

Machine learning applied to the Pokemon game

Pedro Martins 54390

Faculdade de Ciências e Tecnologia, Universidade do Algarve

ABSTRACT

In this article I apply algorithms that were learned during the machine learning course to the pokemon game, more specifically a dataset of this game. Prediction of the type of a pokemon and the winner of a combat were the main focus in this project. The results show that we are able to predict with great accuracy the winner of a combat but aren't able to predict the type of a pokemon.

1 Introduction

The pokemon franchise began back in 1996. It started as a Gameboy game developed by GameFreak and published by Nintendo. It started as a simple game but developed to being today the highest grossing media franchise of all time. The games are the second best-selling video games franchise ever made. Pokemon is more than a game, it has fans everywhere, from anime, card trading games, toys, books, manga, music and even theme parks all over the world. Currently pokemon has gained a lot of attention with the release of *Pokemon Go*.

For someone who doesn't know much about the game, at first sight it seems a childish game. In truth it began that way. But the game, also has a competitive aspect to it. In the game, you are a trainer that gathers pokemon to battle other trainers. Your main goal is to defeat all the trainers and become the Champion. In this game, pokemons are like gladiators, you are the owner of them, and they fight in your behalf, until they die.

In the game, each pokemon has many attributes that represent them. Each pokemon is different, and the strategy to the game is having the perfectly balanced 6 pokemon team that will allow you to become the ultimate trainer. Also, there is strategy to the battle itself, but we are not going into it. Anyways a pokemon has the following attributes:

- **Level** – Represents the total battle experience of this pokemon. The higher the level, the higher all the other stats will be.
- **Type** – A pokemon can have its type water, fire, grass, ghost, and so on. Also, a pokemon can have at maximum 2 types. So, it can be a rock and water pokemon for example.
- **HP** – Also known as hitpoints, it represents the amount of "health" this pokemon has. We can think of this as how much damage can this pokemon resist until they die.
- **Defense** – This attribute represents how tough the pokemon is. The higher the defense the lower enemies attack will be.

- **Attack** – How strong does this pokemon hit with a basic attack.
- **Special Attack** – How strong does this pokemon hit with an attack of its type. (Flamethrower, water gun, etc.)
- **Special Defense** – The defense this pokemon has against special attack moves.
- **Speed** – The higher this attribute the quicker a pokemon will hit. A pokemon with a higher speed stat will always attack first than another pokemon.

A pokemon can have type advantages over another pokemon, giving it extra edge. For example, water beats fire right? So, if you are going to battle a fire type trainer, you may as well only use water pokemon, you'll have a much easier time defeating him.

This is the basic idea (very basic) idea of the game. So, the goal of this article is the following:

- Can we infer the type of a pokemon solely by its attributes? Using domain knowledge, we know that rock pokemon have higher defense, while fire pokemon have a much higher attack.
- Can we predict the winner of a pokemon battle?

2 Implementation

In order to achieve the goals of the project, we need to use some machine learning algorithms. But the algorithms are nothing if we don't have a dataset. So, the first thing was gathering a dataset from Kaggle [1]. With this we got 3 different datasets:

- **Pokemon.csv**: A list containing all available pokemon and their attributes.
- **Combats.csv**: A list consisting on a previously seen combat between two pokemons and its respective winner.
- **Tests.csv**: We can put anything here honestly. I just used one from Kaggle.

Having these datasets, we could start doing some machine learning. But, as we learned during class, it's better to first take a look at our data.

2.1 Data Analysis

Looking inside the Pokemon dataset we can see that we have some attributes associated with each pokemon. Unfortunately, we don't have a level attribute, but we can work with this nevertheless.

#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Atk	Sp. Def	Speed	Generation	Legendary
0 1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	False
1 2	Ivysaur	Grass	Poison	405	60	62	63	80	80	60	1	False
2 3	Venusaur	Grass	Poison	525	80	82	83	100	100	80	1	False
3 3	VenusaurMega Venusaur	Grass	Poison	625	80	100	123	122	120	80	1	False
4 4	Charmander	Fire	NaN	309	39	52	43	60	50	65	1	False
5 5	Charmeleon	Fire	NaN	405	58	64	58	80	65	80	1	False
6 6	Charizard	Fire	Flying	534	78	84	78	109	85	100	1	False
7 6	CharizardMega Charizard X	Fire	Dragon	634	78	130	111	130	85	100	1	False
8 6	CharizardMega Charizard Y	Fire	Flying	634	78	104	78	159	115	100	1	False
9 7	Squirtle	Water	NaN	314	44	48	65	50	64	43	1	False

The “#” column represents the actual id of the pokemon in the game, so this is the column that is meaningful to us to associate in the combats. A further analysis tells us that some pokemons don't have a second type, but we already expected this to happen. Some pokemons are purely Fire, Water, etc....

We can already see that in order to do some machine learning to this, we should scale our features. In other words, we should normalize them between 0 and 1.

Our combats.csv looks something like this:

1. 12# x 37#, 37#

What this tells us is that there was a combat between pokemon 12 and 37. And 37 won. Pretty straightforward right? For predicting the winner of a fight will be easy enough we have got everything we need.

But what about type prediction? We don't have any test dataset for this. But this is easily solvable if we pick from the main dataset a certain percentage for training and another for testing.




Let's view some fun stats about our dataset. Considering our combats dataset and our pokemon dataset, could we see which type of pokemon wins the most? Surely, we know this should be around 50% for each type, since the game tends to be balanced.

Win Percentage	
Type 1	
Flying	0.588832
Dragon	0.587902
Fire	0.578221
Electric	0.574344
Psychic	0.561503
Dark	0.554888
Ground	0.526908
Normal	0.526415
Ghost	0.502949
Grass	0.492613
Water	0.485826
Rock	0.474510
Steel	0.459225
Bug	0.430487
Ice	0.427134
Fighting	0.406426
Poison	0.405238
Fairy	0.369430

Well it seems that is not the case, nevertheless we can see that Flying type Pokemon win the most. Using domain knowledge, we know this could be true, since most of the really strong pokemon have the flying type.

Could we even get more information? For instance, what are the weakest pokemon? In other words, which pokemon won less combats. Yes, we just sort by win percentage which basically is just the number of fights won divided by the total number of fights.

This yields the following result, I called them the “losers”.

Pokedex Number	Name	images	Win Percentage
289	290 Silcoon		0.021739
189	190 Togepi		0.024590
638	639 Solosis		0.031008

So, if we build our machine learning model to predict the winner of a fight and we put these guys, most likely they will lose. Right? We will see later on.

2.2 Algorithms used

As with any machine learning project we should use more than one algorithm because sometimes, one algorithm could work better in a dataset but won't work in another dataset. We should do a comparison between the algorithms used.

The algorithms used were the following:

- Naive Bayes
- Logistic Regression
- Decision Tree (gini index)
- Random Forests

These algorithms were used for the two predictions (type and combat winner). They were not chosen at "random", these algorithms were chosen with a specific order, from "worse" to "best". If the theory is correct, we will have worse results with naïve bayes and better results with random forests.

After loading the dataset and normalizing all the features, and having a train set and a test set, the algorithms were applied and results were collected.

2.3 Building the model

As said before, we need the scale each attribute into a range between 0 and 1. This is called normalization and is often a good technique for all values to be on a common scale without distorting differences in the range of values.

For predicting the type of a pokemon, 95% of the dataset was taken into a train set, and the remaining 5% into a test set.

For predicting the winner of a combat, we have two datasets that we load on initialization that are our train and test set.

The features were normalized with the following formula:

$$z = \frac{x_i - \mu}{\sigma}$$

Where μ stands for the mean, and σ the standard deviation.

To be able to correctly predict the winner of a combat the features should be normalized in a different way. What should be done is the following:

- Get features from pokemon A and pokemon B.

- Get the differences between each feature.
- Normalize the differences

We will get a table like this:

	Winner	HP_diff	Attack_diff	Defense_diff	Sp.Attk_diff	Sp.Def_diff	Speed_diff
0	1.00000	-20.00000	-6.00000	10.00000	-15.00000	10.00000	-19.00000
1	1.00000	0.00000	-39.00000	-18.00000	18.00000	39.00000	0.00000
2	1.00000	-20.00000	-35.00000	10.00000	-45.00000	10.00000	0.00000
3	1.00000	-37.00000	-80.00000	-50.00000	10.00000	-50.00000	-28.00000
4	0.00000	50.00000	50.00000	-105.00000	105.00000	-160.00000	50.00000
5	0.00000	-10.00000	-3.00000	-100.00000	7.00000	-100.00000	5.00000
6	1.00000	-25.00000	0.00000	10.00000	-45.00000	-50.00000	-25.00000
7	1.00000	-80.00000	-20.00000	-70.00000	-65.00000	85.00000	-55.00000
8	1.00000	0.00000	12.00000	28.00000	-23.00000	-28.00000	-29.00000
9	1.00000	5.00000	-25.00000	-10.00000	5.00000	-20.00000	-5.00000
10	1.00000	10.00000	10.00000	-25.00000	-40.00000	-25.00000	-25.00000
11	0.00000	53.00000	99.00000	5.00000	17.00000	25.00000	78.00000
12	1.00000	-50.00000	-90.00000	-35.00000	-60.00000	-65.00000	-30.00000

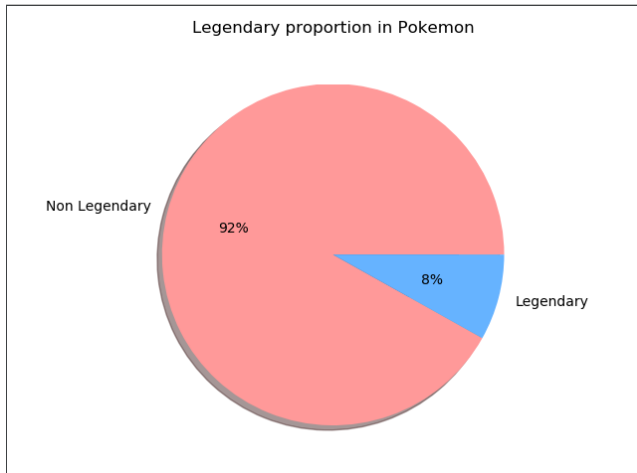
What this tells us is that for index 0, which essentially is the first combat that appears in *combats.csv*, the winner is the first pokemon, and the differences between each stat are that. Then we normalize the differences and get something like this.

	Winner	HP_diff	Attack_diff	Defense_diff	Sp.Attk_diff	Sp.Def_diff	Speed_diff
0	1.00000	0.45556	0.46857	0.51724	0.45042	0.52758	0.45075
1	1.00000	0.50000	0.37429	0.45287	0.54391	0.59712	0.50746
2	1.00000	0.45556	0.38571	0.51724	0.36544	0.52758	0.50746
3	1.00000	0.41778	0.25714	0.37931	0.52125	0.38369	0.42388
4	0.00000	0.61111	0.62857	0.25287	0.79037	0.11990	0.65672
5	0.00000	0.47778	0.47714	0.26437	0.51275	0.26379	0.52339
6	1.00000	0.44444	0.48571	0.51724	0.36544	0.38369	0.43284
7	1.00000	0.32222	0.42857	0.33333	0.30878	0.29976	0.34328
8	1.00000	0.50000	0.52000	0.55862	0.42776	0.43645	0.42090
9	1.00000	0.51111	0.41429	0.47126	0.50708	0.45564	0.49254
10	1.00000	0.52222	0.51429	0.43678	0.37960	0.44365	0.43284
11	0.00000	0.61778	0.76857	0.50575	0.54108	0.56355	0.74030
12	1.00000	0.38889	0.22857	0.41379	0.32295	0.34772	0.41791

Afterwards, using the amazing scikit-learn library that python provides, 4 models were created to apply the algorithms presented.

2.3.1 Problems with the dataset

If we compare the number of legendary pokemon to non-legendary pokemon in the dataset it will reveal a big problem. The vast majority of the pokemon are non-legendary. We know that legendary pokemon have much higher stats, thus being different than normal pokemon. This imbalance of data will not be helpful when training, specially when we use decision trees since it will cause overfitting of the data. Overfitting happens when there is a class that over represents the other classes, in turn making it harder for the model to learn.



To solve this issue, we can under-sample the majority class (non-legendary) and over-sample the minority class (legendary). To under-sample we simply reduce the total number of non-legendaries. A 70/30 train test split was applied to this.

To over-sample is a bit tricky, because we cannot simply generate more data. The data we generate must be similar to existing data. Fortunately there is a good algorithm for this called SMOTE.

This over-sampling technique was only applied to the training data to prevent the test data going “inside” SMOTE. This technique was only applied to the random forests algorithm

3 Obtained Results

The following results were obtained in python 3.7 using the scikit-learn library. The machine executing the code has an i5-8500 with 1.60 GHZ and 8GB RAM.

All the results were ran 30 times and the average was taken.

3.1 Type prediction

The following table shows the obtained accuracies of the trained models for predicting the type of a pokemon solely on its features (attributes).

Test size: 5%

Naïve Bayes	Logistic Regression	CART	Random Forests
0.09	0.14	0.15	0.12

Test size: 10%

Naïve Bayes	Logistic Regression	CART	Random Forests
0.19	0.27	0.09	0.23

Test size: 20%

Naïve Bayes	Logistic Regression	CART	Random Forests
0.16	0.17	0.1	0.18

Test size: 50%

Naïve Bayes	Logistic Regression	CART	Random Forests
0.18	0.18	0.13	0.23

3.2 Combat Prediction

The following table shows the obtained accuracies but this time of the trained models for predicting the winner of a combat.

Test size: 5%

Naïve Bayes	Logistic Regression	CART	Random Forests
0.86	0.89	0.91	0.94

SMOTE Technique: 0.97

Test size: 10%

Naïve Bayes	Logistic Regression	CART	Random Forests
0.81	0.88	0.91	0.95

SMOTE Technique: 0.86

Test size: 20%

Naïve Bayes	Logistic Regression	CART	Random Forests
0.8	0.89	0.91	0.94

SMOTE Technique: 0.92

4 Discussion of the obtained results

The results clearly show that predicting the type based on the attributes of a pokemon was a failure. The results are close to those of a random dice toss. The number of attributes is 6 and we got results similar to 1/6. For any real pokemon fan, we didn’t even need to do this experiment. It’s clearly a very difficult task even for the trained eye, because we have domain knowledge that every pokemon type has it’s attributes evenly distributed, that is, there isn’t a type that has high defense or a type with only very high attack.

But even then, logistic regression had “good” results with a test size of 0.1. The reason I don’t know, most likely statistical noise.

This goes to show that even if we have good data, and a fully working algorithm, sometimes we cannot predict things because they have no connection whatsoever. This is a classic example of that.

Now for predicting the winner we had really solid results. We were able to effectively with high accuracy predict the winner of a combat between two pokemons solely based on it's features.

But this was an easier problem (in my opinion) because we took the difference between each stat on each pokemon. Then we simply "compare" with another pokemon and check for the difference in each stat.

This is a good prediction of a winner. If we think about this, a pokemon with higher stats is better than a pokemon with lower stats, right? An attack of 500 is better than an attack of 300. But of course, it's the combination of those stats that produces a good prediction.

The results clearly show that random forests was the best algorithm (I already knew this beforehand). The reason is that random forests uses a bunch of decision trees and just averages them out, thus not relying on the result of just one decision tree. See attachment A for a real visualization of a tree from a random forest used in this project (with huge by the way).

The results that were collected using the SMOTE technique are inconclusive, but definitely better than naïve bayes or logistic regression. Probably even though there are more legends, there was not that big of a overfit on our data. This can be shown if we analyze the combats data set and verify that the combats legends participate in the same amount of combats (proportionally) than normal pokemon.

Type 1	Win Percentage
Flying	0.606250
Fighting	0.532406
Fairy	0.524606
Normal	0.522927
Dragon	0.511004
Grass	0.510268
Rock	0.507128
Ground	0.505919
Electric	0.501347
Bug	0.500171
Steel	0.498156
Fire	0.496779
Poison	0.493330
Water	0.489684
Psychic	0.482315
Ice	0.481470
Dark	0.473674
Ghost	0.470057

Let's see how the losers performed in the test set. Remember those guys? They had like 2% win rate. First let's see if the stronger types maintain the same.

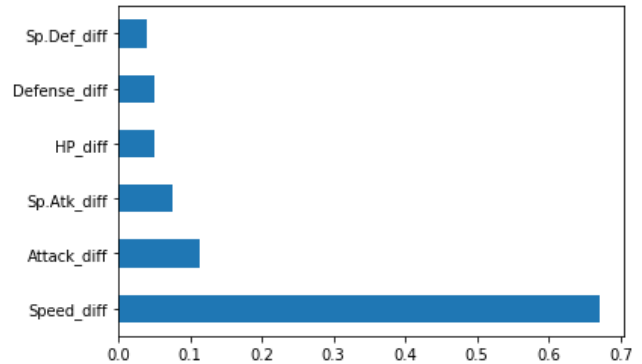
This is not a surprise, the stronger types in our training data were flying, and in the test data still flying.

What about the weaker guys?

Pokedex Number	Name	Total Fights	Win Percentage
638	639 Solosis	28	0.535714
189	190 Togepi	23	0.565217
289	290 Silcoon	20	0.600000

These results show that even though in our training set they had low win percentage, in our test set, have around a 50/50 chance on winning. The reason this happened, well probably they also fought weak pokemon. If we check the pokemons that they fought they were also pokemon with a low probability of winning.

Python has a cool feature in scikit that let's us see the most dominant features in a random forest. In the problem in hand these were the results.



This matches domain knowledge. Pokemon players know that speed is the single most important attribute in a pokemon, this lets it attack always first, sometimes twice in a row.

The final question is: Can we actually predict the winner of a combat between two pokemons? The answer is most likely no, because attributes are not the only thing that comes into play. We have much more things, like strategy, attack moves, and so on. But with the dataset that we have, yes we were able. But we cannot apply this in practice.

5 Acknowledgements

This work was done for the course of Machine Learning for the master's in computer science, supported by the University of Algarve in 16 of December of 2019.

REFERENCES

- [1] Pokemon datasets, <https://www.kaggle.com/ajisamudra/winner-in-pokemon-combat-prediction/data#Winner-Pokemon-Combat-Prediction>