Predicting the Winner of a VGC Pokémon Battle With Machine Learning

Jackson Montuoro | Sean Fontaine

Introduction

The Video Game Championships, or VGC, is the official format of Nintendo's Pokémon tournament battles. Our original goal for the project was to create a machine learning model which could predict the winner of a battle based on information available in an open team sheet. We will reference this original attempt as the **winner prediction model.** As our project developed, we decided to shift to developing a model which can predict teammates based on two given Pokémon. We will reference this as our **restricted teammate prediction model.** In this proposal, our goal is to familiarize the audience with competitive Pokémon, explain the technical details of our dataset, and enumerate challenges we encountered.

Overview of Competitive Pokémon

Pokémon is currently one of the most recognizable and popular media franchises. From games, to merchandise, to even wrapping airplanes, Pokémon has maintained its cultural relevance nearly 30 years after the first games. Pokémon Red and Blue were released in 1996, followed soon by the immensely popular Anime and Trading Card Game. Due to the combination of mass appeal, clever marketing, and accessibility to the Pokémon IP, fans new and old continue to enjoy "Pokémania" in increasingly diverse ways.

Pokémon has always had a dedicated *competitive* fan-base. The first official competition, Nintendo Cup '97[1], was held locally in Japan as a means of promoting their new games. Even then, Nintendo recognized their games were imbalanced for competitive play, and thus introduced tournament rule restrictions. These rules continued to evolve over the years until formalizing into the Video Game Championships (VGC) in 2009, the format and style of which has carried on to this day.

What is VGC?

VGC is Nintendo's official competitive format. With hefty cash prizes, recognition within the community, and the personal fulfillment of being the "very best," players are highly motivated to strategize about their team loadouts. VGC is fought in a Bring 6, Pick 4 Double Battle format. VGC importantly operates in "regulations," which changes the pool of available Pokémon every 3-4 months (Figure 1). For our project, we will be focusing on the current regulation, Regulation I. What makes Regulation I different from previous regulations is the fact that teams are allowed up to two restricted legendary Pokémon. These Pokémon are far stronger than average, and in this regulation, most other Pokémon serve as supporters to this core.

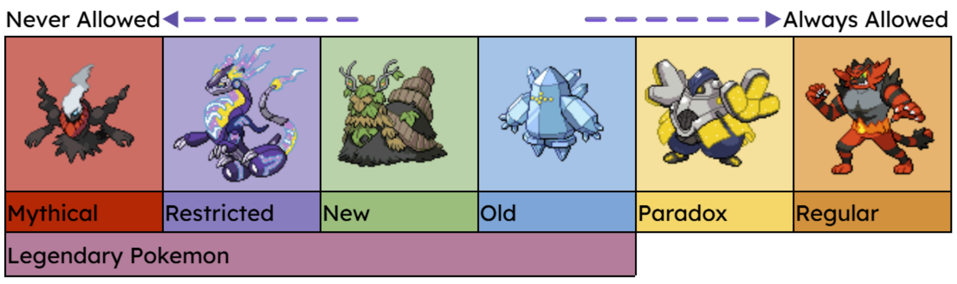


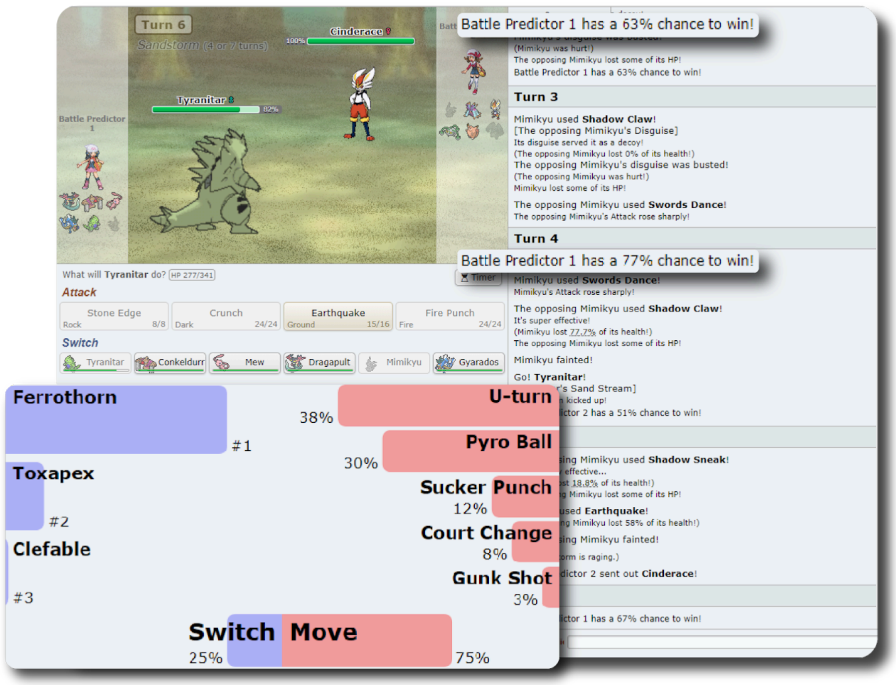
Figure 1 - Pokemon have different availability between regulations

What is the metagame?
A metagame refers generally to the most popular strategies used by players in a competitive game. This shifts quickly in Pokémon as community members adapt their strategies to best take advantage of popular teams and tactics. This is known as a strategic metagame[2]. Strategies developed initially as counters to the dominant strategy may in turn become dominant strategies themselves, leading to an ever-changing new range of counter strategies and plays. This is similar to how strategies evolve within games like chess. The current scope of this project only captures trends within a subsection of the Regulation I metagame (May 1st 2025 - August 31st 2025). Extrapolating results from this project to other metagames or timeframes will yield undesirable results.

What's the goal?
To create a tool to assist competitive Pokémon players. This goal underpinned both our original winner prediction model and our restricted teammate prediction model. As project goals shifted towards teammate prediction, we wanted to improve on existing tools like Pikalytics and PvPoke which assist players with team-building. We wanted to offer extra analytics and extend the usability of existing programs.

Related Work

The Pokémon Battle Predictor project began quite similarly to ours, with the intent of predicting the winner of a Pokémon battle. That project evolved into creating an agent with impressive results against human players as well as a browser extension for players (Figure 2). Our project is different in that it focuses on the 2 vs 2 double battle format.



Battle Predictor 1 has a 63% chance to win!

(Mimikyu was hurt!)
The opposing Mimikyu lost some of its HP!
Battle Predictor 1 has a 63% chance to win!

Turn 3

Mimikyu used **Shadow Claw**!
[The opposing Mimikyu's Disguise]
Its disguise served it as a decoy!
(The opposing Mimikyu lost 0% of its health!)
The opposing Mimikyu's disguise was busted!
(The opposing Mimikyu was hurt!)
Mimikyu lost some of its HP!

The opposing Mimikyu used **Swords Dance**!
The opposing Mimikyu's Attack rose sharply!

Turn 4

Battle Predictor 1 has a 77% chance to win!

Mimikyu used **Swords Dance**!
Mimikyu's Attack rose sharply!

The opposing Mimikyu used **Shadow Claw**!
It's super effective!
(Mimikyu lost 77.7% of its health!)
The opposing Mimikyu lost some of its HP!

Mimikyu fainted!

Go! **Tyranitar**!
's Sand Stream]
kicked up!
ictor 2 has a 51% chance to win!

ing Mimikyu used **Shadow Sneak**!
effective...
st 18.8% of its health!)
g Mimikyu lost some of its HP!

used **Earthquake**!
g Mimikyu lost 58% of its health!)

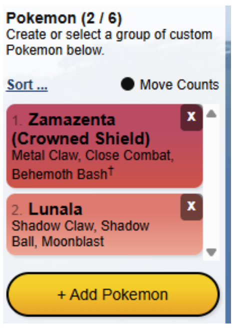
ing Mimikyu fainted!

orm is raging.)

dictor 2 sent out **Cinderace**!

ictor 1 has a 67% chance to win!

Figure 2 - The Pokemon Battle Predictor uses game-state logic to make predictions



Pokemon (2 / 6)
Create or select a group of custom Pokemon below.

Sort ... ● Move Counts

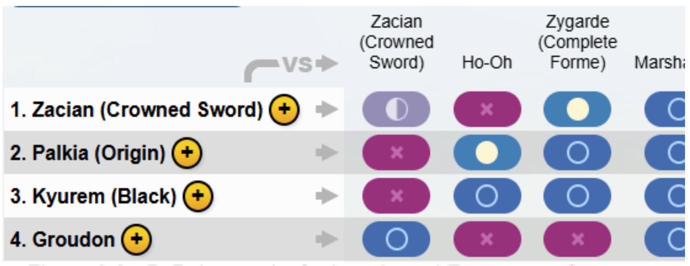
1. Zamazenta (Crowned Shield) ✕
Metal Claw, Close Combat, Behemoth Bash†

2. Lunala ✕
Shadow Claw, Shadow Ball, Moonblast

+ Add Pokemon

Figure 3.1 - PvPoke for Pokemon GO has an intuitive team entry system.

Although PvPoke is used as a team building tool for Pokémon GO, there are aspects of its design which we found useful to incorporate into our final supervised model. Shown on the left, this tool allows you to select any two Pokémon (Figure 3.1) and gives ranked recommendations for the rest of your team (Figure 3.2). Our project differs in several ways. PvPoke derives its rankings from simulating millions of battles–this is possible due to Pokémon GO's extremely simple, less stochastic battle mechanics. Our model focuses on Pokémon co-occurrence on teams created by real users. This is because VGC teams have exponentially more possible game states with items, movesets, EVs, etc.



vs	Zacian (Crowned Sword)	Ho-Oh	Zygarde (Complete Forme)	Marsh
1. Zacian (Crowned Sword)	◐	✕	●	○
2. Palkia (Origin)	✕	●	○	○
3. Kyurem (Black)	✕	○	○	○
4. Groudon	○	✕	✕	○

Figure 3.2 - PvPoke results for Lunala and Zamazenta C

Pikalytics is an information hub and team building database for competitive Pokémon players. Pikalytics hosts analysis on a wide variety of past and present metagames, including our metagame of interest: VGC 2025 Best of 3 Regulation Set I. Just like in our project, you select a restricted core of Pokemon (Figure 4.1) and receive a list of ranked recommended teammates. (Figure 4.2)

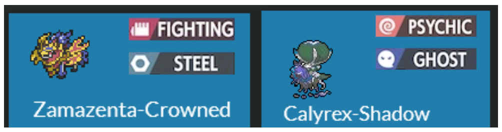
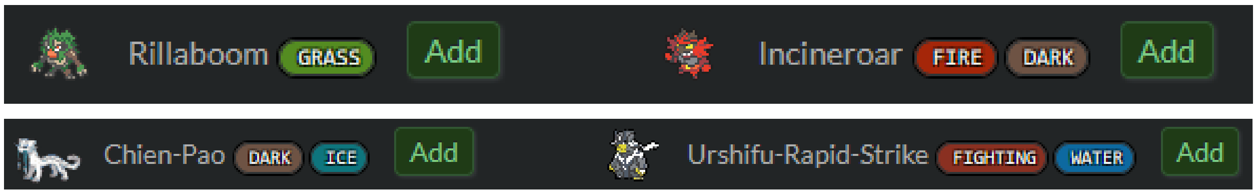


Figure 4.1 - Pikalytics Teambuilder UI

Figure 4.2 - Pikalytics Teambuilder Results



Pikalytics was essential for evaluating both our supervised component (teammate suggestion) and our unsupervised component (k-means clustering) by providing us with baseline team compositions to expect from our models.

However, Pikalytics fails to take into account some a priori assumptions our project aims to address.

As mentioned previously, restricted Pokémon are the most powerful Pokémon in the game, and are capped at 2 per team in VGC Regulation I to keep games balanced. In practice, players always select teams with exactly 2 restricted Pokémon, as selecting fewer of your strongest pieces is suboptimal–almost like playing a game of chess without a queen.

The Pikalytics team builder is limited in the following ways:

1. It allows players to select fewer than 2 restricted Pokémon, which does not line up with typical player behavior.

2. It continues to recommend restricted Pokémon as teammates even when a team already contains the maximum number, leading to the creation of invalid teams.

3. When suggesting teammates, it does not list the likelihood of each of its teammate predictions, leaving users unable to sort strong from weak predictions.

We hope to address these disconnects with a model that takes exactly 2 restricted Pokémon as input and outputs teammate predictions with transparent model likelihood metrics.

Data Sources

Pokémon Showdown Battle Logs

Our dataset consists primarily of 17,402 battle JSON logs scraped from Pokémon Showdown – a leading online platform for hosting VGC battles – using their official API. All battles are fought using the VGC Regulation I ruleset and occur in the window from May 1st to May 23rd 2025. Each JSON file contains the following *key* attributes:

Key	**Value**	**Example**
"format"	Ruleset used for the battle.	VGC 2025 Reg I (Bo3)
"log"	A string log detailing team compositions and turn-by-turn battle events.	turn|4\n|\n|t:|1750931481\n|move|p2a: Strong Tail|…
"uploadtime"	Unix timestamp showing when the battle was completed.	1750930850
"rating"	Average ELO (skill rank) of players in the battle.	1204

Pokémon-level Features

These battle logs were supplemented by data from public Github repository pokemonData, scraped by Len Greski from the website Pokémon Database. We use the file Pokemon.csv, which is indexed by Pokémon species; features include health, speed, offensive, and defensive values, as well as types (e.g. fire, water, grass...).

Preprocessing JSON files

Log data was cleaned, and regular expression string parsing was used on the messy "log" data to transform that data into a csv with the below key features:

Feature	Description	Example
p1_species_*	Species name of each Pokémon on P1's team	Incineroar
p1_item_*	Held item for each Pokémon on P1's team	Leftovers
p1_move_*_*	Moves 1-4 for each Pokémon on P1's team	Fake Out
p1_ability_*	Ability of each Pokémon on P1's team	Intimidate
p1_lead_*	The Pokémon P1 selected to lead	Miraidon
p1_win	Whether P1 won the battle	0 or 1
(same for P2)	Identical structure for P2	
rating	Average player ELO rating	1027
disconnect	Was the battle disconnected early?	0 or 1
forfeit	Was the battle forfeited early?	0 or 1
turn_count	How many turns did the battle last?	7

Combining our Datasets
From our Pokemon.csv file, we extracted the following 7 key Pokémon-level features (Figure 5):

Categorical: type_1 (primary type), type_2 (secondary type)

Numerical: attack, defence, sp_atk (special attack), sp_def (special defense), speed

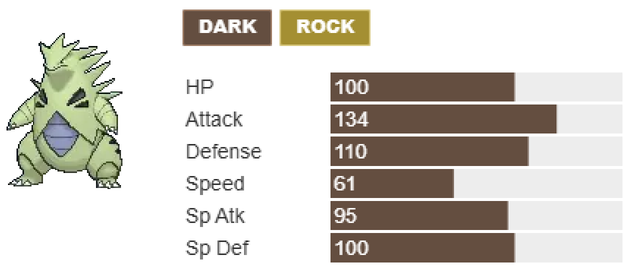


Figure 5 - Tyranitar has types (categorical) and stats (numerical)

These were merged with our battle DataFrame by mapping strings in the "species" columns to the "Name" index column of Pokemon.csv.

Applied to each species on each team of our original data, we produce features in the format "p[1-2]_species_[1-6]_[FEATURE]" (e.g. p2_species_4_type_2). This gives us 7 additional features per species, and 84 new features per battle (each team has a team of 6, so 6 x 2 x 7). These features give our models a more nuanced understanding of the gameplay differences between different Pokémon selections.

Feature Engineering

Additional features were engineered to give models a deeper understanding of team composition.

Counts of Pokemon with Unique Stat Profiles (12 new features)

Speed features: num_fast_mons, num_slow_mons
Speed features give us counts of the number of fast Pokémon and the number of slow Pokémon on each team. These categorizations are derived from percentiles. A “fast” Pokémon has a speed stat at or above the 75th percentile for Pokémon seen in our data. A “slow” Pokémon has a speed stat at or below the 25th percentile.

Defensive features: num_bulky_mons, num_frail_mons
Counts of the number of bulky vs frail Pokémon on each team. Bulk is calculated as the average of hp (hit points), defence, and special defence. Percentiles of this bulk distribution for Pokémon in our dataset are used to designate bulky (>= 75th percentile) and frail (<= 25th percentile) Pokémon.

Attacking features: num_high_power_mons, num_low_power_mons
Counts of the number of Pokémon with high vs low attacking power on each team. Power is calculated by applying argmax to Pokémon’s attack or special attack stat, fetching whichever value is higher. Percentiles of this argmax attack distribution are then used to designate high power (>=75th percentile) and low power (<=25th percentile) Pokémon.

Pokemon Type Features (3 new features)

Diversity: type_diversity
Count of the number of unique types represented on each team.

Matchup: type_matchup_score
Certain Pokémon types have advantageous matchups towards other types (e.g. water types are extra effective against fire types). The type matchup score compares two Pokémon teams by checking how well each team’s types can attack the other. It adds points for super-effective matchups and subtracts points for resistances or immunities, using the Pokémon type chart. The result is a single score showing which team has the better overall type advantage. A positive score indicates a better type matchup for player 1, a negative score indicates a better type matchup for player 2, and a score of zero indicates an even type matchup.

Move and Ability Features (22 new features)

Finally, there are features engineered to signal the presence or counts of particularly impactful moves and abilities (Figure 6). This gives our models data on likely team tactics, as the presence of these moves and abilities are central to several core VGC strategies.

Features of this sort include all of the below:

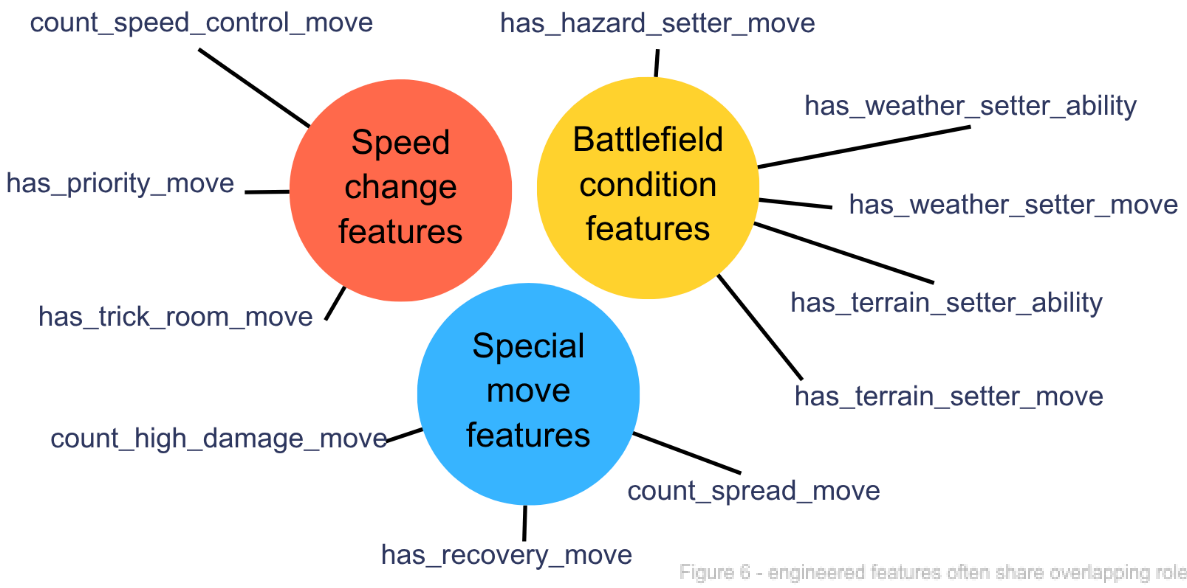


Figure 6 - engineered features often share overlapping roles

Feature Summary

Nominal	Numerical	Boolean	Total
159 features	68 features	35 features	262 Features

Challenges
Pokémon VGC is a complex game to model, and the preparation before the battle is no exception. During team building, players choose a team from over 1000 Pokémon, each with unique stats, moves, abilities, and types. Pokémon have 4 different moves to use per turn, and the same Pokémon can perform wildly differently between matches depending on EV distribution, items, movesets, and synergy with other Pokémon. VGC is played in a bring-6 pick-4 doubles format, expanding the total options available per turn and before-battle multiplicatively. As for other challenges, Pokémon battles include random factors like critical hits, move accuracy, and secondary effects, which complicates modeling. This introduces a stochastic element and makes it so no battle is ever deterministic.

Part A (Supervised Learning)

Methods Description
Our supervised learning task focuses on suggesting the remaining four teammates given a two-pokemon restricted core. This framing aligns with the needs of competitive players during the teambuilding phase of VGC, especially under Regulation I where two restricted Pokemon are nearly always present. The goal is to surface likely teammates from observed co-occurrence patterns in real team compositions.
To model this, we framed the problem as a multi-label classification task. Each Pokemon in our dataset could either appear or not appear on a team given a restricted core, resulting in a binary classification task for each species. We implemented this using a **MultiOutputClassifier** wrapped around ~400 **RandomForestClassifier** estimators- one for each possible

non-restricted teammate. Random forests were chosen for their ability to handle sparse categorical features and for their interpretability via feature importance.
We experimented with multiple multi-label classification strategies. One baseline we tried was a **OneVsRestClassifier** using L1-penalized logistic regression. Our reasoning was that most input features, especially Pokemon not part of the restricted core, contribute very little signal for any given teammate prediction. Since L1 regularization (lasso) forces coefficients on uninformative features to zero, we hoped this would reduce noise and lead to a sparse, interpretable model. However, performance was noticeably worse than with Random Forests. We attribute this to the inability of linear models to capture complex, nonlinear co-occurrence patterns in VGC team design.
As a third model family, we evaluated an instance-based approach using a **KNeighborsClassifier**. Although KNN requires minimal training time, it was significantly slower during prediction and yielded much lower exact match accuracy. This model struggled with rare teammates but managed to predict frequent ones, resulting in decent micro-F1 performance.
Features were extracted from team compositions, including species, base stats, and presence of key abilities and moves, and joined with auxiliary data sources (e.g., stat blocks from Smogon). We excluded player-specific information (e.g., Elo or username) to avoid potential data leakage and ensure the model generalizes to novel matchups. To represent categorical Pokémon identities, we encoded species as hashed or ordinal features and engineered aggregate statistics (e.g., sum of HP stats or average speed) to capture latent patterns in team strength or synergy.
To improve usability, we visualized the final predictions in a format inspired by Pokepaste, using sprite artwork fetched via PokeAPI. Each predicted teammate is displayed alongside its co-occurrence probability, helping players understand the likelihood of different team compositions and explore viable teammates given their chosen restricted core. Visit the web app here.

Supervised Evaluation
Evaluation for this task required multi-label metrics that capture how well the model predicts the correct set of teammates. Evaluation metrics were calculated on a held-out 20% test set. We rely primarily on **exact match accuracy, hamming loss,** and **micro/macro-averaged F1 scores.** These allow us to assess not just whether the full set of predictions is correct, but also how often individual labels are correctly predicted across all samples.

Model Type	**Hamming Loss**	**Exact Match**	**Micro F1**	**Macro F1**
Random Forest	.184	.393	.82	.63
Log. Regression	.0622	.015	.375	.305
KN Neighbors	0.040	0.029	.386	.135

Feature Importance and Ablation
We experimented with engineered features such as type matchup indicators, but observed minimal gains in predictive performance, suggesting our core feature set already captured most of the usable signal. This effectively served as an ablation analysis, informing us which feature additions were not worth the increased complexity.

Hyperparameter Sensitivity
We evaluated the sensitivity of our best performing model (Random Forest) to changes in key hyperparameters. Specifically, we varied the number of estimators (n_estimators) and tree depth (max_depth) using manual tuning. We found that increasing n_estimators beyond 100 resulted in negligible gains in F1 score, while increasing training time. Similarly, deeper trees (beyond depth 10) offered no clear advantage and slightly increased the risk of overfitting. These results suggest that the model is relatively stable across a range of hyperparameter values and does not exhibit high sensitivity, which improves its reliability for practical deployment.
We also indirectly explored sensitivity via K in our KNeighborsClassifier. We tested multiple values of k and found that smaller values (e.g., 3) tended to overfit to frequent teammates, while larger values (e.g., 5) achieved slightly better balance but still underperformed tree-based models overall. This confirms that KNN is less suitable for this high-cardinality, sparse prediction space.

Tradeoff Analysis
Each model presented a different set of tradeoffs:

- **Random Forests** offered the best balance of accuracy and coverage. However, they tend to be computationally intensive. They also tend to slightly favor recall over precision, especially for frequent teammates. This made the model strong at capturing common Pokemon but slightly more prone to false positives.
- **Logistic Regression** achieved low Hamming Loss because it rarely predicted teammates it wasn't confident about. This led to high precision but low recall, particularly hurting the prediction of less common or synergistic teammates.
- **K-Nearest Neighbors** was the *slowest* at prediction time, due to the need to compare each test sample against the full training set. This required cutting down the number of Pokemon to be predicted to 100 for timely testing.

A particularly important tradeoff across all models was **precision vs. recall** in the context of underrepresented teammates. Our models often defaulted to predicting common filler pokemon, leading to inflated precision but missed synergies. Improving recall on rare but meaningful teammates- without overpredicting- remains a key challenge for future iterations.

Failure Analysis

1. **Failure: Alphabetical Output ("Alcremie Bug")**
    a. **Description**
    In early runs of our random forest model, we observed that the predicted teammates often defaulted to a fixed list that began alphabetically- most notably

starting with Pokemon like Abomasnow, Alcremie, and Alolamola. The results appeared regardless of the input restricted core, producing near-identical and useless outputs across teams.

b. **Analysis of Possible Reasons**
This issue appeared when the model failed to confidently predict any teammates. Due to the multi-output structure, some trees returned default predictions for many labels. In these cases, the output defaulted to a set of Pokemon in the training set in Alphabetical order. This was a symptom of extremely low predicted probabilities across all labels, and inadequate post-processing logic.

c. **Category of Failure**
Systematic Error. This failure mode recurred consistently under preprocession conditions or core combinations, and was caused by shortcomings in how Random Forest handled label confidence.

d. **Proposed Improvements**
Current fallbacks include simply removing predictions below a binary threshold. Future improvements could include ensemble smoothing or adding a confidence floor for multi-label output filtering.

2. **Failure: Predicting Match Winners (Win Prediction Model)**
   a. **Description**
   In an earlier iteration of our work, we aimed to predict the winner of a Pokémon battle using only the team compositions (i.e., open team sheet data, without in-battle events). Despite using standard models and carefully engineered features, our best models performed only marginally better than chance.

   b. **Analysis of Possible Reasons**
   The team-only approach lacked key information: battle order, in-game decisions, and matchup-specific synergy. Because of the nature of VGC, even strong teams can lose due to RNG or poor play.

   c. **Category of Failure**
   Model Limitation / Underfitting. The model lacked critical information needed to make meaningful predictions, and the signal was overwhelmed by noise.

   d. **Proposed Improvements**
   Shift focus away from winner prediction and toward teambuilding tools. In the future, a larger dataset or datastream could help make this approach more viable.

3. **Failure: Stat-Based Clustering Introduced Data Leakage**
   a. **Description**
   While experimenting with unsupervised learning, we grouped Pokémon into clusters based on their base stats (e.g., bulky support, glass cannon) and used these cluster labels as features in our supervised teammate prediction model. Performance appeared surprisingly high.

   b. **Analysis of Possible Reasons**
   We realized that because restricted Pokémon were included in the clustering step, the model was indirectly learning about the presence of those Pokémon through the cluster features. Since those same restricted Pokémon were also

prediction targets, this violated the independence between features and labels- a classic case of data leakage.

c. **Category of Failure**
Data Leakage / Feature Contamination*. Features inadvertently encoded label information, leading to inflated performance and misleading results.

d. **Proposed Improvements**
We removed stat-based clustering as a feature source and transitioned to co-occurrence-derived features instead, which rely only on observed pairings rather than label-dependent encodings. This helped maintain a clean separation between inputs and prediction targets.

Part B (Unsupervised Learning)

Clustering

The goal in this component is to use clustering to identify and group similar team compositions. These clusters will aid our analysis in two key ways. First, cluster labels are added as additional features to improve our supervised learning processes. Second, understanding common team cores present in clusters will provide us with an additional evaluation channel for our supervised teammate predictor models. If we input a restricted Pokémon core common to a specific cluster and see our supervised model output teammate suggestions of Pokémon within the same cluster, this is great for the external validity of our results.

Clustering Methods

Data will be clustered using K-means and DBSCAN. As the dataset is high-dimensional and primarily binary, K-means is a good approach as it is scalable and allows for a straightforward comparison of team archetypes based on centroid similarity. Meanwhile, DBSCAN is valuable for its ability to discover clusters of more irregular shape. This could be important in identifying outliers or unusual team compositions that K-means might miss; a reasonable assumption in VGC where most teams may follow a common pattern but some may be highly creative or niche.

Data Transformation

As K-means and DBSCAN clustering cannot be applied directly to categorical data, features such as species, moves, items, abilities, types must first be transformed. These were one-hot encoded, and aggregated as *counts* per team. For instance, a team containing 3 Pokémon with the move Quick Attack would have a feature named “move=Quick Attack” with a value of 3. All features are then min-maxed to ensure all distance metrics are calculated on the same scale.

Due to a high variety of Pokémon species, moves, and items, the result of this encoding is a sparse feature matrix with 1502 features. This is a problem for two reasons. First, if data tend to contain zeroes on many dimensions, data points become artificially close to one another in a euclidean sense, negatively affecting both k-means and DBSCAN clustering. Second, k-means relies on the mean of data points to calculate cluster centroids. As the mean shifts towards zero, these centroids become less and less meaningful.

By removing only very rare features present in less than 1% of teams in the dataset, we reduced dimensionality fourfold down to 342 features. This retains most of the useful data at the cost of retaining a bit of sparsity.

In case this sparsity is still too much, we created an alternative dataset containing only the 18 features that were manually engineered for clustering (see feature engineering section). This reduces dimensionality and sparsity greatly, but at the cost of possible important data loss.

Given that both approaches have tradeoffs, we test both in our analysis.

Hyperparameter Tuning

In our k-means clustering, we conduct a grid search from k=2 to k=15 to find the ideal cluster number. Inertia is the sum of squared distances between a point and its centroid. Graphing this, we determine cluster count by finding the point at which inertia does not decrease significantly with the addition of greater clusters. We further evaluate cluster strength using silhouette score, a measure of cluster cohesion.

In our DBSCAN, we also perform a grid search, but on the epsilon parameter, which we test with values from 0.5 to 4.0 in increments of 0.2 for 20 searches in total. The ideal epsilon parameter is selected based on silhouette score and cluster count (ideally no greater than 15).

Unsupervised Evaluation

	K-means	**DBSCAN**
All Features	Clusters: 11 Silhouette score: 0.09	Clusters: 2 Silhouette score: 0.30
Only Engineered Features	Clusters: 2 Silhouette score: 0.46	Clusters: 13 Silhouette score: 0.49

When k-means is applied to our full feature one-hot-encoded dataset, it can only weakly define clusters, indicated by a very low silhouette score of 0.09. This may be due to noise and poor cluster separation as a result sparse high dimensional data (despite significant feature reduction from 1502 down to 342). DBSCAN similarly struggles to achieve strong clustering on the full feature one-hot-encoded dataset, and can only achieve moderate performance with 2 clusters (Figure 7.1).

Performance, however, greatly improves when clustering is done on only the 18 engineered features. K-means clustering produces two well-separated clusters with a silhouette score of

0.46. Examining the overrepresented species, moves, abilities, items, and features within each cluster, it is apparent that cluster 0 generally contains teams with a faster more offensive style, while cluster 1 contains teams with a slower more defensive play style.

DBSCAN performs even better on the engineered feature data, and identifies multiple well-separated clusters, including 13 nuanced sub-groups (Figure 7.2). Promisingly, these clusters are easily interpretable to those with VGC knowledge, and most contain species, moves, items, and abilities that are often paired together in popular themed teams (e.g. weather-setting teams, recovery-stall teams, trick room teams).

Below is a t-SNE representation of the two different clustering approaches.

K-means Clustering

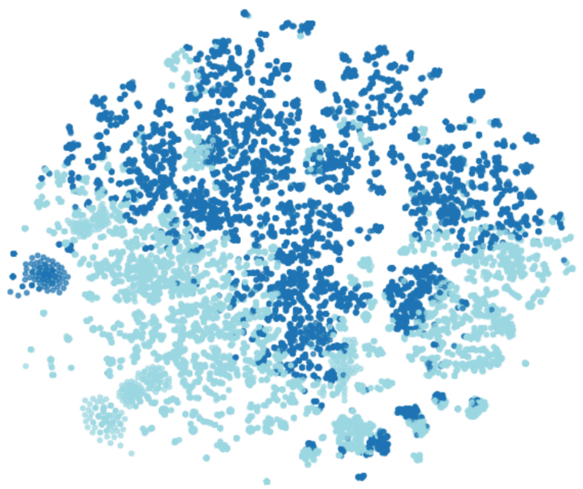


Figure 7.1 - KMeans with 2 Clusters

DBSCAN Clustering

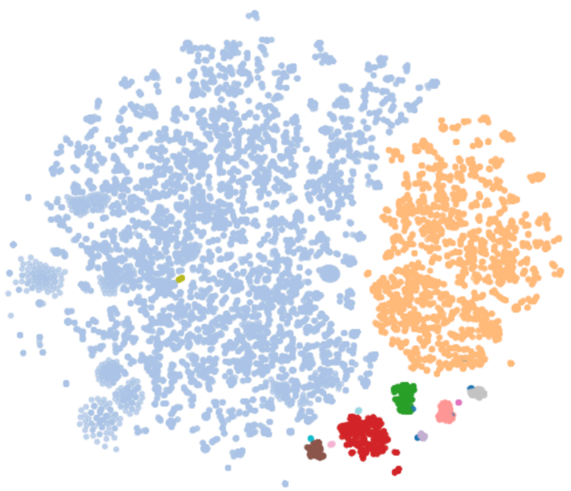


Figure 7.2 - DBSCAN with 13 Clusters

<u>Discussion</u>

What did we learn from doing Part A?

When performing Part A, we learned from both the abandoned **winner prediction model** and our successfully deployed **restricted teammate prediction model.**

A. What surprised us?
    a. One thing that surprised us was just how difficult it was to predict the winner of a battle simply given open team sheets and leads. We address this to the complexity of VGC.
    b. A pleasant surprise was to see how the model handled valid input that it had been sparsely trained on. For example, Calyrex-Ice + Miraidon is a common, proven core, and while the results for that input core made us happy, they were not surprising. However, what about impossible cores (in which both legendaries are alternate forms of the same pokemon), like Giratina + Giratina Origin or Cosmog + Cosmoem? Interestingly, we saw common Pokemon like Dondozo, Tatsugiri, Ditto, and Weezing Galar rise in our predictions. All 4 of these pokemon

have distinctly unique interactions with the mechanics of pokemon, but as to why they appeared in impossible matchups would require further research.

B. What challenges did we encounter?
    a. It was very challenging to pursue our original goal of predicting the winner of a Pokemon battle. We realized that we could barely predict better than chance despite detailed feature engineering and hyperparameter testing. We addressed this issue by pursuing the restricted teammate prediction model, which outputs actionable results.

C. How could we extend our solution with more time/resources?
    a. With more time and resources, I would like the opportunity to revisit our winner prediction model. I firmly believe that a larger sample size of Pokemon would unearth patterns in our current data points.
    b. We would also like to formalize and condense our existing restricted teammate prediction tool. We could take simple steps like preventing impossible predictions (Giratina + Giratina Origin). I would also like to further enhance functionality by comparing low ELO games to high ELO games, like is done on Pikalytics.

What did you learn from doing Part B?

A. What surprised us?
    a. It was surprising to see that the dataset with only the engineered features outperformed the dataset with most of the original data for the clustering task. While reducing dimensionality is likely to improve clustering performance, our concern was that by removing the original data, our model would be losing too much information. This indicates that our features were well designed and likely captured much of the useful variation needed for the clustering task while removing noise.
    b. This was the first time either of us used DBSCAN, and it was surprising to see how different and irregular visually the clusters looked from k-means clusters. Through the different parameter tunings and t-SNE visualizations, the DBSCAN clusters varied greatly in both size and shape. This ultimately allowed them to capture more of the niche variation between certain teams, and highlighted a limitation of simple k-means clustering, which is incentivised to form globular clusters of similar size–maybe not the best way to capture some of the more niche teams in our data.

B. What challenges did you encounter, and how did you respond to them?
    a. Silhouette scores were initially very low (under 0.15), indicating poor cluster division. This motivated us to craft a range of solutions, including grid searches of various parameters, engineering more features, and most importantly reducing dimensionality wherever possible. This was a lot of work, but ultimately gave us the clearly defined and interpretable clusters we were looking for.
    b. DBSCAN's need to check every point's distance to every other point gives it a time complexity of O(n^2). This is computationally expensive for large, high-dimensional data, especially when grid searching parameters. This initially led to several kernel crashes. Grid search was ultimately done on sampled

subsets of our data, and was only run on the full length data during the final label-generation step.

C. How could you extend your solution with more time/resources?

   a. With more computational resources and time, we could experiment clustering with different techniques. One approach we considered was binarizing all data (i.e. turning counts into presence or absence booleans) and rerunning DBSCAN with a binary measure such as hamming. It may also be possible to improve clustering with a completely different neural network based approach like DEC.
   b. With more time, visual analytics with 3D exploration could be built.

Ethical Considerations

I. What ethical issues could arise in providing a solution to **Part A**, and how can we address them?

   A. When originally developing the battle predictor model, we discussed the ethical implications of a user using this tool alongside Pokemon Showdown battles and whether it would give a player an unfair advantage. In order to understand what constitutes cheating, we took a look at Smogon's policies related to add-ons. Smogon operates in an "open-notes" format, where using external sources to aid gameplay is not only accepted but even expected.
   B. Another issue present in both solutions is re-identification of users. Some issues are obvious, like to ensure usernames are de-identified. However, there is a chance for re-identification within sets and playstyles. For example, the YouTube creator and 2016 World Champion Wolfe Glicke mentioned[3] that some users analyzed the order in which the 4 moves are set for your pokemon in a tournament setting. Wolfe learned that he tends to put moves like Protect and Fake Out in the first moveslot- and users were able to identify him amongst thousands of other anonymous applicants. What is interesting is that the order in which you place a pokemon's 4 moves has no effect on gameplay, but is rather an artifact of Wolfe's cognition and biases.

II. What ethical issues could arise in providing a solution to **Part B**, and how can we address them?

   A. As with Part A, reidentification of users could be an issue even if we take precautions like removing unique user tags. While cluster formulation itself is unlikely to be able to identify any single user, these clusters could reveal broad location-dependent metadata. For example, even on the same game update, countries (like Japan and the US) often have split, separate metagames. Therefore, a "Japanese" cluster and an "American" cluster may form- and paired with additional data could lead to user reidentification.

Statement of Work

Jackson Montuoro	Sean Fontaine

Backend Developer, Domain researcher, Head of Supervised Teammate Predictor Model, Logistic/KNN/RandomForest model Trainer, Ethics Analyst, report Writer	Backend Developer, Frontend Developer, Web Scraping, Data Preprocessor, UI Designer, Head of Unsupervised Cluster Modeling, Report Writer

References

Appendix A - Bibliography

[1] Shellnuts. "Other Tiers - Nintendo Cup '97 Hub." *Smogon Forums*, Smogon Forums, 17 Sept. 2024, www.smogon.com/forums/threads/nintendo-cup-97-hub.3682412/.

[2]Hemmingsen, Michael. 2023. "What Is a Metagame?" Sport, Ethics and Philosophy 18 (5): 452–67. doi:10.1080/17511321.2023.2250922.

[3] Glick, Wolfe. "I Went Undercover to Win a Huge Pokemon Tournament." *YouTube*, @WolfeyVGC, 26 June 2025, youtu.be/mpw-V9a1t38?si=zfnx5qYjyv2aG9CX&t=206. [3:26-3:41]

Appendix B - Data Schema

Data Source:

https://replay.pokemonshowdown.com/?format=gen9vgc2025regibo3

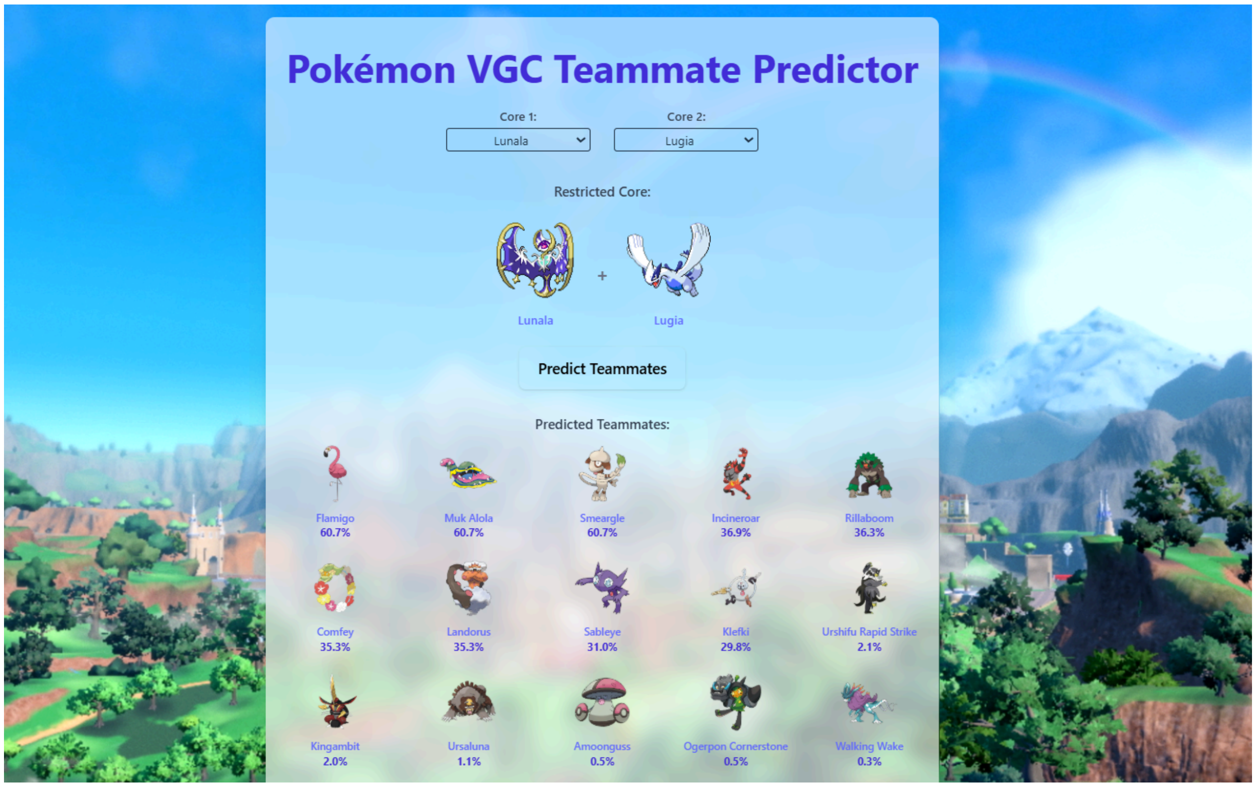
Reg I Battles Data Schema (Preprocessed Features)

Column Name	Var Type	Var Data Type	Example
p1_species_X	string	nominal	"Rillaboom"
p1_item_X	string	nominal	"Choice Specs"
p1_ability_X	string	nominal	"Intimidate"
p1_move_X_X	string	nominal	"Flamethrower"
p1_win	int	binary	1
p1_player	string	nominal	"Krusher77"
p1_lead_1	string	nominal	"Incineroar"
p1_lead_2	string	nominal	"Calyrex-Ice"
p1_species_X_type_1	string	nominal	"Grass"
p1_species_X_type_2	string	nominal	"Normal"
p1_species_X_total	int	discrete	530
p1_species_X_hp	int	discrete	110
p1_species_X_attack	int	discrete	130
p1_species_X_defense	int	discrete	80
p1_species_X_sp_atk	int	discrete	70

p1_species_X_sp_def	int	discrete	60
p1_species_X_speed	int	discrete	80
p1_num_fast_mons	int	discrete	3
p1_num_slow_mons	int	discrete	2
p1_num_bulky_mons	int	discrete	4
p1_num_frail_mons	int	discrete	2
p1_num_high_power_mons	int	discrete	5
p1_num_low_power_mons	int	discrete	2
p1_type_matchup_score	int	discrete	3
p1_team_cluster	int	nominal	5
p1_avg_total	float	continuous	545.9
p1_avg_hp	float	continuous	90.2
p1_avg_attack	float	continuous	80.8
p1_avg_defense	float	continuous	67.2
p1_avg_sp_atk	float	continuous	110.1
p1_avg_sp_def	float	continuous	105.3
p1_avg_speed	float	continuous	77.9
p1_type_diversity	int	discrete	7
p1_has_weather_setter_ability	bool	binary	True
p1_has_weather_setter_move	bool	binary	False
p1_count_speed_control_move	int	discrete	2
p1_has_priority_move	bool	binary	True
p1_has_terrain_setter_ability	bool	binary	True
p1_has_terrain_setter_move	bool	binary	False
p1_count_spread_move	int	discrete	3
p1_has_hazard_setter_move	bool	binary	True
p1_count_high_damage_move	int	discrete	2
p1_has_recovery_move	bool	binary	True
p1_has_trick_room_move	bool	binary	False
*p2 has all the same features	**X represents redundant variables		

Project Submission

spidersocks/poke-team-predictor



Pokémon VGC Teammate Predictor

Core 1: Lunala

Core 2: Lugia

Restricted Core: Lunala + Lugia

Predicted Teammates:

Flamigo 60.7%
Muk Alola 60.7%
Smeargle 60.7%
Incineroar 36.9%
Rillaboom 36.3%
Comfey 35.3%
Landorus 35.3%
Sableye 31.0%
Klefki 29.8%
Urshifu Rapid Strike 2.1%
Kingambit 2.0%
Ursaluna 1.1%
Amoonguss 0.5%
Ogerpon Cornerstone 0.5%
Walking Wake 0.3%

Web App
https://www.seanfontaine.dev/poke-team-predictor