Pokemon matchup analysis with different regression methods

March 2021

1 Introduction

In this report I describe my attempt to evaluate different matchups in the Pokemon video game.

Pokemon is a game where players collect different monsters called 'pokemon' and use them to battle other players. The Pokemon franchise is more than 20 years old and has many games. There is also a competitive Pokemon scene, and this analysis will use data from competitive pokemon battles.

This analysis and machine learning model should be useful in predicting the outcomes of competitive Pokemon battle situations. In particular, this should help when deciding which pokemon to choose in a situation opponent's pokemon is known, or when evaluating which pokemon 'counters' which.

The machine learning problem is explained in more detail in the Problem Formulation section. The Methods section describes the used dataset and regression methods. The different methods are compared in the Results section. The Conclusions section contains discussion on the usefulness of the ML model and some directions for future development.

2 Problem Formulation

The question I try to answer is: given pokemon A and pokemon B, how often will pokemon A beat pokemon B?

The datapoints of the problem are pairs of pokemon. A single pokemon is characterized by several statistics: hit points, attack, defense, special attack, special defense and speed. One datapoint has the statistics of pokemon A and pokemon B as features. In addition to numerical statistics, each pokemon has one or two elemental 'types'. These types have advantages and disadvantages against each other, for example water beats fire, fire beats grass, grass beats water, or flying is immune to ground. To account for type effectiveness with a numerical value, two type effectiveness multipliers are calculated for the pokemon pair (one that applies when A attacks B and the other when vice versa.)

The value we are interested in is how often does A beat B? This is the label of a datapoint and it is a percentage (between 0 and 1).

2.1 Summary of the problem

Label: probability that A wins. **Features**: type effectiveness separately when A attacks and when B attacks (numerical, either 0, 0.25, 0.5, 1, 2 or 4), HP, attack, defense, sp.atk, sp.def and speed of A and B separately (all numerical, ranging from 0 to 300)

3 Methods

3.1 Dataset

To construct the datapoints of this project, I combined usage data from the competitive Pokemon battling site Pokemmon Showdown (‘play.pokemonshowdown.com/‘) and a dataset with the stats of

all released pokemon. The Pokemon Showdown statistics can be found from 'www.smogon.com/stats/'. I used stats from the latest month 2021-02, the most popular game format gen8ou. The page contains lots of usage data from competitive battles played on the Showdown site. I only used the 'checks and counters' statistic, that tells for how often the given pokemon is KO'd or forced to switch out by various other pokemon. This statistic was included for all pairs of pokemon that had more than 20 or so recorded battles with each other. The pokemon types and stats dataset can be found from 'github.com/lgreski/pokemonData'. I combined the data from these sources to produce datapoints as described in the problem formulation.

The Showdown statistics included data for over 400 different pokemon and over 30000 unique pairs. This means I have over 30000 datapoints, which should be enough for the ML methods used. However, it should be noted that the 30000 datapoints are not as diverse as in some other case, since they are constructed by combining the 400 different pokemon.

The datapoints were split to training, validation and test sets such that 60

3.2 Linear regression model

The first method I used to fit the data was a simple linear regression model. The hypothesis space of a linear regression model is all functions of the form

$$\hat{y} = \beta_0 + \beta_1 x_1 + ... + \beta_n x_n$$

where $\hat{y}$ is the predicted label, x_i's are the values of different feature, β_0 is a constant term (called the intercept) and β_i's are coefficients for the different fetures. This corresponds to fitting a staright line to the data.

The regression coefficients β are determined by minimizing the mean squared error

$$MSE = \frac{1}{m}\sum_{i=1}^{m}(y_i - \hat{y}_i)^2$$

where y_i is the real label value and $\hat{y}_i$ is the predicted label value. This loss function was used for the regression fitting, because it is normally used in linear regression[2] and it was the default option in the used Python package.

Since the type effectiveness features are on a different scale than the stat features, I first standardized the features by subtracting the training set mean and dividing by the training set standard deviation.

To perform the linear regression, I used the `sklearn`[1] Python library and its `LinearRegression` class. The standardization step used `sklearn.preprocessing.StandardScaler`.

3.3 Decision tree model

The second method I applied the data was a decision tree regressor. A decision tree is a tree of true/false questions where a leaf node is then associated with a value for the label. A decision tree creates a discontinuous stepwise function to predict the labels.

My idea with using a decision tree was to better account for interactions between many the features. For example, if both pokemon can take the other out with one shot, the faster one will win because it gets to attack first. This means that when both pokemon can damage each other sufficiently, speed becomes the most important stat. On the other hand, a defensive pokemon doesn't need a high speed stat, because it can take a hit and then be ready to strike back. Also, when against a pokemon with high special attack, the normal defense stat is useless and the special defense stat is needed. A decision tree should be perfect for making these kinds of evaluations.

A decision tree is constructed by splitting nodes in a way that minimizes some loss function in the two child nodes. The loss function I used was the same mean squared error as in the linear regression. It was also the default in the used library. A decicion tree can be grown as deep as one wants, but as

the number of splits increases, the model will eventually start to overfit by making exceptions for each individual datapoint. To determine an appropriate stopping depth, I tried different tree depths from 3 to 15 and tested them against the validation set. The one with smallest MSE loss was chosen as the tree depth to be used.

There are many more methods to limit the size of the decision tree, but I didn't utilize them in this proect. I just used the default settings.

The `sklearn.tree.DecisionTreeRegressor` was used to do the tree regression.

3.4 Transformed features

I also tried to apply my domain knowledge of the game to try and construct better features to use in regression. I constructed a non-linear function of the features to use instead of the orignal fetaures.

However, this attempt didn't yield better results than the other methods, and the four page limit is too small to explain the idea in more detail and compare its results to the other methods. Because of that, I have left it out.

4 Results

4.1 Linear regression model

The learned parameters of the linear regression model are in the table below.

Feature	A	B
Type effect	0.079	-0.079
HP	0.008	-0.006
Attack	0.026	-0.020
Defense	0.000	0.013
Sp.Atk	-0.001	-0.004
Sp.Def	0.003	-0.002
Speed	0.006	-0.008

The intercept of the model was 0.5, which makes sense since all else being equal, A and B should have the same probability of victory. Since the situation is symmetric, the coefficients of B's features should be the same as the A's features but with opposite signs. This seems to roughly be the case, but the HP and defense stats in particular are very different. They might have been swayed by a few outlier cases.

From the coefficients we see that type effectiveness is the most important predictor of a matchup outcome, and it is much more important than the other features. Defense seems to be negative for the pokemon with high defense. This might be because usually the same pokemon doesn't have high defense and attack stats and it is specialized in one instead, and ususally attack is more useful.

The training mean squared error was 0.029. The validation mean squared error of the linear model when tested against the validation set was 0.029 and the mean absolute error was 0.14. The training and validation errors being the same means there was no overfitting.

4.2 Decision tree model

The smallest validation MSE was obtained when the tree depth was limited to 9.

In the decision tree model we notice similiar trends to the regression model. The first few branches split the data mainly on basis of the type effectiveness features, and the other stats are only used in the deeper layers.

The training MSE of the tree model was 0.023, while the validation MSE was 0.027. Even though the validation error slightly larger, the errors are still rather close, so the model doesn't seem to be overfitting badly. The validation mean absolute error was 0.13.

4.3 Comparison and selection

To choose between the four models, I compared their validation errors. I used the MSE loss, but also calculated the mean absolute error, which is

$$MAE = \frac{1}{m} \sum_{i=1}^{m} | \, y_i - \hat{y}_i \, |$$

and should give smaller weight to outliers. The MAE also has the advantage that it is on the same scale as the value being predicted, so it is easier to interpret.

The validation MSEs and MAEs of the models are given in the table below.

Method	MSE	MAE
Linear regression	0.029	0.14
Decision tree regression	0.027	0.13

Based on this, the differences aren't great, but decision tree regression with the original features seems to be best.

After finally testing the chosen decision tree model, with maximum depyh 9 and testing it against the test set, it got a testing MSE of 0.027 and a testing MAE of 0.13.

5 Conclusions

The finally chosen model had a MAE of 0.13, which indicates that the estimated win percentage given by the ML model is 13 percentage points on average. This is not very precise, but it should be sufficient to identify which pokemon clarly counter which. Considering that a lot of the complexity of a pokemon battle was left out of the ML problem, a much better precision might not be possible.

There are many improvements that could be tried in the future.

For one, the label to be predicted was a percentage that is bounded between 0 and 1. The linear regression doesn't understand this and might predict out of bounds values in some cases. A variation of logistic regression might make more sense.

The problem could be considered as a classification problem instead of a regression problem. The label would then be simply which pokemon wins, not the probability of winning. I mainly did the ananlysis as the regression version of the problem, because the dataset was in a format where the probabilities for each pokemon pair were given.

The tree model might be improved by better pruning or for example using a random forest method. My attempt at making more relevant features based on theory didn't work out, but it might be worthwhile to try it again with more effort.

There are bound to be outliers in the data, since the rules of the game allow for many exceptions. Using a loss function that is more robust to outliers might make sense already in the treining phase.

References

[1] Scikit Learn, https://scikit-learn.org/stable/index.html

[2] The course book, Chapter 2.3