CSI 4810 Project 2: Pokémon Type Classification

Jason Kauppila
School of Engineering and Computer Science
Oakland University
United States of America
jkauppila@oakland.edu

I. Introduction and Domain Knowledge

Pokémon is a famous video game franchise which is popular around the world, with the first entries in the series, known as the first generation of Pokémon games, being released in Japan in 1996 [1]. As of today, there have been nine generations of Pokémon games. Within the games, there exist fictional creatures known as Pokémon. Pokémon are divided into different species, with 1025 different Pokémon species having been discovered as of today [2]. One important concept within the fictional world of Pokémon is that of Pokémon types. There are 18 different regular Pokémon types: Fire, Water, Grass, Electric, Ice, Fighting, Poison, Ground, Flying, Psychic, Bug, Rock, Ghost, Dark, Dragon, Steel, Fairy, Normal [3]. Of these, each Pokémon will have either one or two types, which allows for 153 double-type combinations and 18 single types, for 171 unique possibilities for the typing of a Pokémon [3]. Notably, only 162 of these 171 unique combinations have currently appeared in the game series [3].

An important tool in the world of Pokémon is the Pokédex. This tool acts as a database that contains information about all of the Pokémon species in the game, including their typing [4]. One of the pieces of information about a Pokémon stored in the Pokédex is a short textual description containing some form of information about the Pokémon, like its habits. This information is called the Pokédex entry or sometimes the flavor text for a given Pokémon. One of the main goals within the Pokémon games is to complete the Pokédex, requiring the player to seek out information on every species of Pokémon in the games [5]. Thus, within the fictional world of Pokémon, it appears that being able to determine the type of a Pokémon is a very important ability, as it may aid in the completion of the Pokédex.

With such a large game franchise, it is important to worry about the reception of new Pokémon designs. In particular, the fifth generation of Pokémon games appear to receive a large amount of criticism about the designs of the new Pokémon in those games [6]. One factor that may influence the reception of new Pokémon designs is the ability for players of the Pokémon games to recognize the type of the Pokémon. With many Pokémon species being established already, it may be important that Pokémon types are portrayed in consistent ways. This concept can be extended outside of the Pokémon franchise though as elemental beasts are a common trope in many genres of fiction, such as fantasy [7]. Thus, when a writer or designer needs to create a new fictional creature, it would be useful to verify if the intended design and description of the creature aligns with the way that the elemental typing has been previously portrayed.

This provides motivation for the creation of a tool that is able to classify the elemental typing of fictional creatures given a short textual description of them. This paper uses publicly available information about various Pokémon species to build a classifier model which will perform this task. By combining an understand of the domain of Pokémon types with Python [8] and libraries such as Matplotlib [9], NumPy [10], Pandas [11], and scikit-learn [12], and Sentence Transformers [13] to visualize the Pokémon type data and implement classification models, it is possible to work towards the identification of the typing of elemental beasts, in particular Pokémon, from textual descriptions, hopefully allowing for a better understanding of the portrayal of these various types in fiction and a better ability to create new designs which hopefully will be positively received for adhering to the way that certain elemental types have been portrayed historically.

II. Dataset Analysis and Understanding

A. Data Collection

To gather the data for the construction of a classifier model that identifies the typing of elemental creatures, web scraping was performed on the website pokemondb.net [2]. To perform web scraping, Jupyter Notebook [14] with a Python [8] kernel was used as the programming environment. Additionally, the Pandas [11], Requests [15], and Beautiful Soup [16] libraries were used to obtain and manipulate the data. The following strategy was used for scraping data.

Using the Requests [15] library, an HTTP request was sent out for the webpage at [2] which lists all the Pokémon in the Pokédex. From there, Beautiful Soup [16] was used to extract hyperlinks to create a list of links to the webpages for each Pokémon species. Next, an HTTP request would be sent out with Requests [15] for each Pokémon species. Beautiful Soup [16] would be used to extract the Pokémon species name, Dex number, Pokédex entries, and Pokémon types. At this point, within the same species of Pokémon, duplicate Pokédex entries were removed without case sensitivity. Next, one hot encoding was applied to the Pokémon types, 1 representing the presences of a given type and 0 denoting the absence of the type. Finally, this data was converted into a Pandas [11] data frame and saved as a CSV file. The Pokémon names and Dex numbers were only scraped for the dataset in order to make it easier to navigate the dataset later on. Notably, the website only contained Pokédex entries for the first 1008 species of Pokémon, rather than all

1025 species of Pokémon. Thus, the dataset only contained information for 1008 different species of Pokémon.

B. *Data Cleaning*

Upon inspecting the CSV file containing the data collected for each Pokémon species, it was discovered that some Pokémon species had more than two types. According to knowledge on the domain of Pokémon types, it should not be possible for Pokémon to have more than two types. At this point, it was discovered that Pokémon within the same species can have multiple different forms, and sometimes these forms have different types [17]. For these Pokémon which take on multiple forms, it was necessary to manually review and adjust the type that corresponded to each Pokédex entry. In total, 76 different Pokémon species had their entries corrected in the dataset. For some Pokémon with multiple forms, the webpage containing information for that Pokémon did not specify which form of the Pokémon the Pokédex entry corresponded to. In this case, the typing of the default form of the Pokémon was used for these Pokédex entries.

Another observation from a brief analysis of the dataset was that some Pokémon of different species had the exact same Pokédex entries. In particular, these Pokémon were Buzzwole, Pheromosa, Xurkitree, Celesteela, Kartana, and Guzzlord. The Pokédex entry that was repeated between all of these Pokémon was "Although it's alien to this world and a danger here, it's apparently a common organism in the world where it normally lives" [18]. As these Pokémon have different types, and the text content does not seem relevant to the type, these 6 entries were removed from the dataset. Overall, this provided a worrisome sign that some Pokédex entries may be totally irrelevant to a Pokémon's type.

C. *Initial Data Visualization*

At this point, after cleaning the data as described above, there were 8724 Pokédex entries associated with Pokémon types in the dataset. Matplotlib [9] was then used to visualize the data in order to try to gain an understanding on how balanced the distribution of Pokémon types and the Pokédex entry length were.

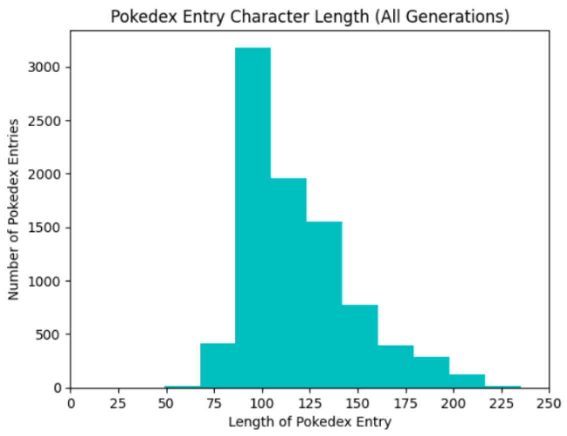

Fig. 1. A histogram of Pokédex entry character length across all nine Pokémon generations.

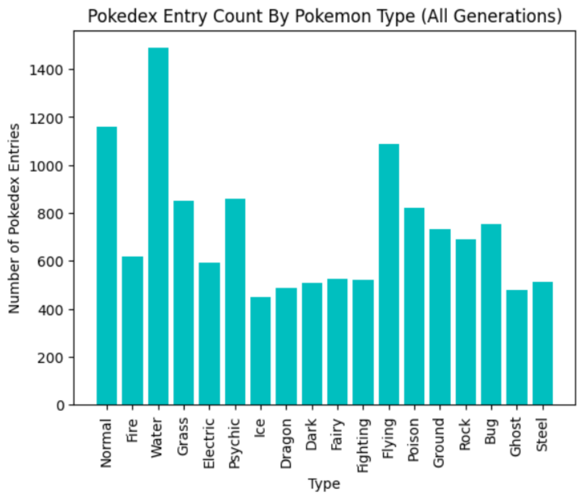

Fig. 2. A bar chart displaying the distribution of Pokédex entries for each Pokémon type across all nine Pokémon generations.

From Figure 1, it is observed that Pokédex entries appear to be between 100 and 125 characters on average. This information served to be valuable later when trying to determine a way to represent these Pokédex text entries as vectors.

Figure 2 shows that Pokédex entries are not uniformly distributed between Pokémon types. The Pokémon types which appear to have the most Pokédex entries are Water, Normal, and Flying. Meanwhile, the Pokémon types which appear to have the least Pokédex entries are Ice, Dragon, and Ghost. It is important to note that the Pokémon which first appeared in earlier generations of Pokémon games often received new Pokédex entries with each game, so this data does not serve to represent how common Pokémon types are overall between species. Some Pokémon types which have fewer entries, like Fairy and Steel were not even present in the first generation of Pokémon games, which appears to be reflected in this graph.

D. *Choosing A Subset Of The Data*

As it was discovered that there were only 8724 entries in the current dataset, it appeared that the size of the dataset would likely be too small to train a model to classify between 171 unique type combinations and obtain good results. This is likely only a temporary problem because it appears likely that more Pokémon games will be released in the future, meaning that more data will be available for this task. An additional point to consider is that from the exploration of the dataset during the cleaning of the data, it was observed that some Pokédex entries contained information that seemed irrelevant to the Pokémon's type. For Pokémon with multiple types, their Pokédex entry may describe one of their types more strongly than the other. This may result in confusion when building the model, possibly resulting in decision boundaries which are not very meaningful. For these reasons, it was decided to modify the problem at hand into a simpler problem so that meaningful information can be discovered from the dataset.

The problem was reformulated such that instead of classifying elemental beasts between 171 unique type combinations, the elemental beasts will instead be classified between three different types. The three types which were chosen for this were Grass, Water, and Fire. These types were chosen because they appear frequently throughout the game franchise and are less abstract than other Pokémon types such as Fairy, so it is hoped that the resulting model may generalize over to other fictious worlds outside of the world of Pokémon.

In order to construct the subset of the original dataset that is going to be used for this problem, all the entries for Pokémon having either Grass, Water, or Fire types were first extracted from the original dataset. Next, the secondary types for Pokémon with multiple types were removed. For Pokémon with two types which were either Grass and Water, Grass and Fire, or Water and Fire, their Pokédex entries were removed in order to keep the Pokémon types distinct and avoid confusion in case some Pokédex entries focused on one type more strongly than the other. Additionally, because the Pokédex entries would no longer correspond to multiple types, the one hot encoding of types was abandoned in favor of a single column with possibly class values of 0, 1, and 2 corresponding to the three Pokémon types.

At the end of this process, only 2908 data entries remained for the subset of Pokédex entries and their corresponding Pokémon types that would be used to train the classifier model. In order to better understand this subset, data visualization was done using Matplotlib [9].

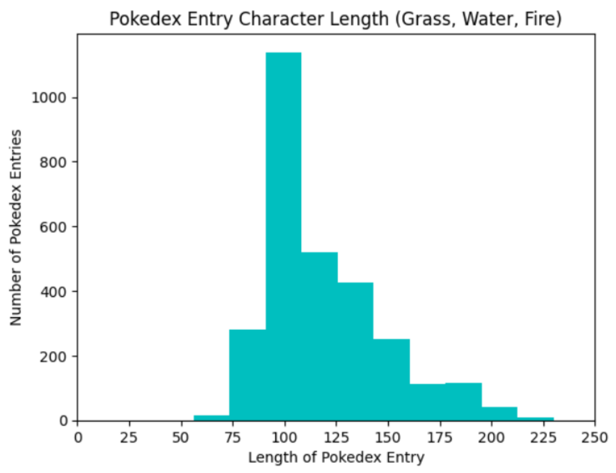

Fig. 3. A histogram of Pokédex entry character length across all nine Pokémon generations for Grass, Water, and Fire type Pokémon.

Figure 3 displays a trend similar to that present in Figure 1, with Pokédex entries appearing to be between 100 and 125 characters on average. Notably, the peak at 100 characters is higher in Figure 3 than in Figure 1, suggesting that there is less variance in the length of Pokédex entries in this subset compared to the original dataset.

Figure 4 shows that Pokédex entries are still not uniformly distributed between Pokémon types. The number of Pokédex entries for Water type Pokémon is roughly equal to the number of entries for Grass and Fire type Pokémon combined. Notably, these amounts appear similar to those present in Figure 2, meaning that the number of Pokémon who had overlapping types between Grass, Water, and Fire were few.

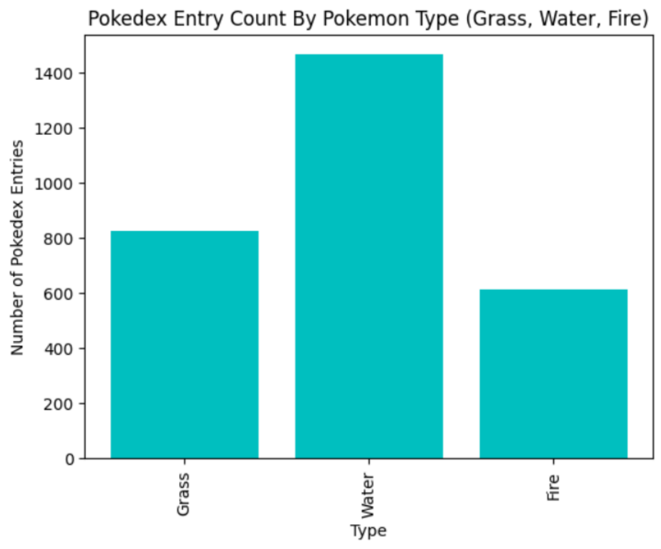

Fig. 4. A bar chart displaying the distribution of Pokédex entries for each of the three Pokémon types, Grass, Water, and Fire, across all nine Pokémon generations.

III. Data Transformation And Models Used

Pursuing the best results, three different classifier models were explored for this task. These three models are the Naïve Bayes classifier, the logistic regression classifier, and a neural network classifier. In addition to the previously mentioned process used to clean the data, it was necessary to represent the textual data as vectors in order to train the model. Although there are many ways to this, such as by using a bag-of-words model, it was decided that textual embeddings would be used.

A. Converting Text To Vectors

Using the Sentence Transformers library [13] with the pretrained model all-MiniLM-L6-v2, the Pokédex entries were converted into 384-dimensional float vectors to be used to train the classification models. According to the information about the text embedding model, it "is intended to be used as a sentence and short paragraph encoder" [19]. The length of the Pokédex entries which was explored earlier ended up being useful when trying to pick a text embedding model for this task. Although the length of an average Pokédex entry appears longer than a sentence, it appears to fall within the range of a short paragraph. Additionally, the length of the Pokédex entries is less than 256 characters, meaning it will be less than the limit of 256 word pieces which the model will truncate text at [19]. Therefore, it is not necessary to worry about processing the longer Pokédex entries so that meaningful information will not be lost during truncation. Thus, this text embedding model appeared suitable for the Pokédex text data in the dataset.

B. Naïve Bayes Classifier

The first type of model selected when trying to select a model for the classification problem was the Naïve Bayes classifier model. The first factor influencing the selection of this model was its popularity for text classification tasks, since the problem at hand is a text classification task. Additionally, this model is a relatively simple model compared to other models, not requiring the selection of multiple various hyperparameters. The idea was to use this model as a first guess in order to determine the best strategy for handling the data imbalance before moving on to models which will require more fine-tuning.

The Naïve Bayes classifier model was implemented with the scikit-learn's [12] GaussianNB model, which is included as part of the library. This Naïve Bayes classifier model was chosen of all of those in the scikit-learn library [12] because the other models available in the library appeared suitable for features with discrete values, which was not the case for the textual data after being encoded into 384-dimensional float vectors with the textual embedding model. For the construction of the model, no additional arguments were specified, meaning that the default settings in scikit-learn [12] for the GaussianNB model were used for the two arguments possible for this model, namely priors and var_smoothing.

C. Logistic Regression Classifier

The next model selected for the classification problem was the logistic regression classifier model. It was hoped that this model would be able to produce better results than the Naïve Bayes classifier model because it does not make the assumption that the features are independent of each other, as it is known that such an assumption rarely holds true. Additionally, with the implementation of the Naïve Bayes classifier chosen, it was assumed that the underlying distribution of the attributes was a normal distribution, which may not necessarily be true. Thus, by using a discriminative model like the logistic regression model, which tries to map input features directly to class labels, it is hoped that we will avoid some problems that may arise when we have to make an assumption about the form of the probability distribution in order to use generative models like the Naïve Bayes classifier.

Scikit-learn's [12] LogisticRegression linear model was used to implement the logistic regression classifier model, as it was included as part of the library. The Stochastic Average Gradient solver was the solver algorithm that was selected for the optimization of the model. In order to determine the best value for the regularization parameter, experimentation beginning at the default value of C=1. To ensure the repeatability of the results, the random state was set to 13. The maximum number of iterations for the optimization was set to 700 because the model was consistently able to converge within this number of iterations during the experimentation performed to determine the optimal value for C, the regularization parameter.

D. Neural Network Classifier

In order to try to improve upon the results of the previous two models, the next model chosen was a neural network classifier model. It was suspected that a neural network would be able to produce better results than the previous two classifiers because neural networks are able to determine complex decision boundaries not possible with linear models like the Naïve Bayes and logistic regression classifiers.

The neural network classifier model was implemented with the scikit-learn's [12] MLPClassifier model, which is included as part of the library. The activation function for the hidden layer that was used was the rectified linear unit function. As for the solver for the optimization of the model weights, Adam was used. The best number of neurons in the hidden layer was determined through experimentation, with only one hidden layer being considered for the model. The random state was set to 13 in order to ensure repeatable results with experimentation. The maximum number of iterations for the optimization was set to 700 because the model was consistently able to converge within this number of iterations during the experimentation performed to determine the optimal number of hidden layer neurons.

E. Addressing The Data Imbalance

As it was observed during the visualization of the dataset that the distribution of the Pokédex entries among the three Pokémon types is notably imbalanced, it seemed that it would be necessary to handle this imbalance in some way in order to avoid introducing bias into the model. Otherwise, it seemed possible that the model might simply be more likely to predict the majority class rather than accurately capturing the relationships present within the data, and thus be more likely to make incorrect predictions for the minority classes.

In order to learn more about ways to handle working with imbalanced datasets, an article from Data Science Horizons [20] was consulted. It was decided that both random undersampling and random oversampling would be experimented with. The results of using these techniques would be compared with the results of just using the imbalanced dataset in order to determine if these techniques are even necessary, and which of the two are better for this problem. In order to oversample the minority class, the RandomOverSampler from the Imbalanced-learn library [21] was used. As for undersampling the majority class, the RandomUnderSampler from the Imbalanced-learn library [21] was used.

IV. Experiments And Model Results

Various experiments were performed in order to determine both the best choice of parameters for the logistic regression classifier model and the neural network classifier model, which were described in the previous section, as well as whether techniques for handling the data imbalance would prove to be useful. For the parameters explored through experimentation, their values would be changed and the results recorded for both the imbalanced dataset and the dataset which uses techniques to address the imbalance. This methodology will demonstrate a general sense of how changes in the parameters affect the model's performance.

To evaluate the performance of the models, two performance metrics were used, namely the balanced accuracy score and the weighted F1-score. The implementation of these metrics was provided through the use of the scikit-learn library [12]. The reason why the balanced accuracy score was chosen opposed just a regular accuracy score was because the problem at hand has an imbalanced dataset. According to the scikit-learn library's [12] documentation, the balanced accuracy score is the average of each class's recall, making it suitable for dealing with multiclass classification problem with imbalanced datasets. As the F1-score takes into account both precision and recall, the weighted F1-score was chosen as another evaluation metric. A factor influencing this decision was the Data Science Horizons [20] article because it described F1-score as a good metric for problems with imbalanced datasets since accuracy can be a misleading metric.

To avoid the overfitting of the models to the training data giving misleading model performance results, an 80-20 stratified train-test split was performed on the dataset using the scikit-learn library [12], with the random state being set to 13 for repeatability. The 80% of the dataset was used for training while the 20% of the dataset was used for evaluation, with data being split in a stratified fashion among the classes in order to preserve the class imbalance between the splits, as recommended by [20]. When comparing the model results, the goal is for both of the performance metric values to be as high as possible, and the parameter value which achieves this result is taken to be the best value for that parameter for this multiclass classification problem.

A. *Naïve Bayes Classifier Evaluation*

As the Naïve Bayes classifier did not have any parameters that needed to be explored through experimentation, the experiments performed with the Naïve Bayes classifier model instead focused on the best method of handling the data imbalance. For all of the experiments, the random stat was set as 13, which is important for the reproducibility of the resulting training sets after random oversampling and random undersampling. The random oversampling and random undersampling were applied only to the training data split, not the test data split. The experimental results are displayed in Table 1.

TABLE 1

NAÏVE BAYES EXPERIMENTATION

Training Dataset	Balanced Accuracy Score	Weighted F1-Score
Random Oversampling	0.8471	0.8372
Random Undersampling	0.8265	0.8288
Unbalanced	0.8574	0.8428

These experiments display a result that was unexpected. The Naïve Bayes classifier model performed better when trained on the imbalanced base training dataset compared to the training datasets that used random oversampling of the minority class or random undersampling of the majority class. This occurs even when the metrics used to evaluate the performance of model were chosen with imbalanced datasets in mind, meaning that it is not likely that these results are simply misleading like a simple accuracy score would be due to the dataset being imbalanced. The results obtained from using the random undersampling of the majority class method for handling the imbalance in the training data were significantly poorer than the other results. The reason for this is likely that valuable information is being lost when data entries from the majority class are removed from the training dataset, suggesting that the size of the dataset is a bit limited considering the dimensionality of the problem. Although the model trained on training data that performed random oversampling of the minority class performed a bit worse than the original imbalanced dataset, the results were not too significantly different for this method to be discarded from consideration yet.

Overall, the best balanced accuracy score observed was 85.74% balanced accuracy. The best weighted F1-score observed was 0.8425. Both of these occurred with the original imbalanced training dataset.

B. *Logistic Regression Classifier Tuning And Evaluation*

Due to the results of the experiments from the Naïve Bayes classifier, only random oversampling as well as the imbalanced base training dataset were used in the following experiments. These experiments aim to determine the best value for the logistic regression classifier model's regularization parameter, C, which is the inverse of the regularization strength. For repeatability, all the random states were set to 13 for the model training as well as the dataset splits and random oversampling. As previously discussed, the solver algorithm used was the Stochastic Average Gradient solver, and the maximum number of iterations was set to 700. All the parameters not specified remained fixed and use the default values provided in the scikit-learn library's [12] implementation of the LogisticRegression model. The experimental results are displayed in Table 2 and Table 3.

TABLE 2

LOGISTIC REGRESSION EXPERIMENTATION

(UNBALANCED TRAINING DATASET)

C Value	Balanced Accuracy Score	Weighted F1-Score
1	0.8284	0.8509
2	0.8351	0.8562
4	0.8465	0.8619
8	0.8542	0.8601
16	0.8518	0.8652
32	0.8428	0.8566

TABLE 3

LOGISTIC REGRESSION EXPERIMENTATION

(RANDOM OVERSAMPLING TRAINING DATASET)

C Value	Balanced Accuracy Score	Weighted F1-Score
1	0.8401	0.8457
2	0.8476	0.8523
4	0.8424	0.8471
8	0.8533	0.8573
16	0.8454	0.8503
32	0.8372	0.8463

Similarly to what was observed with the Naïve Bayes classifier model, the logistic regression classifier model performed worse when trained on training data that underwent random oversampling of the minority class when it comes to the best scores achieved for the chosen performance metrics. Notably, the results between using random oversampling and the original imbalanced dataset were closer than they were with the Naïve Bayes classifier model.

The best balanced accuracy score achieved for the imbalanced training data was 85.42% balanced accuracy, which occurred with a C value of 8. The best weighted F1-score was 0.8652 and occurred with a C value of 16. Notably, the weighted F1-score for the C value of 8 take a slight dip, which likely results from the randomness present in the construction of the model as this value differs from the overall trend observed of the performance metric scores increasing up to when C is 8, and then decreasing.

The best balanced accuracy score achieved for the training data that underwent random oversampling was 85.33% balanced accuracy, which also occurred with a C value of 8. The best weighted F1-score was 0.8573, which also occurred with a C value of 8.

Overall, the same trends appear to be observed whether the imbalanced or oversampled training dataset is used. Both performance metrics demonstrate increasing score up to when the C value is 8 before decreasing as C continues to increase. From these experiments, it was concluded that the best C value to use with the logistic regression model for this problem is 8.

C. *Neural Network Classifier Tuning And Evaluation*

Once again, due to the results of the experiments from the Naïve Bayes classifier, only random oversampling as well as the imbalanced base training dataset were used in the following experiments. These experiments aim to determine the best value for the size of the hidden layer, in terms of number of neurons, for a neural network with a single hidden layer. For repeatability, all the random states were set to 13 for the model training as well as the dataset splits and random oversampling. As previously discussed, the solver algorithm used was the Adam optimizer, the maximum number of iterations was set to 700, and the activation function was the rectified linear unit function. All the parameters not specified remained fixed and use the default values provided in the scikit-learn library's [12] implementation of the MLPClassifier model. The experimental results are displayed in Table 4 and Table 5.

TABLE 4

NEURAL NETWORK EXPERIMENTATION

(UNBALANCED TRAINING DATASET)

Hidden Layer Size	Balanced Accuracy Score	Weighted F1-Score
300	0.8758	0.8845
400	0.8791	0.8847
500	0.8798	0.8881
600	0.8818	0.8931
700	0.8754	0.8861
800	0.8798	0.8880
900	0.8724	0.8828

TABLE 5

NEURAL NETWORK EXPERIMENTATION

(RANDOM OVERSAMPLING TRAINING DATASET)

Hidden Layer Size	Balanced Accuracy Score	Weighted F1-Score
300	0.8825	0.8898
400	0.8771	0.8863
500	0.8807	0.8881
600	0.8834	0.8932
700	0.8753	0.8846
800	0.8798	0.8880
900	0.8818	0.8898

The experiments with the neural network classifier model showed an interesting trend that was not observed with the previous two models. For the first time, the training dataset using random oversampling was observed to consistently outperform the original imbalanced training dataset. The difference between the two results remains marginal, perhaps due to the randomness present in the initialization of the model weights and other random elements involved in the construction of the model. Notably, random oversampling of the minority class results in more training samples that the model needs to be trained on, which appeared to add some extra overhead to the training process. It thus becomes questionable whether random oversampling is worth using due to the results only being better by a very small margin.

In terms of model performance metrics, the best balanced accuracy score achieved when using the imbalanced training data was 88.18% balanced accuracy, which occurred with a hidden layer size of 600. The best weighted F1-score was 0.8931 and also occurred with a hidden layer size of 600.

The best balanced accuracy score achieved with the training data that used random oversampling was 88.34% balanced accuracy, which once again occurred with a hidden layer size of 600. The best weighted F1-score was 0.8932 and similarly occurred with a hidden layer size of 600.

The same trends appeared to be present overall regardless of whether it was the imbalanced or oversampled training dataset being used. The performance metrics both generally demonstrated increasing scores up to when the hidden layer size was 600 before reaching a plateau, or perhaps even decreasing slightly as the size of the hidden layer continued to increase. From these experiments, it was concluded that the hidden layer size for the neural network classifier model for this problem is 600. Overall, the neural network classifier model performed notably better than the other two classifier models that were previously explored; thus, this was the final model chosen for the problem at hand.

V. Conclusion

For the text classification problem that aims to predict the type of an elemental beast from a textual description of that fictional beast, the final model chosen was the neural network classifier model with a single hidden layer containing 600 neurons, using a rectified linear unit activation function and Adam as the optimizer. This model was selected because it performed better on both performance metrics by around a margin of about 0.025 to 0.03 when compared to the Naïve Bayes classifier and the logistic regression model, which notably is an increase of a few percent for the performance metrics.

A. *Lessons Learned*

Many valuable lessons were learned as well as valuable experience gained while working on this project. Working on this project provided experience in web scraping, which was a domain that I previously had absolutely no experience with. Having to set up an arrangement to collect data for my own dataset for this problem was very meaningful as it simulated a situation where there is not a ready-made dataset for a specific problem. Additionally, it provided meaningful experience with cleaning a dataset because the original data scraped from the web had many errors which needed to be corrected manually. From this, I learned how time consuming it is to both collect data and clean it. Additionally, I learned that one should not flood servers with requests when scraping data, which resulted in the data collection process taking a long time. Despite this, cleaning the data was arguably more tedious for this particular project than collecting the data because after a lot of experimentation with code, it was possible to set up an arrangement to automatically collect the data over time, so the process was less interactive than cleaning the data.

Another thing learned was different techniques for working with imbalanced datasets. Firstly, though the concept was not completely new to me, it was meaningful to understand the importance of selecting a suitable performance metric for model evaluation, as accuracy can be misleading with imbalanced datasets. Both the techniques used in this paper, namely random oversampling of the minority class and random undersampling of the majority class, were new techniques learned for handling imbalanced dataset. Additionally, through the experiments performed, it was learned that using these techniques when training a model might result in worse performance metric scores compared to a model that was trained on the imbalanced dataset. Finally, regarding working with imbalanced datasets, it was learned that when performing a train-test split, the split should be done in a stratified fashion in order to maintain the imbalanced distribution of the target classes.

When comparing the model results, it was surprising to learn how well the Naïve Bayes classifier model performed for this problem. It was expected that the Naïve Bayes classifier model would perform notably worse than the other models because it made the assumptions that the features were independent of each other and that there was an underlying normal distribution for the features. It appears that despite these assumptions, it was possible for the Naïve Bayes classifier model to perform comparably with the other linear model explored in this paper, which was the logistic regression model. It was also surprising that the non-linear model that was used for this problem, namely the neural network classifier model, did not outperform these linear models by a larger margin. Thus, it is possible that this problem has a decision boundary which is close to linear in nature.

B. *Challenges, Mistakes, And Future Considerations*

Many challenges were encountered during almost every part of the process of working on this project. The first challenge encountered was the collection of data. Being completely new to web scraping, a lot of time was spent trying to learn how it was done and trying to set up a suitable arrangement to collect the data that I wanted to collect. The original aim for this project was going to be a classifier that would predict the star rating of a product from the text of the product review. A considerable amount of time was spent setting up a web scraping arrangement to collect data for this task; however, only 90 data entries were collected before the arrangement encountered a wall that it could not overcome. In particular, the issue faced was captchas. Although I attempted using user agents, a slightly more advanced web scraping technique, in order to get around these captchas, the arrangement ultimately did not succeed. Had it been accessible to me, the next technique that would have been explored would have been rotating proxies. However, since this was not an option for me during the time I was working on this project, it was necessary to change the direction of the project to what it currently is because 90 data entries did not seem sufficient to train a classifier model for the intended task.

The next challenge faced was the issue of the original problem space. The original intention was to treat the problem as a multilabel classification problem and thus build a multilabel classification model. Some of the concerns with this was whether it would be possible to enforce the rules that Pokémon can only have one or two different types. It may be possible that the model would select three or more type labels, which would be considered incorrect. If the model instead is designed such that it selects the two most likely types as the labels for a particular description, then the model is unable to correctly handle Pokémon with only a single type. As the Pokémon type labels underwent binary encoding as part of the data collection process, the plan was to use Jaccard similarity as a metric for evaluating the performance of the model, which should not necessarily have division by zero issues as every Pokémon is guaranteed to have a type. However, this plan was made prior to considering the data imbalance.

Very brief testing was done with the complete original dataset and it was quickly discovered that various different models all performed absolutely terribly. After some exploration, it was suspected that the issue resulted from the large number of possible class labels, the limited amount of training data available at the current time, and possibly low-quality data for Pokémon with two types, as the Pokédex entry may focus on one type more than the other. Thus, it was necessary to once again adjust the project aim to this time focus only on three different Pokémon type classes. As for why the number of types selected was three, it was believed that users using the model as a tool would already have a type combination in mind for the elemental beast when they wish to verify that its description is consistent with its typing. Having three types allows the user to select the first two types as the intended typing, and then the third type is used to verify that it is not mistaken as another single type. The third type can then be switched out and the verification process repeated in practice. Thus, three types appeared to be the number of types that would be the most manageable given the current dataset while still producing a useful model. In order to address the original problem, it may be necessary to explore more complex options, like creating an ensemble of these three type classifiers and somehow using them to classify text within the original 171 type combination space. Another avenue would be to collect more data from sources outside of the official Pokédex entries, such as entries from Pokémon fan-games or data from other fictional worlds outside of Pokémon. Unfortunately, due to time constraints, this project was unable to explore these options and they remain avenues for future research.

A third challenge faced was trying to deal with the data imbalance. As previously mentioned, it was discovered early on that undersampling methods which remove entries from the training dataset provided poor results for this problem. It is suspected that this was likely the case due to the limited amount of training data available. As for random oversampling, it only produced very marginally better results with the neural network classifier model. Frankly, this finding was not very satisfying because the experiments basically suggested that just using the original dataset is the best, even though it is imbalanced and may introduce bias towards the majority class into the model. Thus, a future direction to explore would be more advanced techniques for handling imbalanced datasets, such as synthetic minority oversampling technique (SMOTE) or Tomek links.

One thing that might be considered a mistake that occurred was with trying to build the neural network classifier model. It is said that the number neurons in the hidden layer should be between the size of the input layer and the output layer [22]; however, the experiments found 600 neurons to be the best size of the hidden layer, even though the size input layer should only be 384, which is the dimensionality of the text descriptions after they have been encoded into vectors using the textual embedding model. Thus, there appears to be a disconnect between the rule of thumb and the results of the experiments. This might suggest a mistake is present in the design of the neural network architecture. For example, perhaps there should have been more hidden layers, or perhaps a different activation function should have been used. Since it was determined that out of all the models explored in this paper, the neural network classifier model performed the best, these considerations appear to be a good direction for future research on this problem.

Some other future directions when it comes to model building for this elemental beast type classification problem could be the use of ensemble methods. In particular, it might be worth experimenting with using different ways of representing the textual data as vectors between the weak learners for the ensemble method. For example, some weak learners might use a bag-of-words model, while others could use various different textual embedding models. Having to encode the textual data so many different ways might result in significantly longer training and inference time, but may be worth the extra overhead if this approach is able to produce better results. One last direction might be to manually review all of the thousands of entries in the dataset and remove those with text that is irrelevant to the type of a Pokémon. However, this would be a lengthy process and the reviewer may accidentally introduce some bias into the dataset by removing entries which may actually hold meaningful information that will be lost when discarded.

Appendix A

A Gradio demo application [23] was used to implement and demonstrate the final model for this text classification problem. The parameters of the models in the demo application are set to be the same as those in the experiments. For the logistic regression model, the value for the regularization parameter C was set to be 8, which was the value that was found to be the best through experimentation. For the neural network classifier model, the number of neurons present in the single hidden layer was set to 600 as this amount that was found to be the best through experimentation. The source code used for this project as well as the demo application can be found as Jupyter Notebook [14] files at the provided GitHub repository. The notebook files are designed to have all their cells executed from top to bottom. As the demo application also includes the option to use the Naïve Bayes classifier and logistic regression classifier models, note that the model chosen to be the best for this problem was the neural network classifier model, which produced the best results. In order to increase the usefulness of the demo application, it is designed so that the user can pick any three Pokémon types between which a textual description will be classified. Although only the three types of Grass, Water, and Fire are discussed in this paper, the same data subset preprocessing procedure followed for the Grass, Water, and Fire types is followed for the three types chosen by the user in the demo application. Notably, random oversampling is not used for the training data for the models in the demo application because it would add extra overhead, taking away from the user experience for only marginally better model results. Overall, the application is useful in practice because the user is able to cycle through the type options to validate if the description of a conceptual elemental beast adheres to an intended double-type combination. The demo application is shown in Figure 5.

https://github.com/OrangeJuiceNoPulp/CSI4810Project2

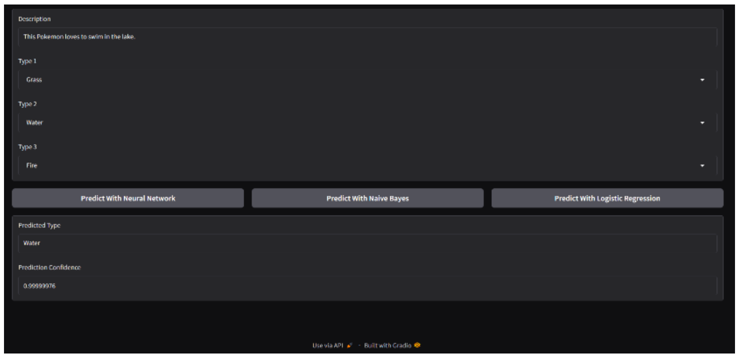
Description: This Pokemon loves to swim in the lake.
Type 1: Grass
Type 2: Water
Type 3: Fire
Predicted Type: Water
Prediction Confidence: 0.99999976
Predict With Neural Network
Predict With Naive Bayes
Predict With Logistic Regression
Use via API · Built with Gradio

Fig. 5. The demo application predicting "Water" type for an elemental beast with the description "This Pokemon loves to swim in the lake."

References

[1] M. Ray and The Editors of Encyclopedia Britannica, "Pokemon: Electronic game," Britannica.com. Accessed: Dec. 10, 2024. [Online.] Available: https://www.britannica.com/topic/Pokemon-electronic-game

[2] Pokemon Database, "Complete Pokemon Pokedex," PokemonDB.net. Accessed: Dec. 10, 2024. [Online.] Available: https://pokemondb.net/pokedex/all

[3] "Type," Bulbagarden.net. Accessed: Dec. 10, 2024. [Online.] Available: https://bulbapedia.bulbagarden.net/wiki/Type

[4] "Pokedex," Bulbagarden.net. Accessed: Dec. 10, 2024. [Online.] Available: https://bulbapedia.bulbagarden.net/wiki/Pok%C3%A9dex

[5] "Pokedex Completion," Serebii.net. Accessed: Dec. 10, 2024. [Online.] Available: https://www.serebii.net/scarletviolet/pokedexcompletion.shtml

[6] A. Cooney, "Why do people criticize Generation 5 from 'Pokemon' as the worst region of them all?," Quora.com. Accessed: Dec. 10, 2024. [Online.] Available: https://www.quora.com/Why-do-people-criticize-Generation-5-from-Pok%C3%A9mon-as-the-worst-region-of-them-all

[7] "Elemental Embodiment," TV Tropes.org. Accessed: Dec. 10, 2024. [Online.] Available: https://tvtropes.org/pmwiki/pmwiki.php/Main/ElementalEmbodiment

[8] *Python Version 3.12.5*. (2024). Python Software Foundation. Accessed: Dec. 10, 2024. [Online]. Available: http://www.python.org

[9] *Matplotlib*. (2024). The Matplotlib development team. Accessed: Dec. 10, 2024. [Online]. Available: https://matplotlib.org/

[10] *NumPy*. (2024). NumPy team. Accessed: Dec. 10, 2024. [Online]. Available: https://numpy.org/

[11] Pandas. (2024). NumFOCUS, Inc. Accessed: Dec. 10, 2024. [Online]. Available: https://pandas.pydata.org/

[12] *Scikit-Learn*. (2024). Scikit-Learn.org. Accessed: Dec. 10, 2024. [Online]. Available: https://scikit-learn.org/stable/

[13] *Sentence Transformers*. (2024). SBERT.net. Accessed: Dec. 10, 2024. [Online.] Available: https://www.sbert.net/index.html

[14] *Jupyter Notebook*. (2024). Jupyter.org. Accessed: Dec. 10, 2024. [Online.] Available: https://jupyter.org/

[15] K. Reitz et al., *Requests*. (2024). ReadTheDocs.io. Accessed: Dec. 10, 2024. [Online.] Available: https://requests.readthedocs.io/en/latest/

[16] L. Richardson, *Beautiful Soup*. (2024). ReadTheDocs.io. Accessed: Dec. 10, 2024. [Online.] Available: https://beautiful-soup-4.readthedocs.io/en/latest/

[17] "Form," Bulbagarden.net. Accessed: Dec. 10, 2024. [Online.] Available: https://bulbapedia.bulbagarden.net/wiki/Form

[18] Pokemon Database, "Buzzwole," PokemonDB.net. Accessed: Dec. 10, 2024. [Online.] Available: https://pokemondb.net/pokedex/buzzwole

[19] *Sentence Transfomers all-MiniLM-L6-v2*. (2024). Hugging Face.co. Accessed: Dec. 10, 2024. [Online.] Available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[20] Team DSH, "Handling imbalanced datasets in scikit-learn: Techniques and best practices," Data Science Horizons.com. Accessed: Dec. 10, 2024. [Online.] Available: https://datasciencehorizons.com/handling-imbalanced-datasets-in-scikit-learn-techniques-and-best-practices/

[21] *Imbalance-learn*. (2024). Imbalanced-learn.org. Accessed: Dec. 10, 2024. [Online.] Available: https://imbalanced-learn.org/stable/

[22] User1865345 and doug, "How to choose the number of hidden layers and nodes in a feedforward neural network?," StackExchange.com. https://stats.stackexchange.com/questions/181/how-to-choose-the-number-of-hidden-layers-and-nodes-in-a-feedforward-neural-netw

[23] *Gradio*. (2024). Gradio.app. Accessed: Dec. 10, 2024. [Online.] Available: https://www.gradio.app/