Yuet Ho

Tanvir Ahmed

Mir Shahiduzzaman

Abigail Fontana

Ari Markos

Pokemon Classifier

**Abstract**

Goal: Given an input pokemon, how can we classify what its type is?

As a Pokemon research team, we looked into the most ethical way to classify the type of Pokemon based on its stats. We first split the dataset into 6 datasets, each with their own attributes. We then removed unhelpful data in determining the type and selected the best K value for our models. The next step was for us to run the models. The first model was Naive Bayes, for which we used Guassian, Bernoulli, and Multinomial. We found Guassian to be least effective compared to the other classifiers in the Naive Bayes. The Bernoulli Chi-Squared’s highest accuracy was 52%. The Multinomial provided the second highest accuracy with the Chi Square feature selection yielding an accuracy of 51%. We also used a decision tree. After optimizing and trimming, we learned the most accurate tree was the gini tree, with an accuracy of 47%. As for the K Nearest Neighbor, it provided the least accurate results, mainly due to the datasets having a high number of data points. While completing the ensemble, we were able to increase the accuracy for each, but only by 0.01 for Multinomial, 0.04 for the decision tree, and 0.02 for the decision tree using the random forest classifier. As a result, we concluded that we cannot classify pokemon solely based on its stats, but we hypothesized that it would be possible given more data, such as color and appearance.

**Introduction**

In the Pokemon world, features about pokemon are discovered through study. If we were Pokemon researchers, then it would be important for us to find effective ways to study Pokemon features. One challenging pokemon feature to study is its elemental types. While it can be simply studied by subjecting the pokemon to different elemental type attacks to see what it is weak against, this method of studying types can be considered unethical due to its abusiveness. So as Pokemon researchers, we want to build a classification system that can help study a Pokemon’s type and avoid such controversial research methods.

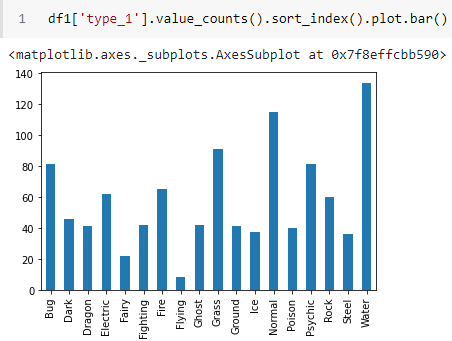
We chose a Pokemon dataset that contains all 1045 pokemon, which are creatures from one of the most popular video game franchises in the world. These creatures come in various shapes, sizes, and types, and our data set includes 52 different features available for each pokemon. Some notable features are battle stats (the stats of the pokemon); egg groups (what egg-type the egg is such as grass, human-like, mineral); types (the type of pokemon, which reflects on other attributes such as what it is powerful against; this is our target column); height and weight; legendary status (is the pokemon legendary, normal, or mythical); species (which family the pokemon belongs to); abilities (what the pokemon can do); catch rate (how often pokemon trainers catch them; friendship (how friendly a pokemon is).

The data set includes a class label, which is the type of the pokemon (column type\_1 and type\_2). Using supervised learning, we trained classifiers to predict the type of each pokemon. Given certain stats, our classifier could help predict the types of new pokemon in the future.

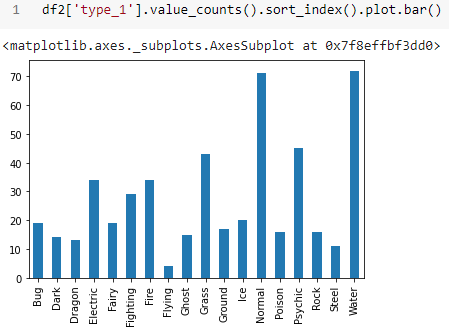
**Experiments**

Preprocessing

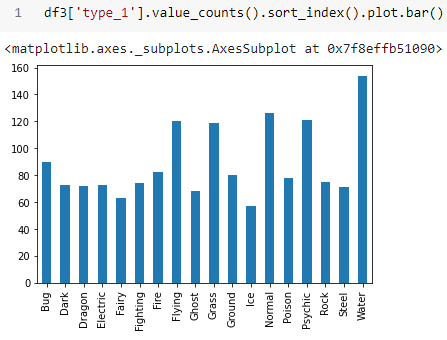
As previously mentioned, our class label consisted of two target columns: type\_1 and type\_2. This means that one pokemon could be two different types, such as flying and grass. This became an issue in our dataset because a classifier can only help us predict one class label at a time. Our solution to this problem was to create three different datasets from the original dataset. Then, we tested each classifier on each dataset to see which has better performance. Dataset 1 looks at just the 1st type of the Pokemon. It removes the 2nd target column that has the secondary type of the pokemon. This is useful in that we are keeping a lot of the data for the pokemon. Dataset 2 removes all the pokemon that have a second type. About half the pokemon in our dataset had a second type, so our code removed 552 pokemon. This made it easier for the classifier since it does not have to deal with any of the Pokemon with 2 types. Dataset 3 duplicated some pokemon in the data set. So, instead of having the second type of the pokemon in a second target column, the program added another row, with the same information, except the type 1 column was set to the original type 2 value. This added 552 more rows of the pokemon who have a second type. We thought this dataset would be the most inaccurate since there would be 2 rows of the same pokemon with the same stats that could have 2 different class labels. After performing the operations on datasets 2 and 3, we dropped the second type column since for dataset 2, the column would not have any data in it and for dataset 3, the column would be redundant. Dataset 1 and 2 worked on the assumption that type\_1 was the primary type for Pokemon while dataset 3 treated both types as equally important. We then made a numerical and categorical version for each of the 3 datasets, ultimately ending up with six different versions of the dataset. Below are the class distributions of type 1 for the 3 datasets. Datasets 1 and 2 have distributions skewed towards the top half of the 18 available types, with flying type being the least occurring primary type. Dataset 3 has a fairly equal distribution.



Dataframe 1 Type Distribution



Dataframe 2 Type Distribution



Dataframe 3 Type Distribution

In total, we had 6 datasets to work on. For our naming convention, their major names were DF1, DF2, DF3, corresponding to our plan to create three different datasets from the original dataset, with each having different assumptions and rows/columns dropped. Then their specific names were divided between version A and B, corresponding to how one version was converted to use numerical values only, and the latter used categorical values which would later be converted to all binary columns. So our final 6 datasets would be named: DF1A, DF1B, DF2A, DF2B, DF3A, DF3B. For example, DF1A would be the major dataset with type\_2 column dropped and all columns converted to numerical values (categorical features were binarized with one-hot encoding).

For the categorical dataset, we performed binning on Height, Weight, HP, Attack, Defense, Special Attack, Special Defense, Speed, Total Points, Catch Rate, Base Experience, and Egg Cycle. We used equal frequency for most of the features because they were skewed. The outliers were then replaced with the upper boundary. We removed 1 pokemon, Eternamax, from the dataset because this pokemon was an outlier in almost every column. Eternamax had almost double the total points of the next highest pokemon and was a huge outlier for height, so we decided to remove him. We also noticed that some 8th generation pokemon had missing data in the base friendship and base experience columns. In response, we filled in the missing data manually in the CSV file with the correct values from Bulbapedia (pokemon wiki).

For preprocessing, we first removed columns we knew intuitively to be unrelated to determining the type of the pokemon. This removed 33 columns. Some examples were pokedex\_number, the German and Japanese names of the pokemon, and ability since it was unique to each individual pokemon and would have too many different values. We removed base friendship because it was the same value for most pokemon, so it wouldn’t be a great determining factor for type. Generation was removed because the type of the pokemon does not depend on which game of pokemon it belongs to. Percentage male was removed due to missing data and it was unclear if it meant the pokemon was female or nonbinary (pokemon can be male, female, or none). Finally, all of the “effectiveness against type” features were removed because it makes it too obvious as to which type the pokemon is as pokemon type gives them a direct advantage against some other types. Additionally, the premise of our classifier was to study pokemon without relying on unethically gathered data; the “effectiveness against type” data describes how weak a pokemon is to certain types of attacks and in order for this type of data to be gathered, a pokemon will have to be subject to many abusive attacks.

Then, for all 6 dataset variations, we had to perform feature selection to get the most efficient amount of features. First, we set up the K parameter for cross-validation and picked which dataset to perform selection. We wanted to optimize the best K for each run of selection such that we end up with the lowest K number of features for the highest accuracy score. Next, we ran the feature selection using our K parameter for Chi2 and mutual info selection methods, testing on each of the 6 datasets. We recorded the accuracy performance of each classifier for a dataset. Multiple rounds of feature selection for each dataset was recorded, and at the end, we would pick the most accurate classifiers with their optimal K value.

Df1a:

| **Dropped Features** | **Classifer** | **CV Folds** | **Selected K** | **No Selection** | **Chi Square** | **Mutual Info** |
| --- | --- | --- | --- | --- | --- | --- |

| Flying | GaussianNB | 10 | 25 | 0.25 | 0.3 | 0.29 |
| --- | --- | --- | --- | --- | --- | --- |
| Flying | Dstree Entropy | 10 | 25 | 0.37 | 0.36 | 0.35 |
| Flying | DSTree Gini | 10 | 25 | 0.37 | 0.36 | 0.35 |
| Flying | KNN | 10 | 25 | 0.21 | 0.21 | 0.21 |
|  |  |  |  |  |  |  |
| 8 types | GaussianNB | 10 | 25 | 0.37 | 0.36 | 0.37 |
| 8 types | Dstree Entropy | 10 | 25 | 0.43 | 0.43 | 0.41 |
| 8 types | DSTree Gini | 10 | 25 | 0.42 | 0.43 | 0.44 |
| 8 types | KNN | 10 | 25 | 0.27 | 0.27 | 0.27 |

| Flying | GaussianNB | 10 | 15 | 0.25 | 0.23 | 0.26 |
| --- | --- | --- | --- | --- | --- | --- |
| Flying | Dstree Entropy | 10 | 15 | 0.36 | 0.29 | 0.33 |
| Flying | DSTree Gini | 10 | 15 | 0.37 | 0.33 | 0.32 |
| Flying | KNN | 10 | 15 | 0.21 | 0.21 | 0.2 |
|  |  |  |  |  |  |  |
| 8 types | GaussianNB | 10 | 15 | 0.37 | **0.39** | 0.38 |
| 8 types | Dstree Entropy | 10 | 15 | **0.43** | 0.38 | 0.38 |
| 8 types | DSTree Gini | 10 | 15 | **0.45** | 0.42 | 0.4 |
| 8 types | KNN | 10 | 15 | 0.27 | 0.27 | **0.28** |

For example, the above table shows a portion of our recordings when determining the best K for dataframe 1a (dataframe that dropped type\_2 column, and is numerical data). We tested the data set in two ways. We then compared the accuracy of the data set with different dropped features. The first way, we just dropped the least frequent type, which in df1a was flying. The second way we dropped the 8 least frequent types, which in df1a was flying, fairy, dragon, steel, ice, poison, ground, and ghost. Then, we optimized the K value many times to see which number of selected features would give us the highest accuracy, which we determined using chi squared and mutual info.

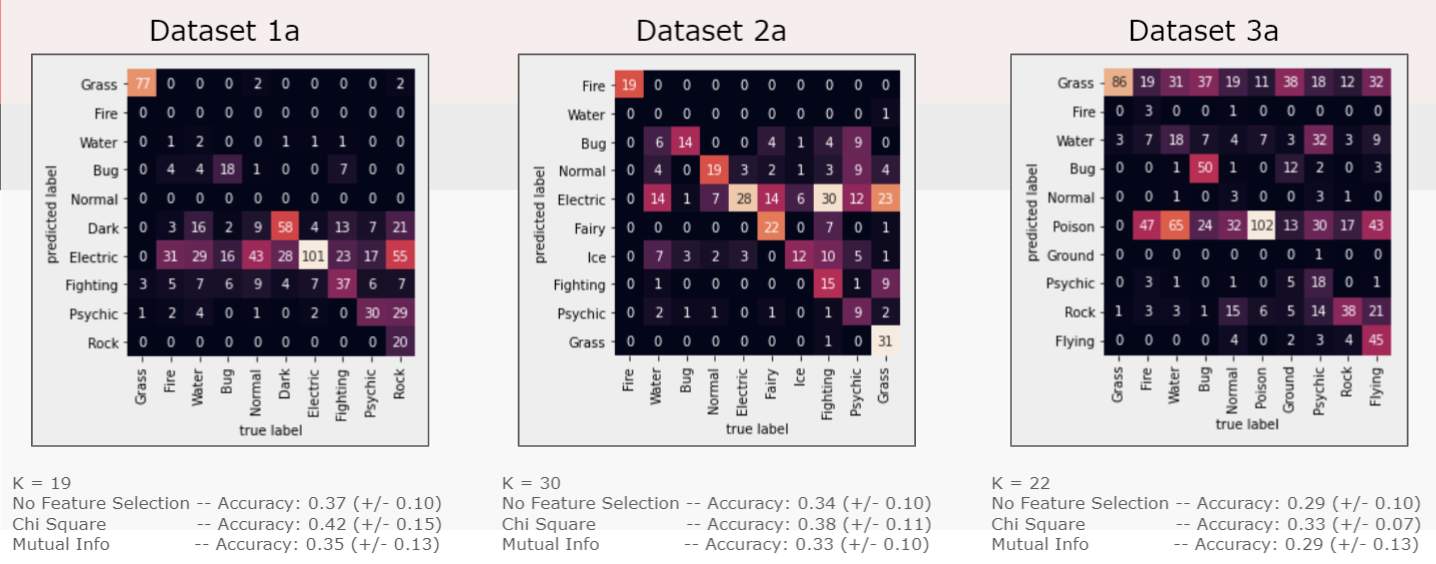
Our results from feature selection was: dropping 8 class labels (types) gave our classifiers the best results without compromising our results too much. In general, accuracy values were higher for data sets B (categorical) than data sets A (numerical); K values around 15-25 gave the highest accuracy for most datasets (in the above table, we see that K=15 gave better results). After testing, the top three classifiers were Bernoulli Naive Bayes (accuracy = 0.52), Multinomial Naive Bayes (0.51), and Decision Tree built with gini (0.47). We used these three top classifiers for further optimization and ensemble. Since our data set had a class label, we focused on supervised learning classifiers like naive bayes, decision trees, and K nearest neighbors.

**Results**

Naive Bayes

Gaussian:

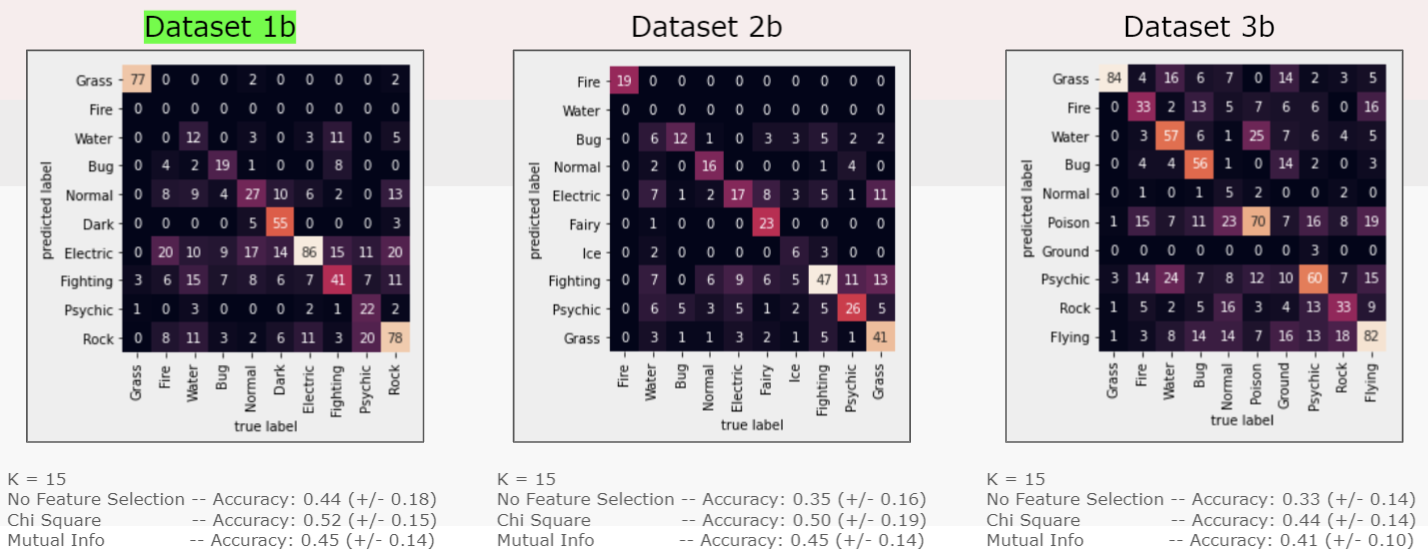
We ran this classifier on the A datasets because Gaussian is more effective on continuous datasets than categorical datasets. This classifier was the least effective in our findings and it did not make the cut to being one of the final three classifiers we decided to use. Here are heat charts of the results of our test on each dataset:



As you can see, the accuracies for dataset 1a were the highest out of all of them but not by far, and it still lagged behind the other classifiers. In addition, the classifier seemed to be biased towards labeling pokemon as either electric, grass, or poison type much more than other types.

Bernoulli:

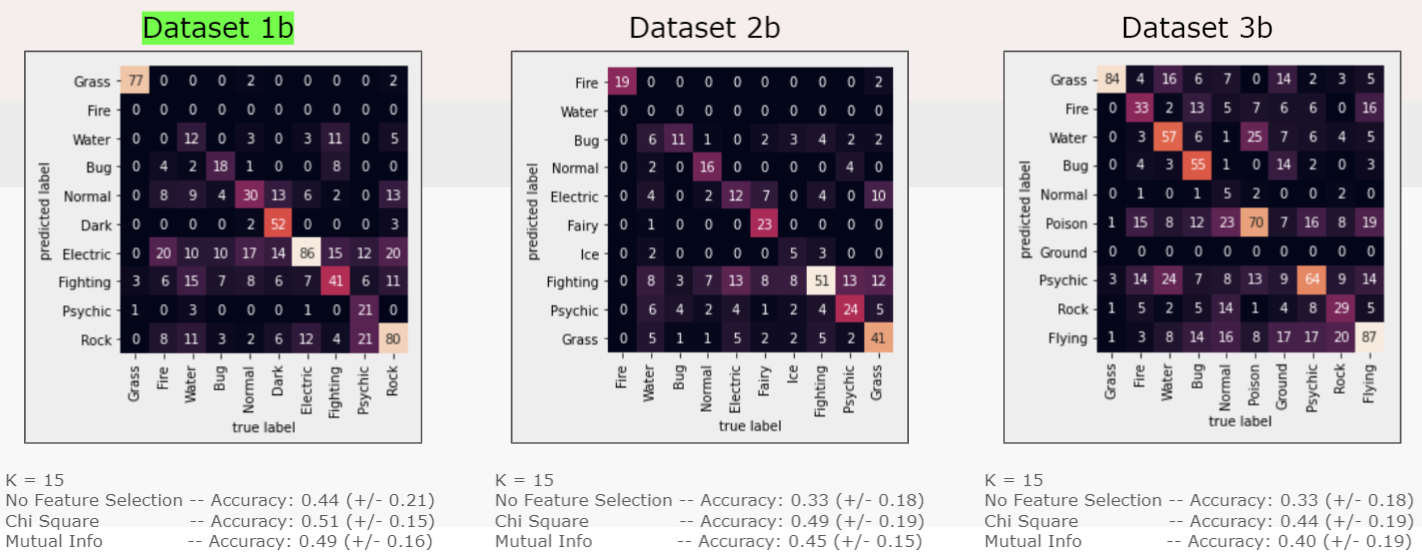
We ran this classifier on the B datasets because Bernoulli is the most effective categorical datasets in binary form, which was the exact format of our B datasets. Unsurprisingly, this classier netted us our highest score in terms of accuracy, as shown in the heat maps below:



Testing on dataset 1b with Chi-Squared feature selection gave us the highest accuracy value of 52%. We used this classifier going forward for additional testing and fine-tuning, such as using ensemble methods. These models also had a slight skew towards labeling as electric or poison type and the grass type prediction was the most accurate.

Multinomial:

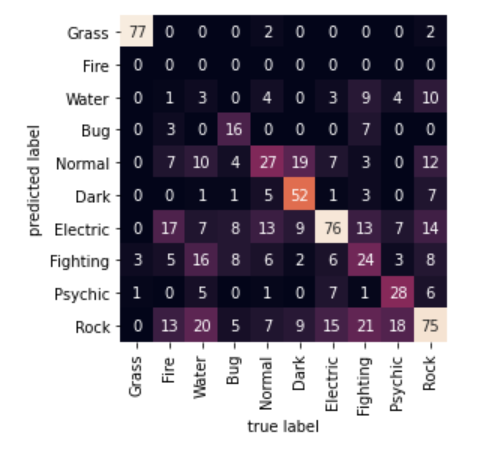
We also ran this classifier on the B datasets because it’s more effective on categorical data in frequency format. This model also ran particularly well and gave us our second-highest scores, as shown in the heatmaps below:



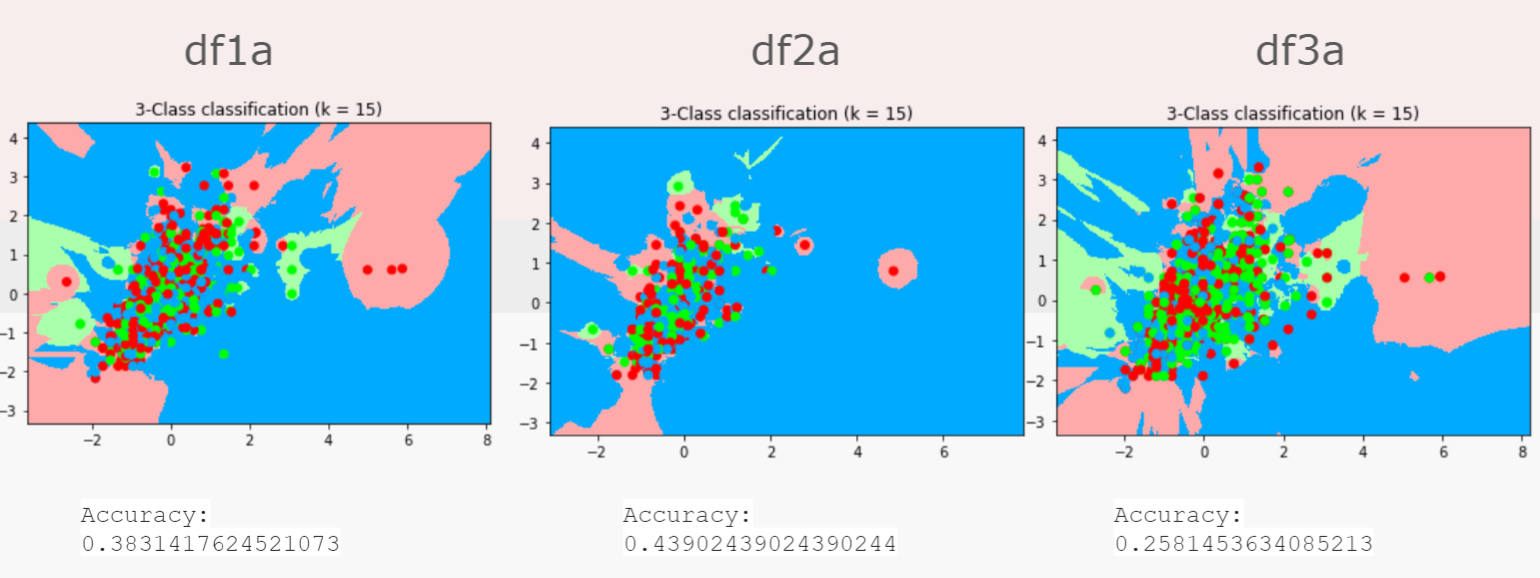
Dataset 1b also proved to be the most accurate and we chose this classifier on that dataset for further fine-tuning in the ensemble step. Very similarly to the Bernoulli models, these models had a slight skew towards labeling as electric or poison type and the grass type prediction was the most accurate.

Decision Tree

Another classifier we tried was the decision tree. We built the tree using cross validation with either entropy as the criterion or the default gini. We found that the gini tree had a higher accuracy than building the tree with entropy. As mentioned before, the B data sets (categorical versions) generally had a higher accuracy than the A data sets (numerical). Since we built the tree using the sklearn package, we made the categorical data binary, and then built the tree using that dataset. Then, we optimized and trimmed the tree by manipulating the parameters. We found a max\_depth of 12 and a min\_leaf\_sample of 6 to give us the most accurate tree of 47%. This accuracy is pretty low, but it was still the most accurate. Below is a confusion matrix heat map of df1b:



Visualizing the accuracy using confusion matrix heat maps shows that some of the types that were more accurately labeled are grass and electric and some of the ones that were less accurate were water and fire.

K Nearest Neighbors 

In these three graphs, the KNN Classifier made very clear what our results were going to look like. Right from the beginning, the assumption was that the dataset with the smallest amount of data would yield the best result. This is because the KNN Classifier struggles with large datasets, especially when the data does not follow any discernible pattern that the machine can use to group the points. The third dataset (df3a), in which every pokemon with a second type was duplicated, had the lowest accuracy score not only because it was the largest dataset but also because duplicating pokemon only proved to confuse the classifier as every pokemon with two types produced two data points with the same values but different types. The second dataset (df2a) produced the best results with an accuracy score of about 44%. This was because it was the smallest of the three datasets, due to the fact that every pokemon with a second type was removed. With that being said, the fact that the *highest performing dataset* only produced an accuracy of 44% points to the conclusion that pokemon types are not easily discernible based on stats alone. The stats do not seem to be related to type, as the data points are clustered very closely together and are just too varied to be able to find any groupings.

Ensemble

For the ensemble, we plugged our top three models into the bagging classifier. Initially, the ensemble classifier with its max features and sample parameters set to 0.5 did not increase the accuracy performance of Bernoulli NB, Multinomial NB, nor Decision Tree Gini. We wrote for loops to automate the process of testing different combinations of these two parameters. Until we got our highest possible accuracies for each model. At the end we had 0.52 for Bernoulli and Multinomial, which was a 0.01 increase, and 0.51 for Decision Tree Gini, which was a better 0.04 increase. The random forest classifier for Gini did not produce as good an increase with Gini, giving only 0.49, a 0.02 increase from just running the Gini tree individually (0.47).

**Conclusion**

Our final findings is that our available data cannot accurately classify pokemon according to type. The performance scores of our models were simply too low. The highest accuracy score we ever got was 0.52 for Bernoulli naive bayes, which is still too low. Our classifiers were inaccurate for various possible reasons. One reason may be that we are missing vital information such as appearance and color. Pokemon stats could vary wildly depending on the evolution state or the different forms that they have (Mega Evolution, Gigantamax, etc), so this information would have also been useful.

Another aspect we found is that egg-type data is favored by the feature selection methods. Egg type was the most occurring selected feature. We believe this is because egg type is exceptionally good at telling Pokemon apart since different types of pokemon usually fall under certain egg types more than others. This differs from the other stats of the Pokemon, which varies a lot more between Pokemon of the same type. However, this gave us very skewed results for certain types. For example, grass-type Pokemon were very accurately predicted compared to other types. This can be explained by the fact that there is an egg type also called Grass that the majority of Grass-type Pokemon fit into. This applies to the other types that had great accuracy such as Rock, which corresponds to egg-type Mineral. However, there are broader egg types that don’t refer to the Pokemon’s type but instead their qualities or physical traits, such as Amorphous, Human-Like, Monster, and Undiscovered. Since we did not have these features in our dataset, it was even more difficult to use egg types to differentiate. These egg types apply to Pokemon of many different types so using them to predict in our model messed with the ability of our model to differentiate the different types.

Limitations to our system included that we had 2 target columns as class labels. Removing a pokemon’s second type, as in dataset 1 and 2, takes away from a pokemon’s true type. However, keeping both confuses the classifier, as seen with data set 3. Another limit was that we had 18 different class labels (18 different types of pokemon). This was a lot of class labels for a model to handle. Typically, the max amount of class labels is around 10. So, we removed some class labels to make the classifier more accurate. We saw that removing the least frequent 8 types of pokemon yielded higher accuracy. But, when we remove a class label, this means that we will never have full accuracy in a classifier, because there will be Pokemon that cannot be identified as that type. Another limit specifically for our K nearest neighbor model was that it doesn’t work well with large datasets. Our data set 3 was especially large, since we added 552 rows, and we see that the accuracy decreased. Our data set in general was too large to get an accurate score using KNN. Our overall limit of our system was that our classifiers were too inaccurate to use in the real world. This is corroborated in another study which concluded that pokemon stats were not effective for determining types.

Future directions for research could be training image classifiers on pokemon photos. It is possible that similar types of pokemon tend to have similar physical features. For example, electric type Pokemon tend to be yellow, such as Pikachu. Many are also spikey like Jolteon. Looking for these specific physical traits in each pokemon could help better classify its type.

References

<https://bulbapedia.bulbagarden.net/wiki/Main_Page>

https://www.kaggle.com/mariotormo/complete-pokemon-dataset-updated-090420