Predicting Pokémon Type

Introduction

The challenge of this report is to use supervised learning models to predict the type of a Pokémon based on a selection of its other attributes. With the introduction of generation IX, there are now over a thousand Pokémon, and each possesses dozens of different attributes, some numerical, some categorical, that make every Pokémon unique. The popularity of the games has led to the creation of many websites, extensively documenting every single Pokémon. Ultimately, the franchise offers a rich source of data for practising machine learning algorithms.

The 18 types encountered in Pokémon can be thought of as the elements of the Pokémon universe. A Pokémon's type implies a lot of things about a Pokémon such as its habitat, the sorts of moves it can learn, its strengths and weaknesses relative to other types (for example, fire-type Pokémon are weak against water-type attacks), and its stats (for example, flying-type Pokémon tend to have high speed). Pokémon may have one or two types. For Pokémon with two types, one type is the primary type and the other is the secondary type. For example, Geodude's primary type is rock and its secondary type is ground (whereas Rhyhorn has these types switched around). This report will be concerned with predicting the primary type.

The dataset I will be using is an edited version of the one available here (Banik, 2017), which is available on Github. Banik's dataset was scraped from serebii.net, and contains 41 variables for 801 Pokémon, containing all Pokémon after the release of the Sun and Moon games. The variables contain, among other things, the base stats of each Pokémon, its height and weight, the effectiveness of each type against it, its abilities and various identifying information (such as its name and Pokédex number).

The two modelling approaches I have elected are a decision tree, and a random forest. A classification tree is the natural choice over an elastic net regression model, because the output variable is categorical. A random forest is an extension to the decision tree, as it outputs the aggregated response of many decision trees. While a neural network may be an interesting choice to model the data, and may even offer superior predictive performance. I have chosen a random forest because of its interpretability, as I am interested in learning which features of the dataset are the most important for predicting Pokémon type. One more benefit is that random forests are quicker to train than neural networks.

Data Cleaning and Exploratory Data Analysis

I began by opening Banik’s dataset in a tabular form and inspected the data manually. One of the most notable properties of the dataset is that there is a variable for the effectiveness of each of the 18 types against the Pokémon. I decided to remove these variables because, taken together, this information encodes the type combination of the Pokémon. Specifically, every type combination has a unique set of type effectivenesses; a fact which I verified using the Pokémon Database (*Pokémon dual-type charts* 2024). Therefore, given the type effectivenesses of a Pokémon, one can easily work out its type (but not which is the primary and secondary type in the case of a Pokémon with two types). Although including these variables would have drastically improved the predictive accuracy of the models, this is not exactly interesting.

I noticed some errors in the data that occur where Pokémon have more than one form. Certain species of Pokémon can appear in different forms and while these forms have the same name and Pokédex number, they can still have very different stats and types. The errors in the data occurred where a Pokémon took the values of some variables from one form but the values of other variables from another form. To address this I decided to add in new rows to the data for different forms (*Bulbapedia* 2024).

There was a variable which contained a list of all the possible abilities of a Pokémon. A Pokémon has one of up to three abilities and many of these combinations of abilities are unique to a Pokémon. Therefore, I split up this variable into three, one for each of the three possible abilities. This greatly decreased the number of categories for this variable (although there were still 173 abilities presented in the dataset).

Finally, I removed the Japanese name variable, simply because there was already the English name variable for the purposes of identifying each Pokémon. I also rearranged the order of the variables to a more logical order. The edited dataset, and the one on which I performed all my analysis in R, can be found on github (ScoDix, 2024). This dataset has 862 Pokémon, including different forms, and 24 variables.

After loading the data into R, I converted any character type variables into factors. It was also necessary to convert the “percentage_male” variable into a factor, because there was no sensible way to assign a number to Pokémon that fall into the “genderless” category.

I already had some intuition into which features would turn out to be the most important for predicting type. First and foremost is the ability. Many abilities are closely linked to a specific type. For example, all Pokémon with the “Overgrow” ability are grass-type, and all Pokémon with the “Blaze” ability are fire-type. Apart from this, there are some links between Pokémon stats and their types. For example, steel and rock-type Pokémon tend to have a high defence stat and dragon-type Pokémon tend to have high stat totals (the sum of the individual stats) while bug-type Pokémon often have low stat totals.

Due to the large number of types, it is not very useful to visualise them all simultaneously. Figure 1 shows the base stat totals for three particular types: Bug and dragon, for reasons mentioned earlier; and electric, as an example of a type with middling base stat totals. If we were given that a Pokémon was one of these types, we could be quite confident that it was dragon-type if its base stat total were 600 or more, and likewise we would say it is bug-type if its base stat total were 250 or less. This becomes far less obvious when we are dealing with all 18 types, however, as there are no obvious boundaries to draw.

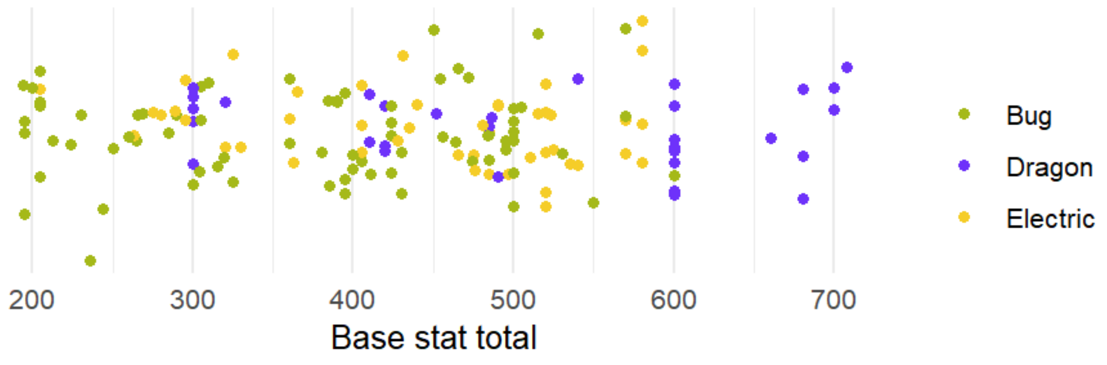


Figure 1: the base stat totals of three different types to show how base stat total influences type. However, if this included all the types then there would be no obvious boundaries. The vertical axis is only there to separate the data points.

Modelling Decision Trees

An overview

A decision tree performs a series of logical tests on the predictor variables in order to determine an estimate of the response variable. In this case, the decision tree is a classification tree, because the response variable is categorical; if it were a continuous variable, then it is called a regression tree. An example of such a decision was provided earlier, where we deem a Pokémon as being bug-type as opposed to dragon-type, if its base stat total is 250 or less. However, the task is to classify Pokémon into one of 18 categories and there are 20 predictor variables to work with. A decision tree is aptly named because it can be visualised like a tree; at each node the data is split into branches based on a logical test, and these subsets of data are split further into smaller and smaller branches until they reach the terminal nodes, or leaves, of the tree.

The CART algorithm compares potential splits using the Gini impurity index, deciding to split the data when this value is minimal. This process is repeated for each of the two subsets and then for each of the resulting four subsets and so on. In principle, this process could continue until all of the data were correctly classified, giving zero training error. However, this decision tree would likely perform poorly when given new data; it would have a very high testing or predictive error. The solution is to prune the tree.

Pruning the tree means cutting off branches of the tree to prevent overfitting and improve predictive accuracy. The algorithm I have used to generate the decision trees uses different values of the Complexity Parameter (CP) to generate trees of different depths (i.e. of different levels of pruning) and works out the cross-validated error for each. Next, the algorithm selects the value of CP which produces the lowest cross-validated error and retrains the decision tree on the entire dataset. Now the decision tree is ready to be tested on unseen data.

Results

My first decision tree was allowed to use all of the available variables (except for name, pokédex number, and classification because these are more or less unique identifiers for every Pokémon). Unsurprisingly, the final decision tree was dominated by the variables for abilities. The very first split in the tree checks whether the Pokémon's ability is "Overgrow" and then assigns it the grass type if this is true. If this is false, then the next split checks whether the Pokémon's ability is "Blaze" and assigns it the fire type if this is true. This carries on for many more splits, until different variables take over to sort the Pokémon deeper in the tree. Figure 2 shows some of the upper levels of this tree to demonstrate this.

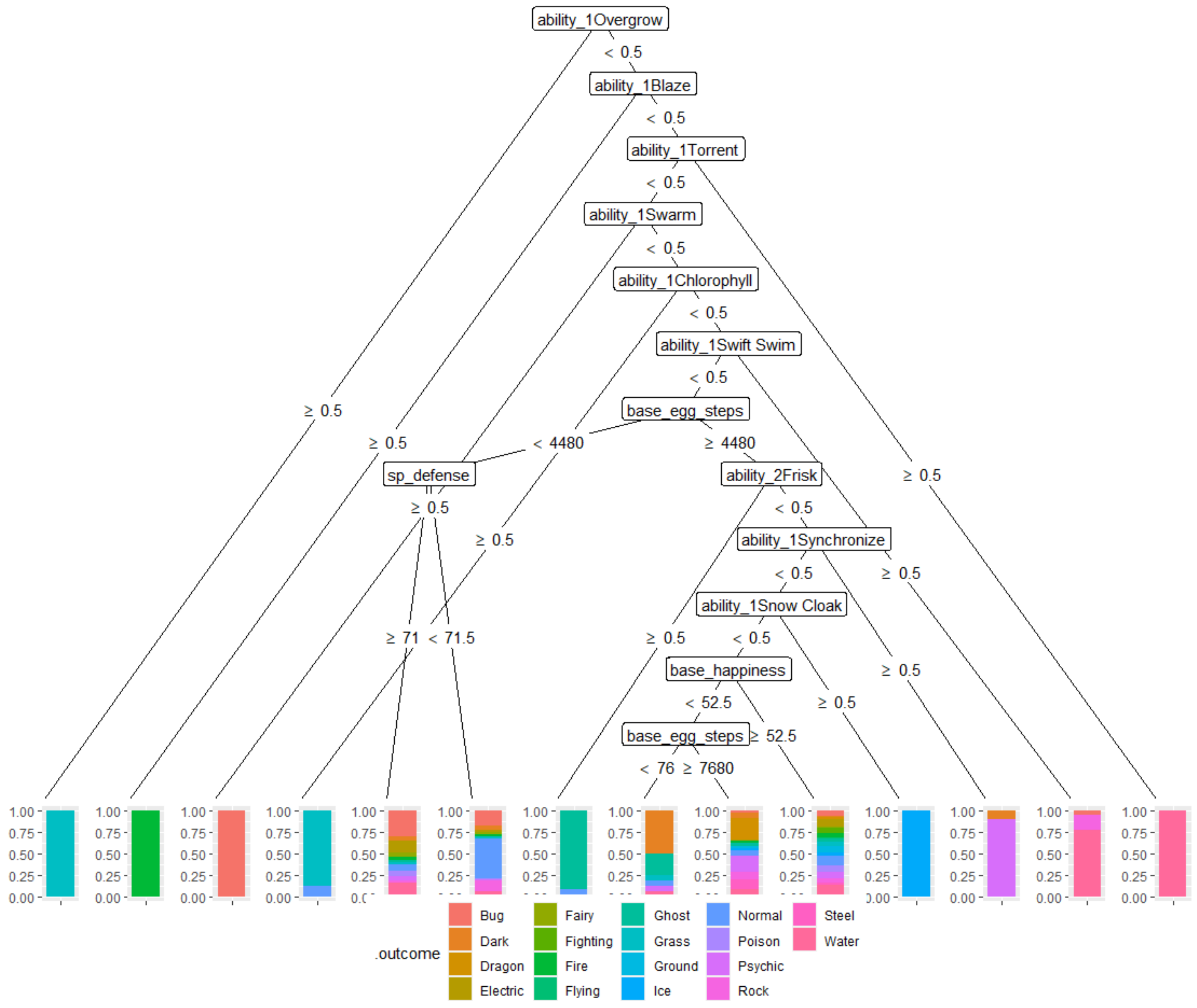


Figure 2: The first 12 levels of the decision tree. These levels are dominated by abilities which are specific to one type and so the decision tree quickly finds subsets which are of only one type.

I was curious to see how well a decision tree without access to the ability variables would perform in comparison. This tree was more surprising, as variables like "base_egg_steps", "experience_growth" and even, at deeper nodes, "generation" were used several times. The generation variable is particularly surprising to see because the game developers try to add a diverse mix of all types of Pokémon with each new generation, so I suspected that it would be of little use to a decision tree. Stats (e.g. HP, attack and speed) are also important features and they appear throughout the tree, as I expected.

When testing on unseen data, the accuracy score for the full tree was 37.2%. This is significantly better than a null model which assigns every Pokémon the most common type (water). Such a model would be correct 13.7% of the time if tested with the full dataset. Performing slightly worse is the tree without access to the ability variables, with an accuracy score of 33.1%.

However, this accuracy varies greatly between the types. Figure 3 shows the accuracy of both decision trees at classifying the Pokémon in the testing dataset of each type. Dark was the most accurately classified type. Fairy and ground were never classified correctly by the tree with all the variables, but were classified correctly about a third of the time by the tree without the ability variables.

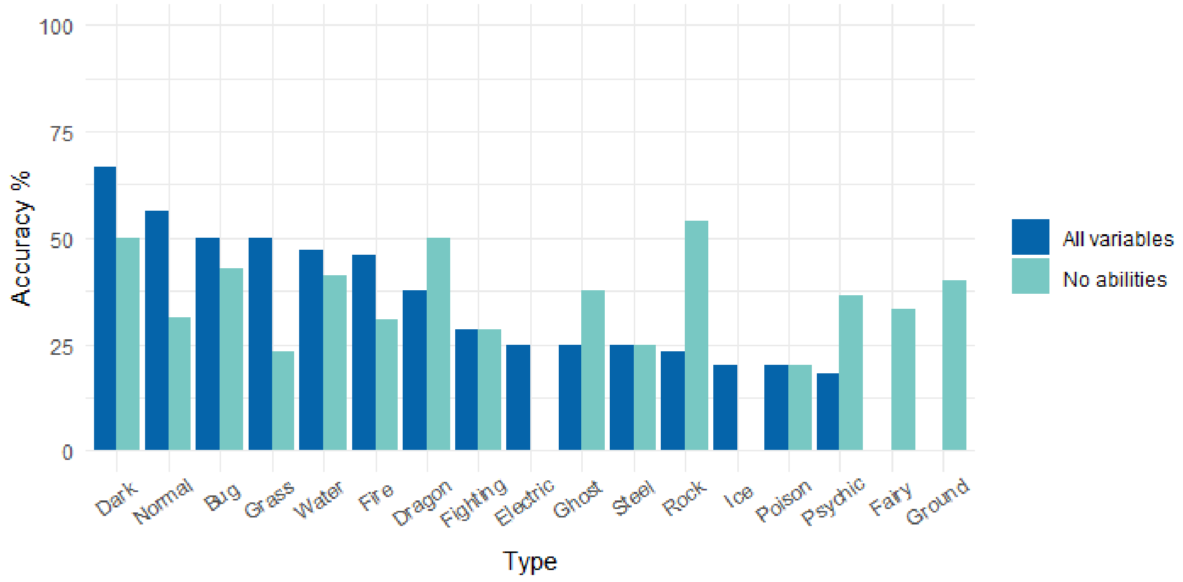


Figure 3: the accuracy of both decision trees for each type. There is great variation in the models' predictive accuracy although a few of the types do not appear very often in the testing dataset, leading to high variance.

Modelling a Random Forest

An overview

The random forest is an extension to the single decision tree. In short, it is a set of decision trees which “vote” on the category in a classification task (for a regression task, the mean of the predictions is taken). These decision trees vary in that for each split, a tree is only permitted to consider a random subset of the full set of features. Random forests also make use of “bagging”, short for bootstrap aggregating. This consists of taking random samples from the training dataset with replacement and training multiple trees on them. This means that every tree is trained on slightly different data and so captures different patterns in the data which helps to reduce variance and prevent overfitting, which decision trees are prone to. Bagging also allows for validation using the “out-of-bag” data, meaning the data points that were not chosen for the random sample, to provide an estimate of the model’s performance on unseen data.

Creating a random forest requires some tuning, because we are able to set the number of variables, m, which are considered at each split of a decision tree. One can also tune, for example, the maximum depth of the trees and the minimum number of observations required to split a node. I trained multiple random forests with different values of m, beginning with values around the square root of the total number of variables (Hastie et al., 2008, p. 592) and found that m = 30 gave the greatest validation accuracy. So I chose this random forest as my final model. Values of m around 30 gave similar validation accuracy, so this value of m might change based on the randomness involved in the training process.

Results

When tested on the same data as the decision trees, the random forest had an accuracy score of 58.1%. Figure 4 shows the top twenty most important variables in the random forest and their relative importance to the most important variable, which is special attack. We see that the stat variables are among the most important, and that “base_egg_steps” is the third most important, despite there being no obvious link between it and the response variable. Height and weight are also important, which is easier to explain, as many types lend themselves to certain sizes of Pokémon. Figure 5 shows how accurately the random forest can classify each type; where we see that the random forest varies greatly in its predictive accuracy from 87.5% for dragon types and 20% for ground types. Additionally, the decision tree was just as good at classifying the dark type as the random forest.

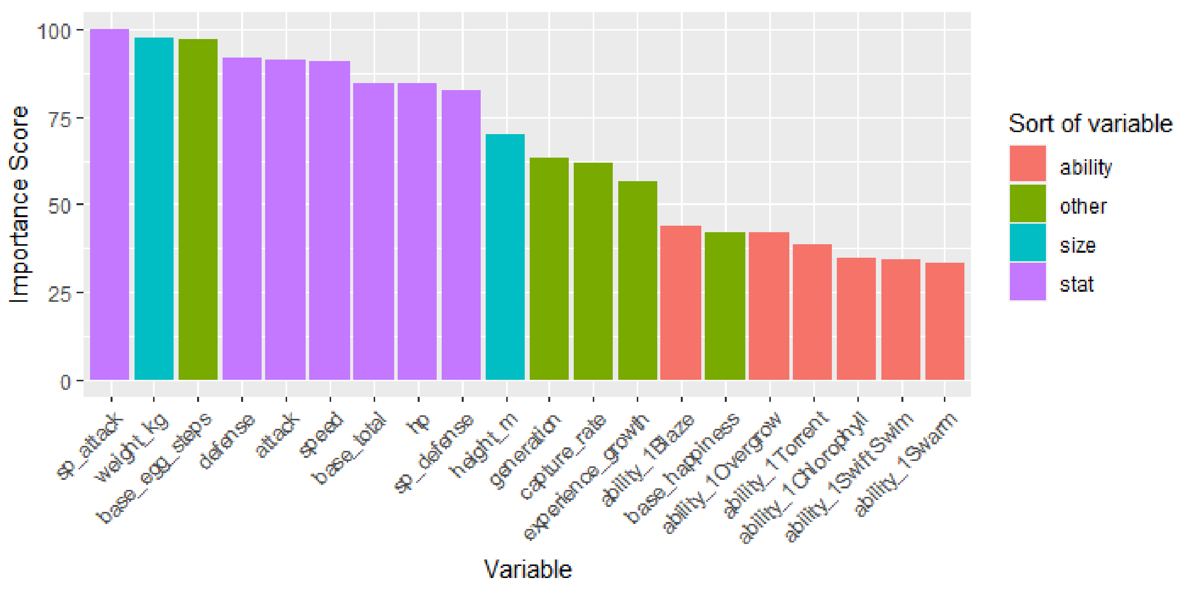
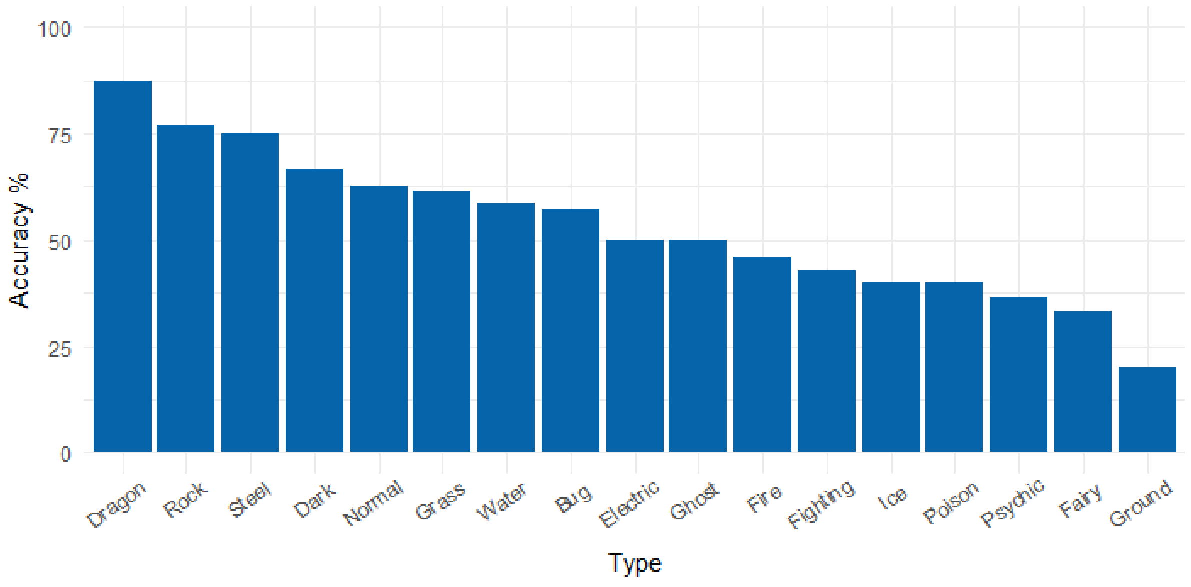


Figure 4 (above): the features selected as most important during the training of the random forest.

Figure 5 (below): accuracy of the random forest on the training dataset, grouped by type.



Model Comparison

For both the decision tree and the random forest, the exact same training and testing datasets were used. Both models were validated with 10-fold cross-validation and both used the same folds. In the case of the decision tree, the final model was the one with the complexity parameter that produced the highest cross-validated accuracy. In the case of the random forest, the final model was the one with the value of m which produced the highest cross-validated accuracy.

The random forest has significantly higher accuracy than the single decision tree. This is expected seeing as the random forest is an ensemble of many decision trees and all these decision trees capture different patterns in the data because of bagging and random feature selection. The random forest has an accuracy of 58.1% when tested on the testing dataset, while the single decision tree has an accuracy of 37.2%.

This improved accuracy comes with a considerable cost however, which is the time it takes to train a random forest compared to a single decision tree. There were 20 variables in the dataset, yet some of these were categorical variables containing many categories (e.g. there were 173 different categories for each ability). Altogether, there were 474 variables to choose from, and so training multiple random forests with different values of the hyperparameter (the number of variables to consider at each node of each tree) led to training times of up to 15 minutes. Another tradeoff is the difficulty in interpreting a random forest, because it doesn't have the same simple tree representation as a single decision tree.

Results and Conclusion

The results show how extending the concept of a decision tree to a random forest can greatly improve predictive accuracy. A single decision tree can vary greatly depending on the data it is trained on, and is prone to overfitting; problems that are mitigated by a random forest.

With an accuracy score of 58%, the random forest provides a good model for predicting Pokémon type, considering that there are 18 types to choose from. This is not an easy classification task, because the designers of Pokémon need to create new and interesting ones which often go against the grain.

Now I will consider possible further steps. First, aggregating the abilities into a type. This would drastically reduce the dimensionality of the data, speed up training time and make the ability variable less sparse (many abilities apply to very few Pokémon). Second, expanding the dataset with Pokémon from newer generations and adding more variables: such as habitat, egg group, shape and colour. Third, training a neural network and comparing its performance with the random forest. Finally, using natural language processing to predict type based on text-based data. For example, many Pokémon have names that convey their type. For example, Squirtle is a water type, and its name is a portmanteau of the words “squirt” and “turtle”.

In conclusion, I have shown that it is indeed possible to predict the type of a Pokémon, to the extent that it is correct most of the time, using only a carefully tuned decision procedure as this is what underpins classification trees and random forests.

Bibliography

Banik, R. (2017, September 29). *The Complete Pokemon Dataset*. Kaggle. https://www.kaggle.com/datasets/rounakbanik/pokemon/data

Bulbapedia. (2024) Bulbapedia, the community-driven Pokémon encyclopedia. https://bulbapedia.bulbagarden.net/wiki/Main_Page

Hastie, T., Tibshirani, R., & Friedman, J. (2008). *The elements of Statistical Learning (second edition)*. Springer.

Pokémon dual-type charts. (2024) Pokémon Database. https://pokemondb.net/type/dual

ScoDix. (2024). *Pokémon Dataset*. GitHub. https://raw.githubusercontent.com/ScoDix/Pokemon/main/pokemon.csv