Pokemon Team Predictions

Which characteristics of a Pokemon give the best winning rate and can they be used for prediction?

ABSTRACT

Within this project, we will attempt to determine the win percentage between Pokemon. To achieve this, we will be gathering our multiple Pokemon datasets from Kaggle. We will preprocess the dataset such that each observation will represent a particular Pokemon and its characteristics. After data preprocessing and exploratory data analysis, we will use machine learning algorithms to predict the win percentages between two Pokémon given a specific set of characteristics of each Pokemon. These algorithms include Random Forest, Support Vector Machine (SVM), and Extreme Gradient Boost (XGBoost). By using these machine learning models, our team found that Random Forest Regressor had the best performance with an R-squared of 93.57%, an MAE of 0.0486 and 0.0039 for MSE. We also found that the top 5 features that were deemed most important for prediction are the same within each of the models. The features include Speed, attack, base_total, hp, and total fights.

CCS CONCEPTS

• Computing methodologies~Machine learning~Machine learning algorithms~Regularization •Computing methodologies~Modeling and simulation~Model development and analysis~Modeling methodologies •Computing methodologies~Machine learning~Machine learning approaches~Classification and regression trees •Computing methodologies~Machine learning~Cross-validation

KEYWORDS

Support Vector Machine, Random Forest, Extreme Gradient Boosting, Regularization, Exploratory Data Analysis, Hyperparameter Optimization, Pokemon, R-Squared, Mean Squared Error, Mean Absolute Error, Visualization, Feature Importance

1 Introduction

In this section, we will be giving a few terms that are needed to understand the concepts behind the franchise Pokemon as well as describing the data sets used in this paper.

1.1 Background

Pokemon is a franchise that includes movies, TV shows, and games. The main premise within the games is that people (also referred to as Trainers) go on a journey to capture Pokemon (animals with special abilities) in pokeballs (a ball that houses a Pokemon), trains them (increasing their stats), then battles with a team of Pokemon that is one-on-one (one Pokemon at a time for each Trainer) until one Trainer beats the whole team of the other Trainer. We will be focusing more on the rules of the battles that take place within the games rather than the ones shown in the movies and TV shows as the rules not in the game are a little too complex to model (personalities of Pokemon, determination of Pokemon, relationship of Pokemon to Trainer, etc.)[9]. However, there are some features in the data sets that can be used as a reference to those characteristics from the movies and TV shows. With this background, we will be analyzing within the one-on-one battles which Pokemon would likely win given their abilities and win percentage from the number of battles they won.

1.2 Data Sets

In order to address our research question we need data sets that have the characteristics of the Pokemon (stats) and how many times the Pokemon won in battles. From the website Kaggle, we were able to find 3 data sets that gave us all the information we needed to address our question of interest. One of them had a complete set of features that described all the Pokemon that exist in the games, shows, and movies. Another one of them had the combat information that gave how many times a Pokemon would win against one other Pokemon (a typical type of battle within the games). The third and final dataset we used was an ID dataset that linked the combat data with the stats data. Table 1 describes the overview of the datasets.

Data set	# of Obs	# of Feat.	Website
Stats	801	41	Link
Combat	50000	3	Link
ID	800	12	Link

Table 1: Data Set Information

1.2 Data Description

To expand the understanding of the data sets and why we used them as well as how we used them, we will define some important features used in data to clarify any

confusion. A Pokemon type (type1 and type2 in the data set) is a set of elemental characteristics as well as physical that determines the strength and weaknesses of that Pokemon. For example, a fire type Pokemon is weak against a water type Pokemon, but strong against a grass type Pokemon[8]. Generation is the season in the show/ era of the games that the Pokemon appears in. Legendary Pokemon (is_legendary) are Pokemon that are extremely rare and hard to capture. They are well known to have legends and are sometimes worshiped within their respective regions (or continents/ areas in the franchise). Hit Points (HP) is the health of the Pokemon in battle and gives the information on how many attacks/ hits it can take in a battle. Special Defense goes along with Special Attack (sp_defense and sp_attack respectively) as when a Pokemon uses a Special Attack move the Special Defense tells how much a Pokemon can defend itself before it takes damage on its HP (similar to Attack with Defense). Basic stats include the following: attack, defense, special attack, special defense, and HP[9]. Egg steps (base_egg_steps) are how many steps it takes to hatch an egg (which is how all Pokemon are born). Experience growth (experice_growth), also known as exp, is the amount of points (gained from winning a battle) a Pokemon needs to level up and become more powerful.

1.3 Plan of Action

The main premise behind our research is to understand the relationship between a Pokemon's type, attack and defense stats, special attack and defense stats, speed, HP, egg steps, happiness, capture rate, experient growth, which generation the Pokemon is from, and finally if the Pokemon is legendary with the probability of a Pokemon winning a battle (win percentage). With 3 different algorithms (SVM, Random Forest, and XGBoost) we hope to model these relationships and see which model would bring us the best set of stats a Pokemon can have to win a battle as well as being able to predict which Pokemon would win based on the variables of interest. We will be using specific performance metrics in order to evaluate and compare our models in which we will then choose the best model that could give us the best prediction on which Pokemon would win given just their stats.

3 Literature Survey

3.1 Support Vector Machine (SVM)

A classification algorithm that has received considerable attention is Support Vector Machines (SVMs). It has its roots in statistical learning theory and has shown promising empirical results in many practical applications like handwritten digit recognition and text categorization. It also works well with high dimensional data and avoids the curse of dimensionality problems. It represents the decision boundary using a subset of the training examples, known as the support vectors (hence the name support vector machine). SVMs are also called Maximum margin classifiers. Decision boundaries with large margins tend to have better generalization errors. Large margin implies there is a high confidence in classification. Intuitively, if the margin is small, then any slight perturbation to the decision boundary can have quite a significant impact on its classification. Classifiers that produce decision boundaries with small margins are therefore more susceptible to overfitting and tend to generalize poorly on previously unseen examples. The decision boundary is referred to as a hyperplane SVMs are further classified into two types, linear and non-linear SVMs.

A linear SVM is a classifier that searches for a hyperplane with the largest margin. It is also called maximum margin classifiers. Consider a binary classification problem with N training examples. Each example is denoted by a tuple (xi, yi) (i = 1,2,….N). Here xi= (xi1,xi2, ...xid) corresponds to the attribute set for the ith training example. Let the class label be denoted by yi= -1 for the negative class and yi= +1 for the positive class. The decision boundary of a linear classifier can be written in the following form:

$$w.x + b = 0$$

Here, w and b are the parameters of the model w is a vector of the same dimension as the number of dimensions of the data.

Now, let's discuss a Non-Linear SVM. Consider the dataset in the adjoining figure. It is similar to the previous dataset, but has two new examples P and Q. The decision boundary B1 misclassified the new examples, but B2 classifies them correctly. But, this does not mean that B2 is a better decision boundary because the new examples may be noisy training samples. B1 is still preferred over B2 as it has a wider margin and is less susceptible to overfitting. The SVM formulation described previously constructs decision boundaries that are mistake-free. This is called the hard-margin formulation. For the non-separable case, we modify the formulation to learn a decision boundary that is tolerable to small training errors. This is known as the soft-margin formulation. Soft margin SVM considers the trade-off between the width of the margin and the number of training errors committed by the linear decision boundary. Previously, we had the following inequality constraints for the hard-margin formulation

$$w.xi + b \geq 1, \text{ if } yi = 1$$
$$w.xi + b \leq -1, \text{ if } yi = -1$$

We now relax the constraints to accommodate non-linearly separable data. This can be done by introducing positive-valued slack variables into the constraints of the optimization problem:

$$w.xi + b \geq 1 - \varepsilon i \text{ if } yi = 1, \ \forall i, \varepsilon i \geq 0$$
$$w.xi + b \leq -1 + \varepsilon i \text{ if } yi = -1, \ \forall i, \varepsilon i \geq 0$$

3.2 Random Forest

Random forests are a combination of tree predictors such that each tree depends on the values of a random vector sampled independently and with the same distribution for all trees in the forest. The generalization error of a forest of tree classifiers depends on the strength of the individual trees in the forest and the correlation between them. Using a random selection of features to split each node yields error rates that compare favorably to Adaboost (Freund and Schapire[1996]), but are more robust with respect to noise. Internal estimates monitor error, strength, and correlation and these are used to show the response to increasing the number of features used in the splitting.

For random forests, an upper bound can be derived for the generalization error in terms of two parameters that are measures of how accurate the individual classifiers are and of the dependence between them. The interplay between these two gives the foundation for understanding the workings of random forests. To improve accuracy, the randomness injected has to minimize the correlation ρ while maintaining strength. The forests studied here consist of using randomly selected inputs or combinations of inputs at each node to grow each tree. The resulting forests give accuracy that compare favorably with Adaboost. This class of procedures has desirable characteristics: i) Its accuracy is as good as Adaboost and sometimes better. ii) It's relatively robust to outliers and noise. iii) It's faster than bagging or boosting. iv) It gives useful internal estimates of error, strength, correlation and variable importance. v) It's simple and easily parallelized.

Random forests are an effective tool in prediction. Because of the Law of Large Numbers they do not overfit. Injecting the right kind of randomness makes them accurate classifiers and regressors. Furthermore, the framework in terms of strength of the individual predictors and their correlations gives insight into the ability of the random forest to predict. Using out-of-bag estimation makes concrete the otherwise theoretical values of strength and correlation.

Forests give results competitive with boosting and adaptive bagging, yet do not progressively change the training set. Their accuracy indicates that they act to reduce bias. The mechanism for this is not obvious. Random forests may also be viewed as a Bayesian procedure. Although I doubt that this is a fruitful line of exploration, if it could explain the bias reduction, I might become more of a Bayesian.

The simplest random forest with random features is formed by selecting at random, at each node, a small group of input variables to split on. Grow the tree using CART methodology to maximum size and do not prune. Denote this procedure by Forest-RI. The size F of the group is fixed. Two values of F were tried. The first used only one randomly selected variable, i.e., F=1. The second took F to be the first integer less than log2M+1, where M is the number of inputs.

3.3 Extreme Gradient Boosting (XGBoost)

Ensemble methods use multiple (weak learners) models to create an optimal model. Boosting is where individuals' models are created iteratively. Each model's output will influence the following model. If any errors arise in the creation of the model, weights are given to the incorrect data points, and the final prediction is an average of the ensemble [5]. Gradient Boosting is similar but attempts to minimize the expected value of the loss function [4]. Built upon the Gradient Boost algorithm, Extreme Gradient Boost (XGBoost) has quickly gained popularity since its initial release in early 2014. This is partly due to the number of domains in; which XGBoost can be implemented, such as customer behavior prediction, malware classification, and product categorization. Some advantages of XGBoost are the ability to handle sparse data, regularization, speed and flexibility, and performance [2]. XGBoost prevents overfitting by using regularization, shrinkage (learning rate), and column subsampling. Regularization modifies the loss function to reduce variance by increasing a small amount of bias in the training stage. There are some benefits to the type of regularization used in a model. Ridge, or L1, regularization is used when some features are irrelevant to the model. L1 regularization encourages the algorithm to remove unnecessary features by setting their coefficients to zero [7]. Using L1 regularization in XGBoost is beneficial since we will have some Pokemon features that are irrelevant to predicting win percentages (discussed in the Exploratory Data Analysis section). Some of these features are base_happiness, base_egg_steps, experience_growth, and capture_rate.In contrast, L2, or Lasso, regularization shrinks the coefficients to zero but keeps all the features in the model. Shrinkage, or the learning rate, increases or decreases a particular model's influence in the ensemble method. The smaller the value, the less effect a specific model has on the ensemble [4] [6]. Column subsampling randomly selects a subset of features to build a tree. This

can speed up the tree-creation process since it requires less computation [2].

Gonzalo Martinez-Munoz has comprehensively compared three machine learning algorithms, including XBGoost, against 28 different datasets from various fields of applications in the UCI repository. His group concluded that an XGBoost model with default parameters was the least successful method. They also found that a "parameter search is necessary to create accurate models based on gradient boosting" [1]. We can play with some of the parameters for our dataset to help us create a more optimum XGBoost model. Some of the parameters include the learning_rate (learning rate), gamma (minimum loss reduction), max_depth (depth of the tree), lambda (L2 regularization), and alpha (L1 regularization) [3].

3.4 Performance Metrics

In order to compare and evaluate our models, we need performance metrics. The metrics that we have chosen for this project are the coefficient of determination (R-squared), mean absolute error (MAE), and mean squared error (MSE). The R-squared is calculated, as seen in Equation 1, given the sum of squares of the regression line and then the sum of squares of the mean line (a kind of baseline model for comparison)[12].

$$R^2 = 1 - \frac{\text{sum squared regression (SSR)}}{\text{total sum of squares (SST)}}, \\ = 1 - \frac{\sum(y_i - \hat{y}_i)^2}{\sum(y_i - \bar{y})^2}.$$

Equation 1: Coefficient of Determination (R-Squared)

R-Squared is a type of measurement for the goodness of fit of the model. It gives a good idea on the proportion of predictions is explained by the feature variables. The typical range of values for this metric is 0 to 1, with 0 being the worst and 1 being the best fit model. Though this is a good measurement in and of itself for how well a model performs, we still wanted to have other measures for comparison.

$$MAE = \frac{1}{n}\sum_{i=1}^{n}\left|y_1 - \hat{y}_i\right|^2$$

Equation 2: Mean Absolute Error (MAE)

The MAE is shown in Equation 2. For this kind of metric, it gives a basic reading on the average error within the dataset. This is a more clear and understandable way to measure the accuracy of a model even though it is a more basic calculation[13]. It can be more easily explained to a client when trying to describe the accuracy of the model. The values for this metric can range from 0 to infinity wherein the smaller the value the better the model is.

$$\text{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

Equation 3: Mean Squared Error (MSE)

The last we will use (as a way to thoroughly examine and compare our models) we have the MSE. As seen in Equation 3, it is the measure of the squared difference between the actual and predicted values of the model. It gives a sense of how close the predicted values are to the actual values[14]. Compared to the other 2 metrics we have here, this is the most sensitive to outliers, thus this could give more intel on how our data is distributed within the predictions in addition to giving another simple method of error to explain to the client. The range for this metric is similar to that of MAE wherein it can be values from 0 to infinity and the smaller the value the better the model is.

3.5 Hyperparameter Optimization

Within this project, we will also be exploring the topic on hyperparameter optimization. Within machine learning models, there are many different parameters that can be edited or adjusted in order to make the model learn in different ways or in which ways it should assign weights on the features. Specifically, a hyperparameter is a specific kind of parameter in which its value determines how the learning process is done[15]. Each model that we used has different sets of parameters that can be tuned in order to be optimized for the data set we will be using, thus we need to find an approach that would help us find what each of the hyperparameter values should be for each model.

There were 2 ways in which we utilized finding the optimizing hyperparameters for each of our models. We used the Random Grid Search approach as well as the regular Grid Search approach. We wanted to have a comparison on different types of hyperparameter tuning in order to cover more information on how well this kind of optimization works with our dataset. We will also compare these tuned models with a baseline model that contains the default values of the algorithms we chose from the packages provided in Python.

These approaches are a type of hyperparameter optimization technique wherein you define a set of hyperparameters of a certain model you wish to chose (which is also called a parameter space or grid) in which there is a set of those hyperparameters that would produce

the best results to fit a model for the data set used. Specifically, the Grid Search algorithm places them in a grid-like structure (as the name suggests) and finds all possible combinations given from the parameter space until the algorithm finds the model with the best performance, where the comparison is done by a, k, specified amount of K-Fold Cross Validation. The main difference for the Random Grid Search approach is the random component of this techniques wherein instead of all possible combinations from the parameter space, there is a random sample of combinations that will be tested and then compared (in the similar K-fold Cross Validation way) to find which combination (from that sample) is the best performing set of parameters. The main benefit of Random Grid Search is that it takes much less time to run as just Grid Search does a type of exhaustive search from all possible combinations within the parameter space. [15][16]

4 Exploratory Data Analysis

After cleaning the data, we are left with these features: pokedex_number, Number, name, type1, type2, attack, defense, sp_attack, sp_defense, speed, hp, base_total, base_egg_steps, base_happiness, capture_rate, experience_growth, generation, is_legendary, and Total Fights; with our target variable being Win Percentage. The stats that are not typically used to determine who wins in a battle in the game, but can be used to reference stats from the movies and TV shows (as referenced before) are: base_happiness, base_egg_steps, experience_growth, and capture_rate[9].

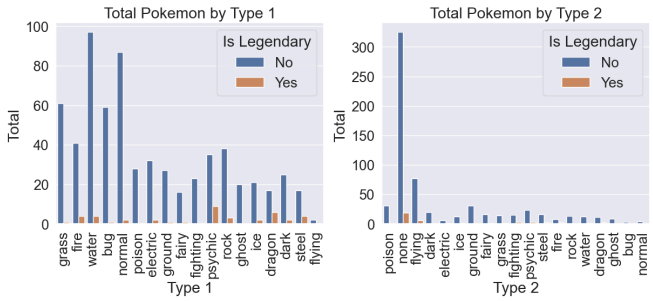

Figure 1: Total count of pokemon by Type and Legendary

Comparing all the categorical variables, the types water, normal and grass seem to be the most common type 1 while flying, ground and poison are the most common for type 2 (from Figure 1). The least common type of Type 1 is flying, fairy, and dragon. The least common type of Type 2 is normal, bug, and electric. For Pokemon being Legendary, we can see that they mostly only have one type and are psychic. With this information we can see what types of Pokemon we can use to counter the most probable type that will appear. In this case, the most probable type of a Pokemon that will be in battle is water, therefore we probably would not recommend using a fire type Pokemon[8].

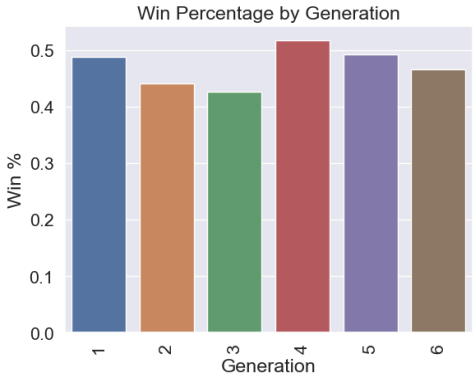

Figure 2: Win Percentage by Generation

Taking a look at how the Generation of Pokemon affect the chance of winning, in Figure 2, they do not seem to really differ in win percentages between each Generation. Generation 4 is the one with the highest probability of winning (0.51) but only by a small margin. Then, the Generation with the lowest probability of winning is Generation 3 (.42), again by a small margin (about .1 in difference between Generation 3 and 4).

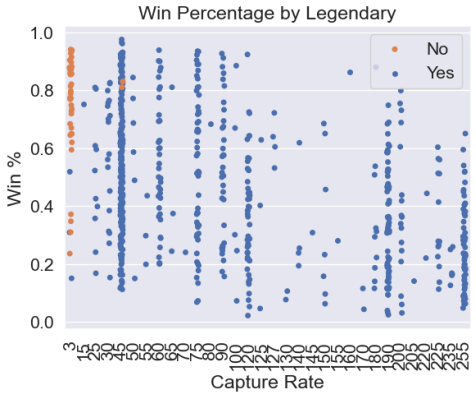

Figure 3: Win Percentage by Capture Rate and Legendary

On the other hand, when it comes to choosing whether or not a Pokemon should be Legendary, from Figure 3 most of the Legendary Pokemon have a win rate of 60% or above (with only about 4 Legendary Pokemon below that threshold) meaning that it is a safe bet to have that kind of Pokemon on a team. However, there is a cost of having a Legendary on a team. Since they are so powerful, their capture rate (or how easy it is to capture the Pokemon) is very low. This means that they are hard to find and take up a great deal of time and resources (ex. Pokeballs, money, Pokemon's health, etc.) to catch[9]. Therefore, one must think of the tradeoffs between taking a chance at maybe

finding a Legendary and then taking another chance to see if it can be caught when found (if the Trainer fails to catch, if the Pokemon runs away, or if the Pokemon faints, then the Trainer will have to find that Pokemon again and try to catch it again) or just finding a different stat to focus on when adding a Pokemon to their team.

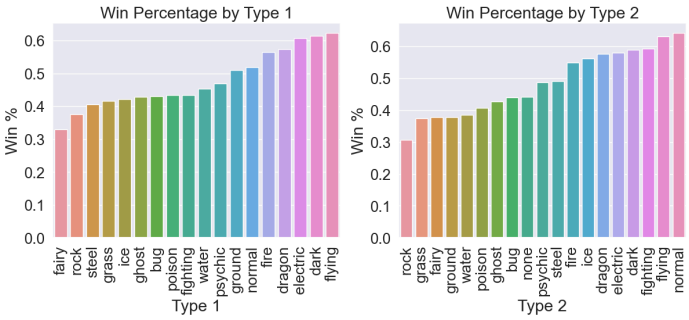

Figure 4: Win Percentage by Type1 and Type2 (respectively)

When looking at Figure 4, the relationship of types to the win percentage seem to help define which types could win. The highest winning type 1 are electric, dark, and flying (which wins against water, bug, and grass[8] which are the most common Type 1 Pokemon). In the lowest winning type 1 Pokemon (fairy, rock, and steel), there does not seem to be an obvious relation to the common types of Pokemon. The highest type 2 are fighting, flying, and normal (which all of them have different win rates for different types that show no real pattern with the most common types). The lowest for the Type 2 of Pokemon are rock, grass, and fairy, thus making it so that fairy and rock does not seem to have the best odds in winning a Pokemon battle. Additionally, looking at Type 1 seems to have significant information on how to pick which Pokemon to use in a battle that has the highest probability of winning.

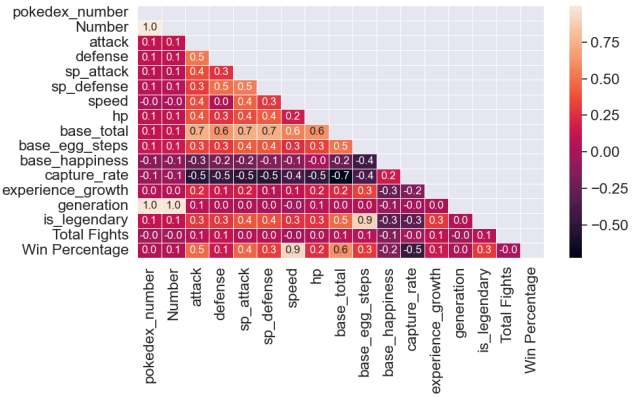

Figure 5: Correlation Matrix of variables of interest

Given the correlation matrix in Figure 5, we can see that speed, attack, and special attack (0.92, 0.46, 0.44 respectively) seem to be the top contributors to having a higher Win Percentage than that of the other stats. The base total (0.62) is also highly correlated with Win percentage, but that makes sense as the basic stats are highly correlated to base total. All the stats just mentioned seem to have a positive relation with Win percentage. As seen in the findings from Figure 2 and Figure 3, the correlation matrix supports those ideas in that the Legendary feature does have a correlation with the Win percentage while Generation does not. The correlation matrix also shows the idea behind the capture rate given they have a negative relationship (meaning the harder it is to capture a Pokemon, for example a Legendary Pokemon, the higher chance that Pokemon has to win in a battle, a tradeoff given the difficulty of capture).

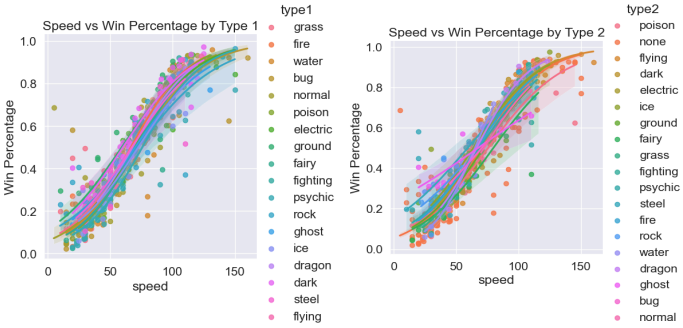

Figure 6: Win Percentage vs Speed by Type 1na Type 2 (respectively)

To dive deeper into the relationships between the highly correlated numerical variables, referencing Figure 6, we can see that the types of Pokemon are randomly scattered and that the Speed variable has a positive relation even with the subsetting of types. Speed seems to have a specific pattern which resembles a logistic curve, but with no overall relation of a Pokemon's type to Speed or Win Percentage.

Overall, through the exploratory data analysis, these are the main trends and patterns seen in the data:

1. The types water, normal and grass seem to be the most common types of type 1 and flying, ground and poison are the most common types for type 2. This will give a good idea on predicting what a Trainer would choose in a battle thus giving way to figuring out which Pokemon types we should have on our team and using more resources on those types of Pokemon.
2. The Generation of a Pokemon does not seem to have a major effect on the probability of a Pokemon winning.
3. Having a Legendary Pokemon does seem to increase the chance at winning a battle, however it does have a

cost (as seen with the capture rate) of more time and resources.

4. The stats that should be the main focus on a team is a Pokemon's attack, special attack, and speed.

5 Results

To ensure that the models would have the certain criteria would be able to have the best prediction power for this data set, we decided to do some hyperparameter tuning using RandomGrid Search and Grid Search from the Sklearn package to find the best parameters that would allow the algorithms to train on the data provided in the best way possible. The chosen parameters will be defined within each of the different algorithms below.

5.1 SVM

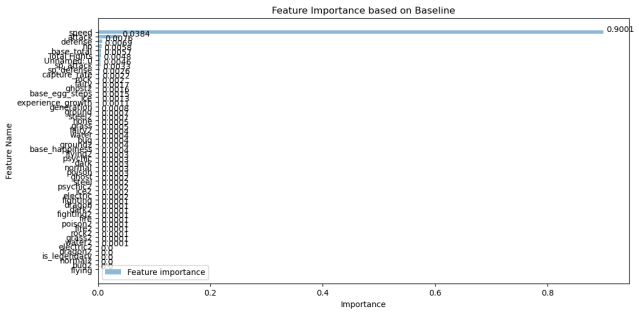

Figure 7: Feature Importance of Baseline model

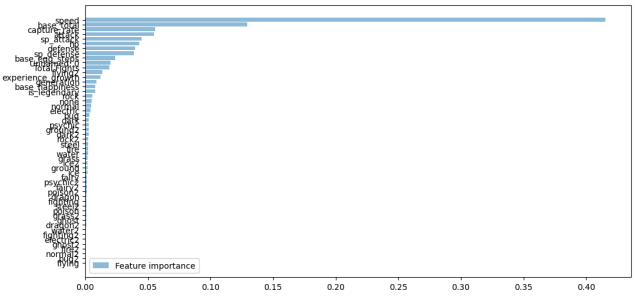

Figure 8: Feature Importance of Random Search model

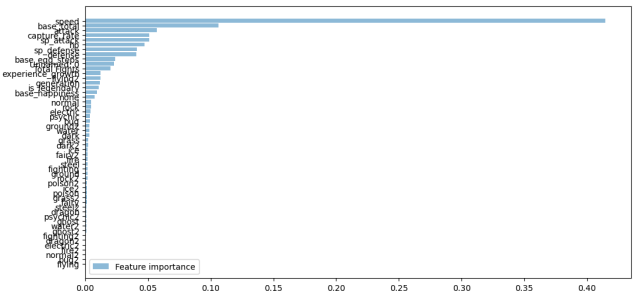

Figure 9: Feature Importance of Random Search model

The SVM (Support Vector Machines) had a lot of parameter to find the final conclusion. The SVM found the following values for the parameters: Best parameters: {'C': 10, 'coef0': 1, 'degree': 4, 'gamma': 0.001, 'kernel': 'poly'}. The Support Vector Machine (SVM) algorithm also has hyperparameters that need to be tuned to achieve the best performance. One of the most important hyperparameters is the kernel, which specifies the type of function used to transform the data into a higher-dimensional space. The other important hyperparameters are C and gamma. C is the penalty parameter that controls the trade-off between maximizing the margin and minimizing the classification error. A smaller value of C creates a wider margin at the expense of allowing more misclassifications, while a larger value of C creates a narrower margin and may lead to overfitting. Gamma, on the other hand, controls the shape of the decision boundary. A smaller value of gamma creates a smoother decision boundary, while a larger value of gamma creates a more complex decision boundary.

The Support Vector Machine (SVM) algorithm also has hyperparameters that need to be tuned to achieve the best performance. One of the most important hyperparameters is the kernel, which specifies the type of function used to transform the data into a higher-dimensional space. The other important hyperparameters are C and gamma. C is the penalty parameter that controls the trade-off between maximizing the margin and minimizing the classification error. A smaller value of C creates a wider margin at the expense of allowing more misclassifications, while a larger value of C creates a narrower margin and may lead to overfitting. Gamma, on the other hand, controls the shape of the decision boundary. A smaller value of gamma creates a smoother decision boundary, while a larger value of gamma creates a more complex decision boundary.

Fitting 5 folds for each of 27 candidates, totalling 135 fitsBest hyperparameters: {'C': 0.1, 'gamma': 0.1, 'kernel': 'linear'}. These are the hyperparameter values that we get after implementing SVM model on our dataset.

The table below displays the performance metrics for each of the hyperparameter tuning methods in addition to the baseline model with no hyperparameter tuning. From the 3 graph plots, we can conclude that speed plays the most important part in deciding which Pokemon will win the battle.

	Baseline	Random Search	Grid Search
R2	-0.000617	0.886196	0.904790
MAE	0.228272	0.068252	0.064968
MSE	0.068797	0.007824	0.006546

5.2 Random Forest

The Random Forest regressor algorithm has a good amount of hyperparameters to tune. Random Grid Search found the following values for the specified parameters: ‘n_estimators’: 500, ‘min_samples_split’: 6, ‘min_samples_leaf’: 1, ‘max_features’: None, ‘max_depth’: 35, ‘criterion’: poisson, ‘bootstrap’: True. The n_estimators specify the number of decision trees that are built, and trained independently (in parallel) to create the final form. The next parameter is min_samples_split which controls the minimum number of observations required to split a node within the decision trees. The min_samples_leaf parameters let the model show how many observations it takes to be a leaf node for each decision tree in the forest. Next is the max_features, which describes the number of features to look for when considering splitting at a particular node. In our case, the value chosen was ‘None.’ This means that all features in our dataset were considered for each decision tree. The max_depth parameter is the maximum depth of the decision tree. The best criterion metric chosen was poisson, which minimizes the native log-likelihood of the poisson distribution. In total, there were 200 candidates randomly selected out of 242,000 different combinations available for Random Search to try out.

We had to condense the grid space due to time and resource limitations. Based on previous iterations of Random Search during the tuning process, we found some hyperparameters, such as n_estimators and min_samples_split, did not change values. For those hyperparameters, we chose a smaller set of values to test for. Grid Search found the best set of hyperparameters out of 4,320 candidates. Here are the hyperparameter values Grid Search chose: ‘n_estimators’: 500, ‘min_samples_split’: 5, ‘min_samples_leaf’: 1, ‘max_features’: None, ‘max_depth’: 20, ‘criterion’: absolute_error, ‘bootstrap’: True.

In Table 2 it displays the performance metrics for each of the hyperparameter tuning methods in addition to the baseline model with no hyperparameter tuning.

	baseline	random search	grid search
r2	0.9255	0.9331	0.9357
mae	0.0485	0.0486	0.0486
mse	0.0044	0.0041	0.0039

Table 2: Performance metrics of Random Forest Regressor models

When comparing each of the Random Forest models, it seems as though the grid search hyperparameter tuned model has the best model. We also looked at each model's importance.

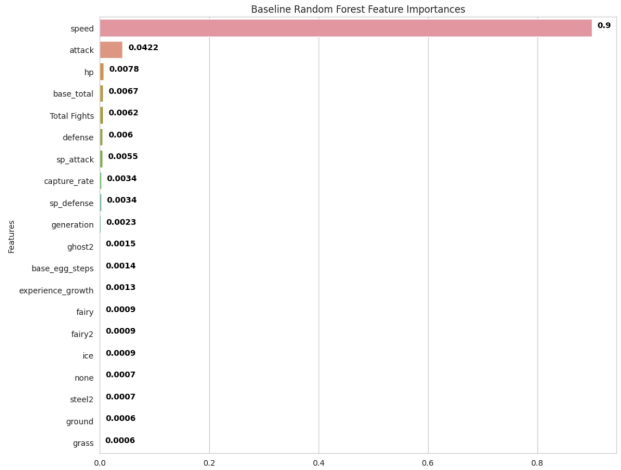

Figure 10: Feature Importance of Baseline model

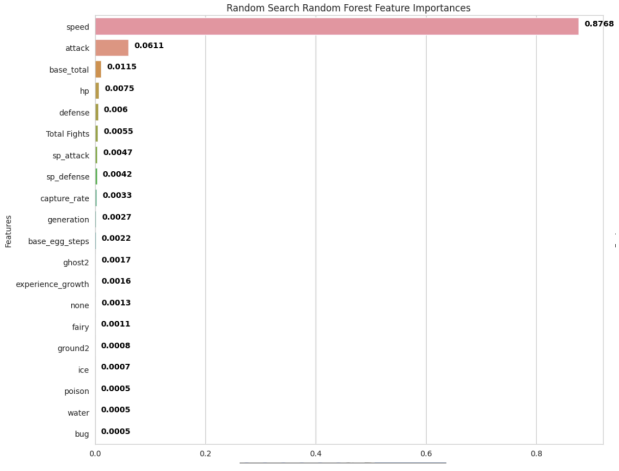

Figure 11: Feature Importance of Random Search model

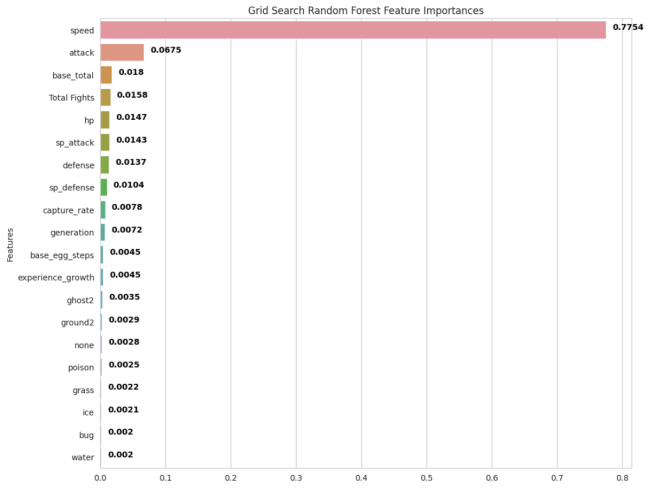

Figure 12: Feature Importance of Grid Search model

5.3 XGBoost

When running the pipeline for finding the best parameters for the XGBRegressor algorithm, Random Grid Search came up with these values for the specified parameters within the pipeline: 'reg_lambda': 0.1, 'objective': 'reg:logistic', 'n_estimators': 400, 'max_depth': 5, 'learning_rate': 0.01, 'alpha': 1. For the objective parameter, that specifies which loss function is to be minimized when training the data. With this combination, the random grid search chose logistic regression. The next parameter tuned is the reg_lambda which determines the L2 regularization term on the weights which was chosen to be 0.1. Nest is the n_estimator parameter. This gives the number of gradient boosted trees which is equivalent to the number of boosting rounds. It was chosen to be 400. Next is the max_depth, is the maximum depth a base learner's tree can grow. It was assigned the value 5. Then, the learning rate is the shrinkage factor which is set to control the weighting of new trees added to the model. That value was set to 0.01. Finally, the alpha parameter is the L1 regularization term on weights which is set to 1.

For the regular grid search, it had the same type of parameters as the random grid search, but it did set different values for some. Here are the parameters values the grid search set the model to: 'alpha': 0, 'learning_rate': 0.01, 'max_depth': 3, 'n_estimators': 400, 'objective': 'reg:logistic', 'reg_lambda': 0.001. The main differing values are alpha, reg_lambda, and max_depth.

In Table 3 it displays the performance metrics for each of the hyperparameter tuning methods in addition to the baseline model with no hyperparameter tuning.

	baseline	random search	grid search
r2	0.9086	0.9128	0.9082
mae	0.048	0.0467	0.0477
mse	0.005	0.0048	0.005

Table 3: Performance metrics of XGBoost models

When comparing each of the XGBoost models, it seems as though the random grid search hyperparameter tuned model has the best model. Thus, we also wanted to look at that model's feature importances in order to answer our second part of our research on which stat would have the most weight in determining the win percentage of a Pokemonas well as compare it to the other models..

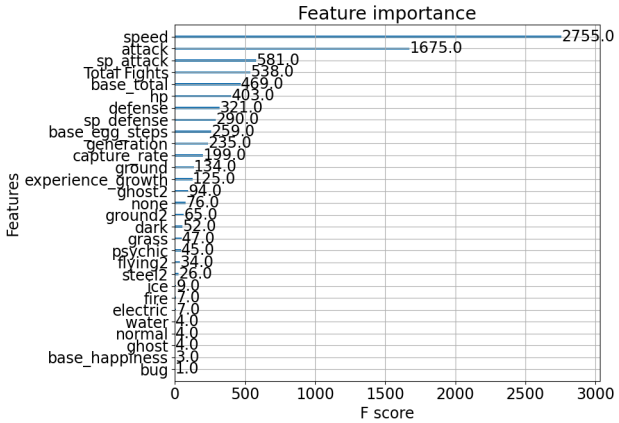

Figure 13: Feature Importance of Random Search model

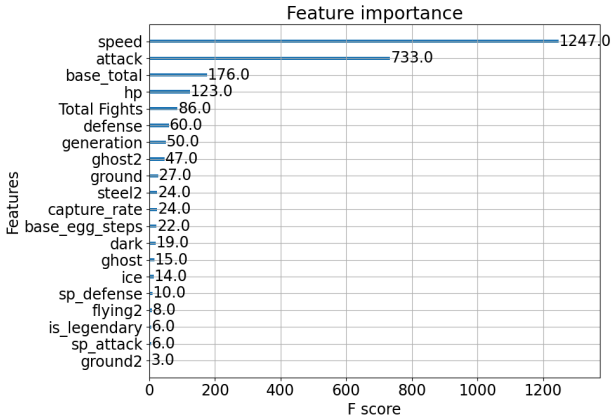

Figure 14: Feature Importance of Grid Search model

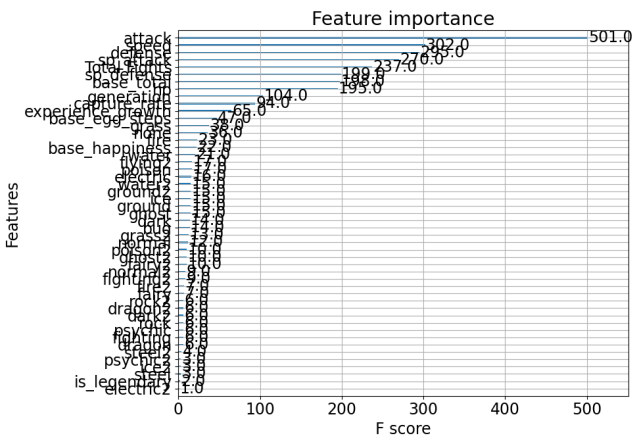

Figure 15: Feature Importance of Base model

From the random grid search hyperparameter tuned model in Figure 13, the top 3 features that had the most importance in determining win rate are speed, attack and special attack with speed being significantly more important than the rest of the stats. Similarly, the grid search tuned model from Figure 14 also elected speed and attack within the top 3 features that had the most importance with base_total being the differing feature. Again, speed and attack were chosen to be in the top 3 most important features for the base model, as seen in Figure 15, however the defense stat was the differing feature of importance. Additionally, most of the types of a Pokemon do not contribute heavily in the win percentage of a Pokemon according to all 3 models..

6 Conclusions and Discussion

The original questions we wanted to address in the beginning of our project is if we could accurately predict which Pokemon would likely win given their abilities and what features are most important for Pokemon battles. Using Exploratory Data Analysis techniques has shown us some interesting key facts about a Pokemon's characteristic and how they contribute to their overall all win percentage. We noticed that the speed stat had the highest difference between the top 10 and bottom 10 Pokemon. Special defense and special attack and similar differences (42 and 43 respectively) with hp having the smallest difference (17). This finding aligns with the game's battle system in which Pokemon with a higher Speed stat will generally make a move before those with a lower speed. Therefore, when two pokemon are in battle and have low health, the faster Pokemon will likely win in battle.

By using popular machine learning algorithms, Random Forest, SVM, and XGBoost, we confirmed our initial observations. We found that most of our models were able to produce accurate results with an acceptable amount of error. We saw that for the most part all of our models fitted our dataset well. However, R2 scores can be swayed due to overfitting. In this case, we can look at the Mean Absolute Error values to see, on average, the errors in our dataset were minimal. Random Forest and XGBoost models had similar scores for MAE. Compared to the other models, SVM did have worse MAE values. This could be due to the hyperparameter grid space chosen for a random search and the number of combinations tested for the Grid search model. Overall, the Random Forest Grid Search model had the best results.

We also found all models have the same top five feature importance. These include Speed, attack, base_total, hp, and total fights. We saw other stats, such as sp attack and defense, make it to some models' top 5 important features. However, that is to be expected due to the base stats being positively correlated, as shown in Figure 5.

We can expand upon the information we learned in this project by incorporating mega evolutions of their respective Pokemon counterparts.

REFERENCES

[1] Bentejac, C., Csorgo, A., & Martınez-Munoz, G. (n.d.). *A Comparative Analysis of XGBoost*.

[2] Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. 785–794. https://doi.org/10.1145/2939672.2939785

[3] *Ensemble Machine Learning Cookbook: Over 35 Practical Recipes to Explore Ensemble Machine Learning Techniques Using Python*. (n.d.). Retrieved March 7, 2023, from https://web.p.ebscohost.com/ehost/ebookviewer/ebook?sid=6306ac01-7bab-4a27-b11c-70062e0188cf%40redis&vid=0&format=EB

[4] Friedman, J. H. (2002). Stochastic gradient boosting. *Computational Statistics & Data Analysis*, *38*(4), 367–378. https://doi.org/10.1016/S0167-9473(01)00065-2

[5] Polikar, Robi. (2006). Polikar, R.: Ensemble based systems in decision making. IEEE Circuit Syst. Mag. 6, 21-45. Circuits and Systems Magazine, IEEE. 6. 21 - 45. 10.1109/MCAS.2006.1688199.

[6] Lever, J., Krzywinski, M., & Altman, N. (2016). Regularization. *Nature Methods*, *13*(10), 803–804. https://doi.org/10.1038/nmeth.4014

[7] University of Cincinnati (n.d.). *UC Business Analytics R Programming Guide*. UC Business Analytics R Programming Guide. Retrieved March 9, 2023, from https://uc-r.github.io/

[8] Pokémon type chart: Strengths and weaknesses. Pokémon type chart: strengths and weaknesses | Pokémon Database. (n.d.). Retrieved March 9, 2023, from https://pokemondb.net/type

[9] Archaic, Pip, T. W., Rhodehawk, & Max, W. (1970, March 9). *The original Pokémon Community*. Bulbagarden. Retrieved March 9, 2023, from https://bulbagarden.net/

[10] RANDOM FORESTS Leo Breiman Statistics Department University of California Berkeley, CA 94720, from https://www.stat.berkeley.edu/~breiman/randomforest2001.pdf

[11] Support Vector Machines: Theory and Applications,Theodoros Evgeniou, Massimiliano Pontil (2001)
https://www.researchgate.net/publication/221621494_Support_Vector_Machines_Theory_and_Applications

[12] Casella, Georges (2002). Statistical inference (Second ed.). Pacific Grove, Calif.: Duxbury/Thomson Learning. p. 556. ISBN 9788131503942.

[13]Willmott, Cort J.; Matsuura, Kenji (December 19, 2005). "Advantages of the mean absolute error (MAE) over the root mean square error (RMSE) in assessing average model performance". Climate Research. 30: 79–82. doi:10.3354/cr030079.

[14]Wackerly, Dennis; Mendenhall, William; Scheaffer, Richard L. (2008). Mathematical Statistics with Applications (7 ed.). Belmont, CA, USA: Thomson Higher Education. ISBN 978-0-495-38508-0.

[15]Claesen, Marc; Bart De Moor (2015). "Hyperparameter Search in Machine Learning". arXiv:1502.02127

[16]Bergstra, James; Bengio, Yoshua (2012). "Random Search for Hyper-Parameter Optimization" (PDF). Journal of Machine Learning Research. 13: 281–305.