**Motivation**

I’ve written before about my learning plans, which always seem to be in a state of flux, and in particular learning about machine learning. Part of the reason why I’m so reticent is because I’m a mathematician and statistics does not come natural or easy for me.

My limited past experience has exposed to me just how much I don’t know. It’s fairly easy to apply a statistical model in R, and even have a go at assessing its performance, however I am acutely aware that there is a certain ‘dark art’ to it requiring a deeper understanding of knowing exactly how to interpret results, and how far you can take it. This is not something I don’t think I would ever feel comfortable doing without being a statistician.

However, my mental model of machine learning has this being particularly applicable to supervised learning. Unsupervised learning, to me, seems to be mainly linear algebra from what I can tell – a subject I am much more comfortable with. Yes, I’m conveniently ignoring reinforcement learning, and yes, there is some overlap between supervised and unsupervised learning. However, speaking crudely, in a manner that helps steer my own development, I believe it’s a decent rule of thumb to go with for now. It also seems to nicely align with my preference for EDA, rather than prediction.

I also realise that such a decision may draw criticisms such as “to be a decent data scientist you need to at least know how to apply linear and logistic regression”. I get that, and I do know the principles, but I’m a perfectionist and I am burdened with a need to nail a topic (within reason) before moving on to the next, and unsupervised learning seems like lower hanging fruit (and seems to have more utility).

With that preamble out of the way, I’ve decided that now is the time for me to start looking into unsupervised techniques. I plan to cover bread and butter algorithms such as PCA, k-means clustering, and hierarchical clustering, all the way through to more exotic algorithms like Self-Organising Maps and t-distributed stochastic neighbour embedding. In a series of blog posts I want to cover these algortihms and find packages and workflows I feel comfortable using. I hope to have got through most of it by Christmas.

**The data**

In one of my first blog posts, I wrote about a fairly substantial personal project to write an optimisation algorithm to help with the mobile game Star Wars: Galaxy of Heroes. Despite not having played the game for several years, I do know it has some nice datasets to use for unsupervised learning and I think some contextual knowledge may help. I may need some other data eventually, but this will do for now.

I’m going to attempt to scrape the data from a website, clean it up, and try out the ggpairs() function from the GGally package. This seems to be the most versatile function for creating matrix plots to examine distributions and correlations of variables in a dataset.

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(rvest)

## Loading required package: xml2

##

## Attaching package: 'rvest'

## The following object is masked from 'package:purrr':

##

## pluck

## The following object is masked from 'package:readr':

##

## guess\_encoding

library(GGally)

##

## Attaching package: 'GGally'

## The following object is masked from 'package:dplyr':

##

## nasa

I get the URL of the website, and then use Hadley Wickham’s fantastic rvest package to scrape the table from the website.

swgoh\_url <- "https://swgoh.gg/characters/stats/"

stats <- read\_html(swgoh\_url) %>%

html\_table() %>%

purrr::pluck(1)

glimpse(stats)

## Observations: 176

## Variables: 28

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

## $ Power 18972, 18972, 21378, 21378, 21378, 24358, 2…

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

## $ Health "32,108", "35,392", "27,138", "30,636", "47…

## $ `Max Ability` "11,375", "6,020", "11,223", "10,728", "4,4…

## $ `Physical Dmg` 3602, 3286, 3657, 3617, 2056, 2970, 3804, 4…

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

## $ `Special Dmg` 2356, 3718, 1469, 1770, 3366, 2772, 1559, 1…

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

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

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

## $ Potency 0.41, 0.36, 0.23, 0.38, 0.07, 0.43, 0.33, 0…

## $ Protection "41,388", "39,690", "34,900", "39,500", "54…

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

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

## $ Tenacity "50%", "60%", "33%", "54%", "56%", "42%", "…

## $ `Health Steal` "15%", "15%", "35%", "15%", "30%", "55%", "…

## $ Tier 13, 13, 13, 13, 13, 13, 13, 13, 13, 13, 13,…

## $ Credits "1,434,250", "1,485,950", "1,469,750", "1,5…

## $ Raid 1, 4, 3, 4, 3, 2, 3, 3, 4, 2, 3, 1, 3, 5, 3…

## $ Gold 860, 810, 970, 860, 590, 810, 750, 640, 750…

## $ Purple 1515, 1290, 1315, 1650, 1410, 1340, 1720, 1…

## $ Blue 198, 390, 338, 228, 254, 304, 317, 452, 240…

## $ Green 74, 87, 68, 100, 79, 69, 74, 120, 122, 86, …

## $ White 66, 94, 63, 98, 49, 33, 74, 150, 114, 78, 4…

## $ `Max Damage Ability` "11,375", "6,020", "11,223", "10,728", "4,4…

## $ `Base Ability` "6,221", "6,020", "5,052", "10,728", "4,485…

## $ `AoE Ability` "0", "0", "0", "0", "3,855", "3,981", "0", …

That was just ridiculously easy. I love R.

The data is pretty clean already, and mostly consists of numeric variables. There are however some columns which require some work. Since some of the numbers are large enough to use commas, we need to parse these as numbers, and also convert some of the percentages to decimals. I’ll also only use a subset of attributes of characters.

stats\_clean <- stats %>%

select(`Character Name`:`Health Steal`) %>%

mutate\_at(vars(Health, `Max Ability`, Protection, Tenacity, `Health Steal`), parse\_number) %>%

mutate\_at(vars(Tenacity, `Health Steal`), ~ ./100)

glimpse(stats\_clean)

## 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,…

As I don’t want to keep scraping the website every time I want to use the data, I’ll write it to CSV:

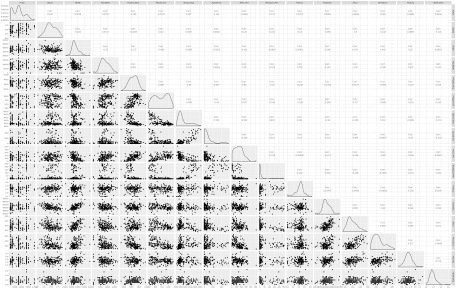
stats\_clean %>%

write\_csv(here::here("content/post/data/unsupervised-learning/swgoh\_stats.csv"))

**Initial EDA**

Next, I’ll plug the data (except character names) into the ggpairs() function. As I was drafting this blog post, it was taking a very long time to run, but at the time of writing it seems to have corrected itself. Nevertheless, I’ve saved the output as an image as it has quite a few plots on it.

ggpairs(stats\_clean[,2:17])



This is some great functionality, and I was somewhat surprised that these attributes didn’t seem to be more correlated. I think this means that when I get around to doing PCA, I’m going to find I’m going to need to retain quite a few Principle Components to preserve the majority of the variability in the data.

The ggpairs() function has quite a few options which I’m not going to explore here now as I’m going to try to keep the posts in this series fairly brief.