CSCI 4830 - Final Project Report: Predicting Spawns of Pokémon in Pokémon GO

Report by Travis Roundy, Justin Olson, Anna Yudina, and Taylor Thomas

Abstract

In this paper, we will be discussing the development of our final project which took a deeper look into how Pokémon are predicted in Pokémon GO. We gathered information from Kaggle to produce our testable data sets. Our group created a parser in order to split the data up depending on what information we gave it. We tested our data by increasing classes and decreasing data points and then raising both to see if we could accurately guess which Pokémon was chosen. It turned out if we raised both the classes and data points we were more accurately able to estimate which characters would be chosen. These results are only the start of an experiment that could continue to grow to better estimate the outcomes of Pokémon GO.

1. Introduction

Pokémon GO is a mobile application that allows users to capture virtual Pokémon that are near the user's current GPS location. This game was released in July 2016, and it became one of the most popular games for the next few months. Fans of this game soon learned that certain Pokémon can only be found in certain locations. For example, a water Pokémon will most likely be found near a water source. They also realized that there are other factors/features that influence where the next Pokémon will appear or spawn. Since there are so many factors/features, it would be nice to know which ones are significant for future spawn predictions.

1.1. Our Data

Our data set came from the website Kaggle by the user SemionKorchevskiy. The data set consists of 25 features and 296,022 Pokémon sightings. Some of the features were categorical while others were continuous. For example, wind direction would be considered as a continuous feature, while the urban-rural feature is categorical (has pre-defined parameters for urban vs. rural). The data set can be found at the following link: https://www.kaggle.com/semioniy/predictemall

1.2. Parsing the CSV

In order to take the data file above, contained as a .csv file, we created a python script that takes the data and parses it depending on the option given in the command line. The main functionality is to put the .csv file into the Vowpal Wabbit format .vw file. Another functionality is to take the resultant .vw file and randomly separate it into Training and Test data, given a decimal for the size of the Training data. With these files, the data can be analyzed in Vowpal Wabbit, which is the main program we use in the following experiments.

1.3. Our Code

All of our work on this project can be viewed and contributed to on our Github page. This includes the parser as well as the training and test scripts needed to replicate our project. NOTE: Our data set is too large to fit on Github, so it must be downloaded externally from the link in Section 1.1. https://github.com/travisroundy/MLFinalProject

2. Related Work

While other members of the Kaggle community have done different tests on the data, one in particular is similar to what we are attempting in this project. Joe Ramir performed exploratory analysis and classified various Pokémon as "Rare", "Very Rare", and "Common" and did analysis based on those labels and came up with very interesting data. While this does not predict the spawn of various Pokémon, it is an analysis that could potentially help in later experiments. His results can be found here: Joe Ramir's Analysis.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

3. Experiments

3.1. Initial Tests

After creating the parser, we ran initial tests on the data in order to judge where to start our project. After running the created VW file as both the Training and the Test data, we discovered something that was skewing our data tremendously - our prediction file was only predicting the class ”16”. Out of the 296,022 data points, 52,114 of those corresponded to the class 16 which is the Pokémon Pidgey. Due to previous knowledge and playing the game first hand, this made sense as Pidgey spawned more than any other Pokémon. When only guessing Pidgey, the accuracy would return 17.605% which proved as the first baseline for the data.

In order to manage this, we edited our parser to only include certain classes of Pokémon based on how many times they appeared within the data set. Our first experiment started with limited data to 3000 data points which narrowed the classes down to 23 different Pokémon and prevented the skewing that we saw earlier. The second experiment lowered the data points to 2000 which allowed for 4 more classes to be added. The third experiment lowered the data even further to 1000 data points and allowed 45 classes to be classified. Finally, our fourth test allowed all 150 classes to be tested, but to prevent huge skewing of the data, we limited any class with over 5000 data points to 5000 to create a better balance.

3.2. Constants For Experiments

While testing our data in Vowpal Wabbit we used the same training and testing script for all of the below tests, other than changing the number of classes. We ran each training session at 50 passes through the data with a 0.2 learning rate to prevent over training the first few classes. We varied the oaa (one against all) with the number of classes for each experiment. For all of the tests, we used a logistic loss function because we were calculating probability of a class given the various features. For each experiment, we used our parser to randomly split the data into six different training sizes: 90%, 85%, 80%, 75%, 70%, and 60%. We then ran each training size five times and took the average of the five runs to give an accuracy result. We ran it five times because we believed it would model and get an accuracy similar to if we ran the experiment 100 times. After all of the training sizes received an accuracy, we took the average to get an overall accuracy for the experiment.

4. Results and Analysis

4.1. Experiment 1: 23 Classes with 3000 Data Points

For our first experiment, we took those classes that had over 3000 instances in the dataset and limited them all to 3000 instances to prevent any skewing. There were 23 Pokémon[1] This creates a baseline of 4.348% for this experiment. After performing the tests for each training size set, we received an overall accuracy of 18.071% which is 13.723% higher than the calculated baseline.

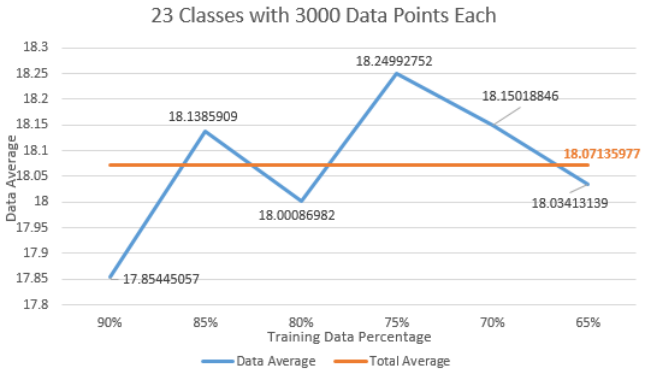
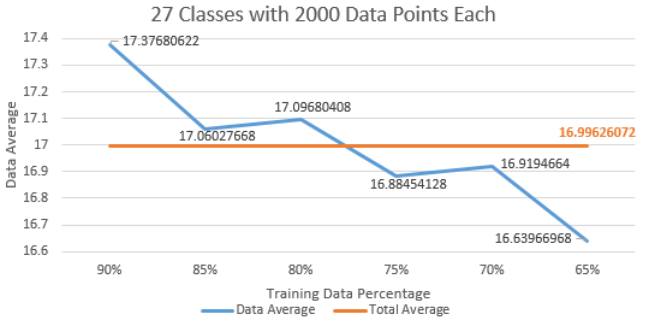


Figure 1. Graphical view of results for Experiment 1

4.2. Experiment 2: 27 Classes with 2000 Data Points

In the second experiment performed, we lowered the instance requirement down to needed at least 2000 data points within the dataset. This expanded the number of possible classes by four to a total of 27 different Pokémon[2]. This experiment has a baseline of 3.074% if a single class were guessed every time. The tests performed on this set of data resulted in a 16.996% accuracy. This is the largest difference between the baseline and the calculated accuracy with 13.922%.



[1] [”10”, ”13”, ”16”, ”17”, ”19”, ”21”, ”23”, ”29”, ”32”, ”35”, ”41”, ”43”, ”46”, ”48”, ”54”, ”60”, ”69”, ”96”, ”98”, ”118”, ”120”, ”129”, ”133”]

[2] [”10”, ”13”, ”16”, ”17”, ”19”, ”21”, ”23”, ”29”, ”32”, ”35”, ”41”, ”43”, ”46”, ”48”,”54”, ”60”, ”69”, ”96”, ”98”, ”118”, ”120”, ”129”, ”133”, ”27”, ”74”, ”92”, ”116”]

Figure 2. Graphical view of results for Experiment 2

4.3. Experiment 3: 45 Classes with 1000 Data Points

The next experiment dropped the instance requirement to 1000 data points each, which greatly increased the number of classes to 45 Pokémon[3]. The baseline accuracy for this experiment is lower than the other experiments and is 2.222%. For each of the training sets, there was a large difference between each run. Overall, this resulted in a total average accuracy of 14.456%. The resultant difference is 12.234%.

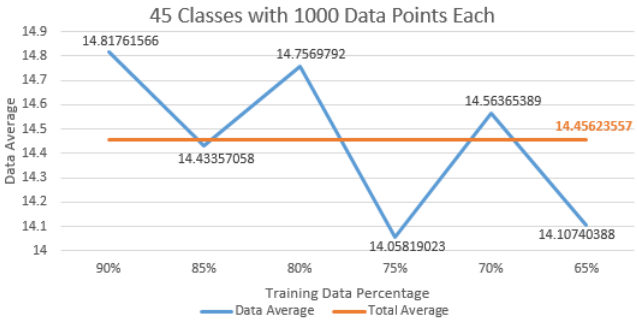


Figure 3. Graphical view of results for Experiment 3

4.4. Experiment 4: 150 Classes with up to 5000 Data Points

The final test expanded the dataset back to almost full bringing back all 150 Pokémon for classification. For this test, we included all of the data except for one change. To counteract the initial skewing of the data encountered, we limited all classes with over 5000 instances down to 5000 for this experiment. In total, this brought the total datapoints down to 156,790 creating a baseline of 3.190%. This experiment has the greatest impact on whether this overall project would succeed in the future. When performing the same methods as above, we get a 12.884% accuracy average. While this results in the lowest difference, 9.694%, we still achieved a large difference between baseline and average which, with some more refining, would promote the continuation of research on this project if one desired.

[3]["10", "13", "16", "17", "19", "21", "23", "29", "32", "35", "41", "43", "46", "48", "54", "60", "69", "96", "98", "118", "120", "129", "133", "27", "74", "92", "116", "1", "14", "20", "39", "42", "52", "56", "58", "63", "72", "77", "79", "81", "90", "100", "102", "111", "127"]

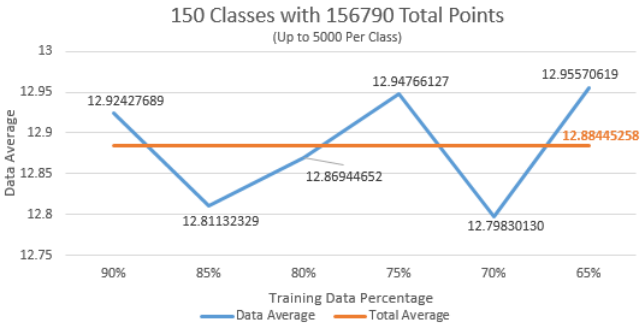


Figure 4. Graphical view of results for Experiment 4

5. Conclusion

Coming into this project, we were hoping to achieve greater than 50% accuracy with our classification. After our initial test, we realized that would not be the case and our focus turned towards getting the best results that we were able to. Our highest accuracy and greatest differentials came when we had less classes with more data points like in Experiments 1 and 2. But the most valuable experiment was Experiment 4 because it showed us that we are able to classify the data better than a baseline even without an equal number of instances between all of the classes. Due to this result, we have the possibility of furthering the project and improving it even further with future work.

6. Future Work

In terms of future work, it is important that we gather more data for the Pokémon that did not have much information. This will allow us to test more classes and help us figure out which features are important for predicting Pokémon spawns. With the data we do have, determining the best/worst classes could also help us raise the accuracy of our classifier. Finally, we can try running the same tests that we ran, but changing the set data points to try and pinpoint the key balance in the predictions.