Competitive Pokemon Predictive Model

Zachary Evans, Timothy Lencioni, Calvin Pugmire, Carlos Borja Leano
CS 201R
Brigham Young University
zmevans2017@gmail.com, tlen20@byu.edu, calvinpugmire@gmail.com, cdborja@byu.edu

Abstract

The goal of this paper is to find a model which can accurately predict whether or not a Pokemon team will win or lose against another Pokemon team without knowledge of the experience of the player. To gather information using the python package Selenium from smogon.com. Additionally we took each Pokemon's name and split it into base stats and Pokemon type of each pokemon. We then trained perceptron, multilayer perceptron with backpropagation, clustering and decision tree models to find the model that works best. In the end our models were inconclusive and did not do much better than random guessing. But we were able to deduce information about the data we had to make decisions on optimal team composition.

1. Introduction

Pokemon is a popular video game produced by gamefreak in 1996. The games eventually became popular enough to receive a competitive format. The first competitive tournament started on June 14th of 1997 in Japan. The competitive format began to grow with the first world championships beginning in August of 2009. Competitive Pokemon has now become a tournament where thousands of players play daily to test their skills and try to improve their strategies in hopes of competing in the world championships.

The format includes each person bringing a team of six unique pokemon. Each Pokemon has unique typings, abilities, stats, items, Effort Values (or EVs), and Individual Values (IVs), these are individual "qualities" that can affect the statistics of different Pokemon of the same species, to try to give it the best chance of completing its role on the team. Once the battle has begun and each player has seen the opposing six Pokemon, each player can then bring four of the six Pokemon to participate in a double battle format (where each player has two pokemon on the field at a time). The games are best two out of three and between each game players can change which four of the six Pokemon they bring. With over 1025 Pokemon currently available this means before the game begins there are over 3 quintillion possibilities of what the starting Pokemon will be in any given game. With this variance of possible starting states players must be adaptive and know rough estimates of each of these Pokemon's potential stats, moves and abilities in order to have the greatest advantage of each of these Pokemon.

Many players prefer to keep the information about their teams private until it is time to participate in a tournament however and there is not a great way to numerically represent players' experience. We wanted to see if there was an accurate way to determine which pokemon team would win by only using the information used during team preview which only shows the pokemon that will be used.

2. Methods

2.1 Data

2.1.1 Data Gathering

Initially when gathering the data we reached out to the Collegiate VGC League, a group of colleges which challenge each other to improve their battling skills. When asking if they had any data we could use they provided us with over 600 unique battles that we could use to help train our model. These 600 battles included the names of all pokemon used in the six versus six team preview battles and the winners team was always imputed first. This gave us a table in the following format.

Urshifu-Rapid-Strike / Landorus-Therian / Amoonguss / Kingambit / Flutter Mane / Ogerpon-Hearthflame	Ogerpon-Hearthflame / Farigiraf / Landorus-Therian / Urshifu-Rapid-Strike / Flutter Mane / Rillaboom

Although this was great to begin cleaning our data, we needed to find other ways to gather more data. We then went to look at the website pokemonshowdown.com. Pokemon Showdown[1] is a common simulator that competitive and amateur Pokemon players use in order to practice using their teams. To do this we began looking at web scraping using the Selenium[2] Python Library.

We also needed a way to enumerate our data from Pokemon names to numerical data on which we could train machine learning models. Pokemon Showdown and VGC League does not give information about the specific IV and EV data of Pokemon on the team, nor would that information be common knowledge in future games we would be trying to predict with our models, so we need to limit the data we collect to general information about the Pokemon on the teams. So, we limit the data to basic information we have on each Pokemon.

We downloaded a CSV file from Kaggle[3] of Pokemon with the Name, Type and Base stats for each species, sampled below:

Base stats are the generic stats of each species of Pokemon which are then further raised or lowered with IVs and EVs. These give a good baseline for how a Pokemon will perform in a match.

2.1.2 Data Cleaning

In order to use the data we collected, it needed to be formatted in a coherent way. First, we needed to figure out which features we needed to keep. In the Pokemon dataset, we discovered we did not need to consider the Pokemon Number, Total (sum of all the base stats), the Generation it was introduced, or Legendary status. Total and Legendary Status features only are cause and effects of the Base stats anyway, and Typing is important because some types give an advantage or disadvantage over others.

#	Name	Type 1	Type 2	Total	HP	Attack	Defense	Sp. Attack	Sp. Defense	Speed	Gen.	Legendary
1	Bulbasaur	Grass	Poison	318	45	49	49	65	65	45	1	FALSE
7	Squirtle	Water		314	44	48	65	50	64	43	1	FALSE

The only missing values we had in the Pokemon dataset were in the Type 2 feature, which is a result of many Pokemon only having one singular type. Since we wanted Integer values in these features, we used a LabelEncoder in Python to turn every type, even NaN values, into an integer ranging from 0 to 21.

The missing data in the scraped dataset of team compositions amounted to teams that were composed of less than 6 Pokemon. Since this is allowed, we needed our model to account for this, so we added a Pokemon into our DataFrame called None, which had all zero stats and NaN for Type 1 and 2.

Another issue we ran into was different variants of the same Pokemon. For example, the Pokemon Urshifu has two forms, Rapid Strike and Single Strike, that give different Typings and Base stats. The problem is, that Pokemon Showdown does not give us the information of which form is being used, as the opponent would not know until the Pokemon was used. So, we had to modify the name from the team composition data and used the base stats of the more commonly used Rapid Strike. This was a common pattern for other Pokemon with multiple forms.

There was also a bit of cleaning to do on the datasets since some names would be misspelled, that was done manually on the dataset itself to correct those human errors.

After the datasets were all clean, we cycled through the team composition datasets, and for each name in the team, we extracted the types and stats, and appended them as a feature in our new DataFrame. So there were 8 features extracted per Pokemon per team, meaning each datapoint had 96 features, with the target feature being a 0 if Team 1 won and a 1 if Team 2 had won.

After combining the DataFrames from the two team composition datasets we had collected, our DataFrame was ready to be used in the training of machine learning models.

2.2 Methods

We decided on using four main tools in analyzing our data: Perceptron, Multilayer Perceptron with Backpropagation, Principal Component Analysis, Clustering, namely KMeans, and Decision Tree Model, specifically XGBoosted Forest. We chose a

[1] https://pokemonshowdown.com/

[2] https://selenium-python.readthedocs.io/

[3] https://www.kaggle.com/datasets/tuannguyenvananh/pokemon-dataset-with-team-combat?resource=download

Perceptron and MLP Classifiers in order to utilize the large data inputs which are handled very well by these Classifiers compared to others. Since our data is all numerical after our cleaning and engineering, we hypothesized that Perceptron based Classifiers would be a good model to start with.
Principal Component Analysis is a way to examine the data for the features that contribute most to the overall information of the dataset. So we would use a PCA model to examine the most important features to make our model more computationally efficient and possibly more accurate.
Finally Clustering was considered since we assumed that the winning teams would be “closer” to each other in the 97-dimensional space and would therefore be clustered together.

2.2.1 Single Layer Perceptron

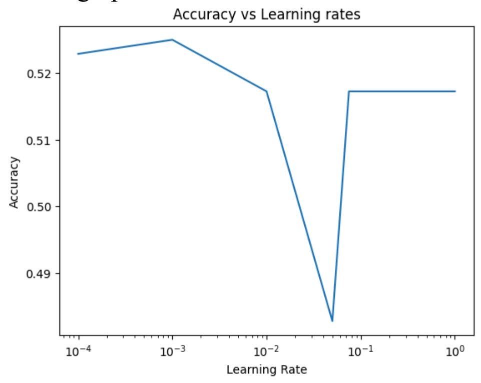
Since our model is based on victory or defeat, we decided that it may be best to base our model off a single layer Perceptron which can find us a linear division between victory and defeat. Additionally, the line separating the two victories may give us an idea of the ideal team to use (since stats along sed line define 2 teams which should be close in power) giving us a glimpse of what an ideal team looks like.
The goal would be to use the highest possible stats in each dimension. It also would show which stats should be prioritized over others with any positive stats being very important and negative stats negligible. To train our cleaned data we first converted the csv with all our data into a pandas file. Then we separated the results from the features. Then after shuffling the data we trained the model.
After running the model a few times we got the coefficients in this table.

```
[[-1.154e+02  2.460e+01  3.350e+01  1.850e+01 -6.100e+00  5.400e+00
  -7.680e+01 -7.300e+01 -1.733e+02 -1.250e+02 -3.000e-01  1.020e+01
  -7.210e+01 -8.840e+01 -9.600e+00  8.700e+00  1.790e+01  3.830e+01
  -9.400e+01 -3.090e+01  7.920e+01 -5.940e+01  5.630e+01  6.310e+01
  -1.206e+02  7.230e+01 -2.540e+01  3.400e+00 -2.240e+01  3.530e+01
   3.000e+01 -7.820e+01  8.250e+01  1.733e+02 -8.760e+01 -5.330e+01
   2.630e+01  1.170e+01  2.600e+00  2.860e+01 -3.440e+01 -1.942e+02
   5.600e+00 -1.140e+01 -5.120e+01 -2.830e+01 -6.280e+01 -1.123e+02
  -5.490e+01 -1.130e+01  5.310e+01  1.320e+01 -3.180e+01  2.000e-01
   4.930e+01  7.000e+01  2.083e+02 -2.330e+01  1.500e+00 -2.630e+01
  -8.490e+01  1.180e+01  5.100e+00  3.330e+01 -1.535e+02 -6.870e+01
   9.500e+00  4.880e+01  1.490e+01 -9.200e+00 -3.860e+01 -3.760e+01
   3.058e+02  1.600e+00  1.710e+01  7.010e+01  3.650e+01  4.270e+01
  -5.670e+01  2.020e+01 -1.070e+01 -2.970e+01  8.910e+01 -4.140e+01
  -2.480e+01  5.560e+01  5.150e+01  7.930e+01  1.029e+02  1.288e+02
   4.880e+01  7.740e+01  1.531e+02  3.220e+01  4.720e+01  2.450e+01]]
```

The accuracy that was given after running this was 52.59%.

2.2.2 MLP with Back Propagation

After using a single layer perceptron, we decided that we should use a multilayered perceptron which would allow the model to have a better understanding of the relationships of each of the features with each other. We hoped this would allow us to get a model with a higher accuracy. We decided to train the model with a wide variance of hidden layers. After testing several amounts of hidden layers we decided that 15 hidden_layers produced the best results. With a learning rate of .001 we got a test accuracy of 51.1377%. After which we decided to try to find the best learning rate to train on. We re-randomized the data set and began to train again.
After this run and making a table graph to see which learning rate had the best results we were given this graph.

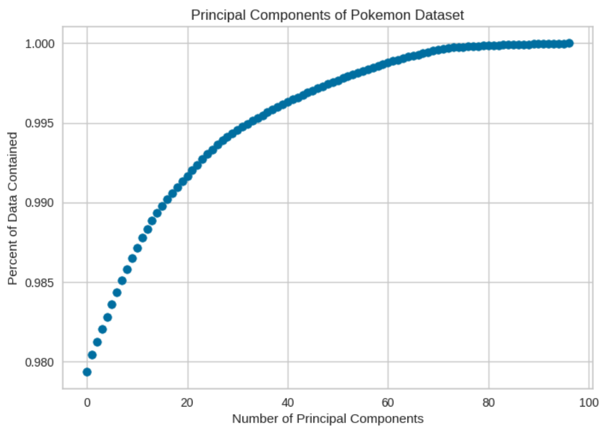


As seen the highest accuracy based on the test accuracy was .001 however as we take a look at the test accuracy at the table that helped give this graph we see the following.

```
   Learning Rate  Epochs  Training Accuracy  Test Accuracy
0         0.0001  1058.0           0.606850       0.522871
1         0.0010   703.0           0.534131       0.524982
2         0.0100    64.0           0.507624       0.517241
3         0.0500    97.0           0.492376       0.482759
4         0.0750    82.0           0.507624       0.517241
5         0.1000   110.0           0.507624       0.517241
6         1.0000   144.0           0.507624       0.517241
```

2.2.3 PCA

To further improve accuracy, we tried analyzing the principal components of the data. We thought that with too many features in the dataset, the algorithms might be overemphasizing less important features that could not be overcome in the training phase. By conducting Principal Component Analysis, or PCA, we would be able to find and keep only those components with the most relevant information to the dataset and hopefully be able to only feed the model the most important information available to us.
When conducting PCA on the Dataset we discovered that almost 100% of the data can be handled in the largest 25 Principal Components, as shown in the following plot.

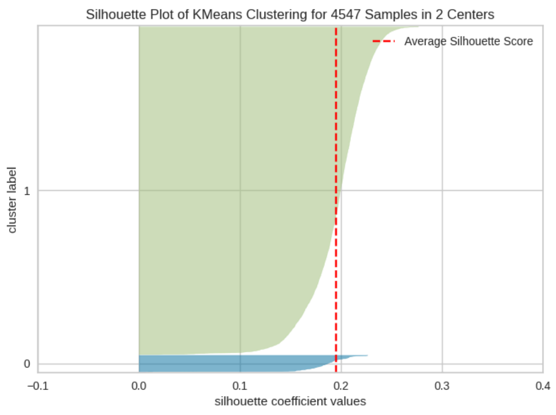


When training the models with only the most important principal components, however, our accuracy stayed consistent around 52%, which means that the issue is not the model getting stuck on irrelevant features, but an issue not covered in our dataset.

2.2.4 Clustering

For our Clustering algorithms, we started off using KMeans with initialization of K-means++. This means that the model runs a probabilistic analysis to find the initial guesses of the centers of the clusters. This is usually the best method in practice among professionals so we used that.

The other parameter we used was 2 clusters. We used this to mirror the binary classification problem. We experimented with several different numbers of clusters and 2 seemed to yield not only the best average silhouette score, but the best graph, which is shown below.



2.2.5 Decision Trees

In using Decision Trees, we used the Python package `xgboost`. This gives us a classifier of a Random Forest created from boosted trees. Boosted Trees are used more widely than vanilla Random Forests since Boosted Trees are designed to increase accuracy based on the results of the previously built trees in the ensemble.

Using a `GridSearchCV`, we were able to fine tune the hyperparameters to get the highest accuracy we could find. We found that the best learning rate was `0.3` and everything else was pretty much the default parameters. Decreasing the maximum depth of each tree actually decreased the testing accuracy of the model.

The next thing we can do is to run a feature importance algorithm on the forest to see how the algorithm ranks the importance of each feature. We would expect all of the features to be relatively similar feature scores, but seeing how important each feature is to the model will be enlightening for our data.

3. Results

3.1 Comparing Models

3.1.1 Perceptron

At first glance with the perceptron the accuracy is about equivalent to what the baseline accuracy would be at 50% having the advantage by about ~2.59% this is not ideal for the odds however, we can use the coefficients to attempt to find what an ideal team would look like. According to the table seen in 2.2.1 and looking at the highest absolute coefficients we see that of the first 8 stat values the ones that matter most seem to be HP since 3 of the 8 stats correlate to the HP stat. This is not enough though to be seen as a major aspect as Special Attack and Special Defense both appear twice in the top 8 as well.

3.1.2 MLP

When looking at the MLP it became obvious that higher hidden layers were more accurate. Thus we gave the model 15 hidden layers. However, when looking at the learning rate we only had a max test accuracy of 52.4982% This was barely better than the test accuracy of a learning rate of .0001, which we believe had such a good training accuracy due to overfitting. The MLP managed to get worse results than the single layer perceptron by .1% so maybe there were more problems than just the learning rate and number of layers.

The low training accuracy was also cause for concern. This usually happens when the data is too complex for the model to capture. This is most likely the case here where there are just too many additional factors that we are not able to see with our data that go into predicting the winner of a Pokemon battle.

3.1.3 Clustering

The clustering does not seem to work well, giving a fairly low silhouette score, as well as the inertia, or Total Sum Squared Error, which comes out to a staggering `228413862.65477383`, records rather exceptionally high (This was standard for most different numbers of clusters).
The high inertia value means that the clusters are very far away from their centers and spread out, while the silhouette score of `0.19` means that the data inside of the clusters are not all that similar.
Also, since we are basing our models on predictive capability, in calculating accuracy we found that it was not all that impressive.
Most of the data was clustered together, showing that most of the data is all mostly close together originally. This is not good for predictions, but it is not unexpected when considering our data. We are looking at Pokemon teams among professional Pokemon players and high level amateurs. In most video games, Pokemon included, there is a common "meta" used which most players follow. So, the teams will all be fairly similar and have similar stats.
Even among those relatively few points labeled 0, half of the data points were truly labeled 0 and the other half were labeled. So even the points that were labeled 0 do not give us much information, so putting them as a feature into a stacked ensemble method would not do too much more in terms of prediction.

3.1.4 Decision Trees

The Boosted Tree was able to get 52.5% testing accuracy with the parameters mentioned in the earlier section. One interesting parameter to note is that decreasing the maximum depth of the tree decreased the testing accuracy. Usually, pruning trees in this way is a way to avoid overfitting and as such increase generalization. However, that is not the case here.
This could be the case because the trees need access to every feature in order to obtain the highest possible accuracy. Even training with PCA made the accuracy decrease, which is more evidence that decreasing features in this case with a forest of Boosted Trees is counter productive.
Another important note to add, is that of the 6 most important features recorded by `xgboost` classifier, 4 of them were related to Speed of varying Pokemon. This makes intuitive sense as, in a turn based strategy game such as Pokemon, getting to move first can be the difference between winning and losing.
Still, with the 52% accuracy being one of the best of the models, it is apparent further work will be required on a deeper level in the data in the future to get a more successful model.

3.1.5 PCA

While Principal Component Analysis was able to find a good cutoff of components that contained almost 100% of the information in the data, it did not help too much of our models' predictive power. This shows that while the Principal Components may hold most of the information of the data, the information is still not relevant to making predictions on the data.
Using the first 25 on KMeans was not very much better than the entire dataset, but still saw some improvement in terms of clustering the data. The average silhouette score increased from `0.19` to `0.25` which shows that the data inside the clusters becomes more similar and the inertia gets cut almost in half, showing the clusters are more compact.
So, while the Clustering algorithm works better technically, the desired predictive power does not improve.

3.2 Future Improvements

As seen for almost all these models, our model only has an average of around .5 which is our baseline accuracy of random guessing method: victory or loss. This means that with the information we have given the model it is not possible for it to determine which team will win or lose. So in the future what information can we give the model in order to improve its accuracy?
One of the simplest points of data would be to give the model more data. This would mean that we would try to find out how often the 5000 battles on Pokémon showdown reset and scrape multiple times over to give the model more points to base its modeling after.
One major point of interest would be to give it a more complete understanding of the battle data. This would involve giving the model each Pokémon's complete stats (meaning stats with EVs and IVs included), their held items, and what moves they are using since each Pokémon can only know 4 different moves. With more data about each team, it may become more obvious which team would win.
A final component that is not included that may help the model know more would be to include player experience. This can roughly be calculated since Pokémon Showdown has a rating system

however this is much harder to calculate when looking at a professional Pokémon match. Thus, making it one of the most difficult implications of increasing accuracy of the model.

4. Conclusion

While our models were not very successful, we were able to learn much about our data and pinpoint areas for improvement in the future. We have concluded that the best model to predict the winner of a Pokemon game based on team composition is single layer perceptron.

The single layer perceptron gives us the highest testing accuracy for our main goal of predicting the winner. It also gives us an intuitive sense of which stats are most important in predicting who will win, with HP becoming seemingly more important than others. This is different from the feature importance analysis we did with the Boosted Trees, where Speed was important. This makes us believe that if, in future results, we can combine the Boosted Tree and Perceptron into a Stacked Ensemble method, we could take advantage of the two higher performing models we trained and get high importance of both HP and Speed, both very important in Pokemon Competitive Battles.

However, there is much more that needs to be considered if we want to make a more accurate prediction. None of our models were able to very accurately predict the winner and we have concluded that it is most likely due to the Pokemon battle system being more complex than who has the better team. There is an element of strategy and how a player uses their team that gives them a competitive edge. Since we do not have access to that information through our sources, we will seek to find that information in future experiments of this project.

This information could be used for giving odds on who is more likely to win, like gambling halls do for sports teams. But in terms of actually making an accurate prediction, we would need more information on specific items, moves and Individual and Effort Values for the Pokemon on each team, as well as potential information regarding the players skill level and performance history.