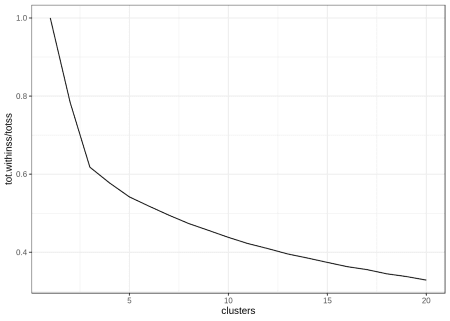
mutate(mod = map(clusters, ~ kmeans(swgoh\_scaled, centers = .x, nstart = 50))) %>%

mutate(glanced = map(mod, glance)) %>%

unnest(glanced) %>%

ggplot(aes(x=clusters, y = tot.withinss/totss)) +

geom\_line()



The scree plot doesn’t look too different, so I’ll stick with 10 clusters for now.

set.seed(124)

scaled\_clustering <- kmeans(swgoh\_scaled, centers = 10, nstart = 50) %>%

augment(swgoh\_scaled) %>%

bind\_cols(swgoh\_stats[,1])

Let’s see if the clustering has improved across attributes:

scaled\_clustering %>%

mutate(`Character Name` = fct\_reorder(`Character Name`, paste0(.cluster))) %>%

group\_by(.cluster) %>%

sample\_frac(0.5) %>%

ungroup() %>%

gather(attribute, value, -`Character Name`, -.cluster) %>%

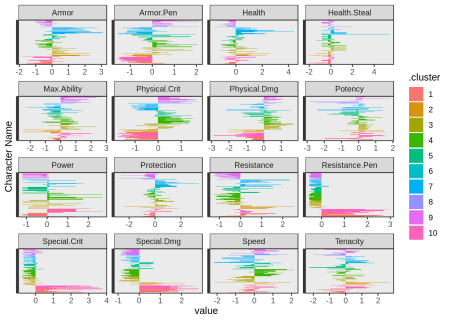
ggplot(aes(x = `Character Name`, y = value, fill = .cluster)) +

geom\_col() +

coord\_flip() +

facet\_wrap(~attribute, drop=TRUE, scales = "free") +

theme(axis.text.y = element\_blank())



This seems much better. Next, I’d like to see which characters are being clustered together to see if I can spot any patterns. As there’s no wordcloud geom for ggplot2 I just create a bit of a simple one, using ggrepel to avoid overlapping character names as much as possible.

scaled\_clustering %>%

select(`Character Name`, .cluster) %>%

mutate(x = runif(n(), 0, 10),

y = runif(n(), 0, 10)) %>%

ggplot(aes(x=x,y=y, label=`Character Name`)) +

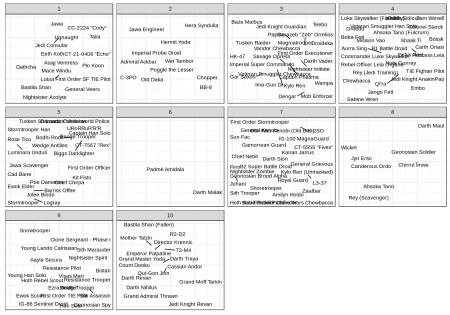
ggrepel::geom\_text\_repel(size = 3) +

theme(axis.text = element\_blank(),

axis.ticks = element\_blank(),

axis.title = element\_blank()) +

facet\_wrap(~.cluster)



From what I know about the game, some interesting things pop out to me straightaway:

* Cluster 2 seems to contain support characters – for example Jawa Engineer speeds up droids, and Hermit Yoda provides buffs to Jedi characters.
* Cluster 6 contains only two characters and both of these were introduced very recently. Perhaps the developers are trying characters with a different kind of balance;
* Cluster 7 seems to have captured many characters known as “tanks”. These have high health/protection and are designed to soak up damage;
* Cluster 8 contains some notable attackers with high damage output.

Spotting these patterns is quite encouraging! I wasn’t originally intending to do this, but I am fascinated to find out how these clusters map to character roles. There’s no way I can find of downloading this data, so I manually created it from data on the website:

swgoh\_roles <- read\_csv("content/post/data/unsupervised-learning/swgoh\_roles.csv") %>%

mutate(role = case\_when(!is.na(attacker) ~ "Attacker",

!is.na(support) ~ "Support",

!is.na(tank) ~ "Tank",

!is.na(healer) ~ "Healer")) %>%

select(`Character Name`, role)

## Parsed with column specification:

## cols(

## `Character Name` = col\_character(),

## attacker = col\_double(),

## healer = col\_double(),

## support = col\_double(),

## tank = col\_double()

## )

glimpse(swgoh\_roles)

## Observations: 176

## Variables: 2

## $ `Character Name` "Aayla Secura", "Admiral Ackbar", "Ahsoka Tano"…

## $ role "Support", "Support", "Attacker", "Attacker", "…

Let’s recreate the ten ‘wordclouds’ to see how the characters are being clustered by role:

scaled\_clustering %>%

left\_join(swgoh\_roles) %>%

select(`Character Name`, .cluster, role) %>%

mutate(x = runif(n(), 0, 10),

y = runif(n(), 0, 10)) %>%

ggplot(aes(x=x,y=y, label=`Character Name`)) +

ggrepel::geom\_text\_repel(aes(col = role), size = 3) +

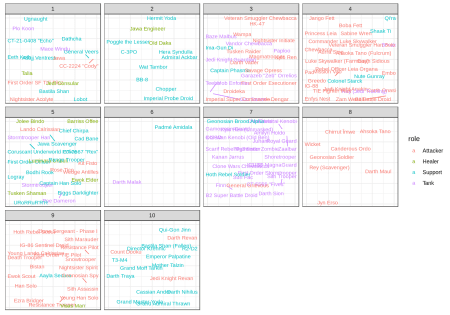
theme(axis.text = element\_blank(),

axis.ticks = element\_blank(),

axis.title = element\_blank()) +

facet\_wrap(~.cluster)

## Joining, by = "Character Name"



It does seem that my observations were correct, and tanks are indeed focussed in cluster 7. Cluster 8 also seems to exclusively contain attackers, with the role also being prevalent in clusters 4 and 9. Clusters 2 and 10 seem to be focussed on support roles.

Given there are 4 roles, I’m curious to see how well k-means is able to distinguish between them if we assume 4 clusters:

set.seed(125)

kmeans(swgoh\_scaled, centers = 4, nstart = 50) %>%

augment(swgoh\_scaled) %>%

bind\_cols(swgoh\_stats[,1]) %>%

left\_join(swgoh\_roles) %>%

select(`Character Name`, .cluster, role) %>%

mutate(x = runif(n(), 0, 10),

y = runif(n(), 0, 10)) %>%

ggplot(aes(x=x,y=y, label=`Character Name`)) +

ggrepel::geom\_text\_repel(aes(col = role), size = 3) +

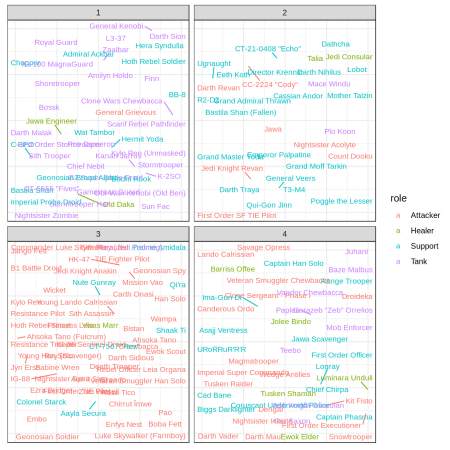
theme(axis.text = element\_blank(),

axis.ticks = element\_blank(),

axis.title = element\_blank()) +

facet\_wrap(~.cluster)

## Joining, by = "Character Name"



Interesting! Three of the clusters seem to be capuring support, attacker, and tank characters pretty well, however the fourth cluster seems to be a bit of a mess…this is probably because healing abilities are given to a range of characters.

This seems to be a pretty good algorithm, I’m interested to see how others compare. Next up, hierarchical clustering!