Pokémon Fight Winner Prediction

Name:	**Ayush Yadav**
Roll No.:	19070
Institute Name:	IISER Bhopal
Program:	DSE
Problem Release date:	January 13, 2022
Date of Submission:	April 24, 2022

Introduction

Pokémon (an abbreviation for Pocket Monsters in Japan) is a Japanese media franchise. It is centered on fictional creatures called "Pokémon". In Pokémon, humans, known as Pokémon Trainers, catch and train Pokémon to battle other Pokémon for sport. There are currently 905 Pokémon species.[1]

This project aims to develop and use an effective supervised ML classification technique to predict the winner of a random battle between two Pokémon based on their features like Attack, Defense, Speed, etc.

The Dataset

pokemon_data.csv

The size of this dataset is 800 × 12. It contains the data of 800 Pokémon over 12 features as follows:

Feature	DataType	Feature	DataType	Feature	DataType
#	int64	HP	int64	Sp. Def	int64
Name	object	Attack	int64	Speed	int64
Type 1	object	Defense	int64	Generation	int64
Type 2	object	Sp. Atk	int64	Legendary	bool

combats.csv

The size of this dataset is 50000 × 3. This dataset contains the data of fights between 50000 sets of two Pokémon each and the winner of each battle. The Pokémon listed in the First_pokemon column will attack first. Description is as follows:

Feature	DataType	Description
First_pokemon	int64	Serial No. (#) of First Pokemon
Second_pokemon	int64	Serial No. (#) of Second Pokemon
Winner	int64	Serial No. (#) of Winner Pokemon

test.csv

The size of this dataset is 10000 × 2. This dataset has the same format as the **combats.csv** dataset, the only difference being that the winner column of this table has to be predicted by the ML framework and appended to the table after all the predictions are made.

Methods

Data Preprocessing

Dropping insignificant columns

We do not need the *Name* column and hence, we can drop it from the table.

Handling Categorical Values

Since any ML algorithm works on numerical data an not on categorical data, thus we have to convert all the categorical values of the data into numerical values.

- *Legendary* column's values are converted from boolean to 0 and 1 for False and True respectively.
- *Type 1* and *Type 2* columns' values are converted to binary using the one hot encoding method using *pandas.get_dummies*[2] function. After this conversion, we can drop the *Type 1* and *Type 2* columns as well.

The processed data now contains 800 rows and 27 columns.

Training Data Preparation

The processed dataset is now merged with the ***combats.csv*** dataset. Another column *Winner_1st* is added to this merged dataset to specify if the *First_pokemon* is the winner or not. This column has binary values.Also, we can again remove insignificant columns.
Our prepared dataset now has 50000 rows and the following 53 columns:

HP_1st	Fire_1st	Psychic_1st	Attack_2nd	Water_2nd	Rock_2nd
Attack_1st	Water_1st	Rock_1st	Defense_2nd	Bug_2nd	Ghost_2nd
Defense_1st	Bug_1st	Ghost_1st	Sp. Atk_2nd	Normal_2nd	Ice_2nd
Sp. Atk_1st	Normal_1st	Ice_1st	Sp. Def_2nd	Poison_2nd	Dragon_2nd
Sp. Def_1st	Poison_1st	Dragon_1st	Speed_2nd	Electric_2nd	Dark_2nd
Speed_1st	Electric_1st	Dark_1st	Generation_2nd	Ground_2nd	Steel_2nd
Generation_1st	Ground_1st	Steel_1st	Legendary_2nd	Fairy_2nd	Flying_2nd
Legendary_1st	Fairy_1st	Flying_1st	Grass_2nd	Fighting_2nd	Winner_1st
Grass_1st	Fighting_1st	HP_2nd	Fire_2nd	Psychic_2nd	-

The *Winner_1st* column contains the labels for our prepared training dataset, hence we drop the column from the training dataset and assign it to a label variable.

Train-Test Split

The dataset is randomly split into training and testing sets by a factor of 0.8, i.e., 80% data (40000 rows) is selected for training and 20% data (10000 rows) is selected for testing the model.

Models Used for Training

This problem is clearly a classification problem as we have two discrete classes (0 and 1). Hence, the main classification algorithms used for this problem are:

- Logistic Regression
- K-Nearest Neighbors
- Multinomial Naive Bayes
- Decision Tree Classifier
- Random Forest Classifier
- Support Vector Classifier

Hyperparameter Tuning

Hyperparameters are special parameters that are used to tune the behaviour of a machine learning algorithm. These are initialized before the training and supplied to the model, whereas normal parameters are the values that the model learns during training.
Scikit-learn's[3] *GridSearchCV* is one of the most basic hyper parameter tuning techniques. To tune models, all feasible permutations of the hyperparameters for a specific model are used. The performance of the model is evaluated on all the combinations of the hyper-parameters and the best performing ones are chosen. The hyperparameters for the used models are as follows:

Model	**Hyperparameters**	**Default Values**
LogisticRegression()	solver, random_state	['lbfgs', None]
KNeighborsClassifier()	n_neighbors, weights	[5, 'uniform']
MultinomialNB()	alpha	[1.0]
DecisionTreeClassifier()	criterion, max_features, max_depth, ccp_alpha	['gini', None, None, 0.0]
RandomForestClassifier()	criterion, max_depth	['gini',None]
SVC()	kernel, C	['rbf',1.0]

The complete project can be found here.

Experimental Analysis

Firstly, the baseline versions of all the models are trained on the dataset. Then, the models' hyperparameters are tuned using grid search and then the tuned models are trained on the prepared dataset and the model with highest performance metrics is used for the prediction.
In this problem, the performance metrics used are accuracy, macro-averaged precision, recall, and f-measure since this is a classification problem. These measures are described as:

$$Accuracy = \frac{Correctly\ predicted\ data\ points}{Total\ number\ of\ data\ points} = \frac{\text{TP+TN}}{\text{TP+FP+FN+TN}}$$

$$Precision = \frac{Correctly\ predicted\ positive\ points}{Total\ predicted\ positive\ points} = \frac{\text{TP}}{\text{TP+FP}}$$

$$Recall = \frac{Correctly\ predicted\ positive\ points}{Total\ actual\ positive\ points} = \frac{\text{TP}}{\text{TP+FN}}$$

$$f\_measure = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

Logistic Regression (solver='saga', random_state=1)

Model	Accuracy	Precision	Recall	f1_score
Baseline	0.89	0.89	0.89	0.89
Tuned	0.89	0.89	0.89	0.89

Table 1: Performance metrics

K-Nearest Neighbors (n_neighbors=50, weights='distance')

Model	Accuracy	Precision	Recall	f1_score
Baseline	0.87	0.87	0.87	0.87
Tuned	0.89	0.89	0.89	0.89

Table 2: Performance metrics

Multinomial Naive Bayes (alpha=0.0)

Model	Accuracy	Precision	Recall	f1_score
Baseline	0.79	0.79	0.79	0.79
Tuned	0.79	0.79	0.79	0.79

Table 3: Performance metrics

Decision Tree Classifier (entropy='gini', max_depth=10, ccp_alpha=0)

Model	Accuracy	Precision	Recall	f1_score
Baseline	0.94	0.94	0.94	0.94
Tuned	0.94	0.94	0.94	0.94

Table 4: Performance metrics

Random Forest Classifier (entropy='gini', max_depth=50, n_estimators=250)

Model	Accuracy	Precision	Recall	f1_score
Baseline	0.94	0.94	0.94	0.94
Tuned	0.96	0.96	0.96	0.96

Table 5: Performance metrics

Support Vector Classifier (kernel='rbf', C=300)

Model	Accuracy	Precision	Recall	f1_score
Baseline	0.91	0.91	0.91	0.91
Tuned	0.93	0.93	0.93	0.93

Table 6: Performance metrics

Discussions & Future Plans

This analysis model can perform better with the following improvements:

- Integration with the actual game software, where Pokemon can be trained, their ratings changed & a real-time algorithm continuously deciding the winner for all possible match-ups.
- Allowing users to build their own Pokemon which can be used to fight other normal Pokemon, and the results of these fights can be derived by our machine learning model.

In the longer run, this project can be expanded in several ways such as:

- Making similar models for other equally popular gaming franchises like Beyblade, Marvel Superheroes, etc.

References

[1] Wikipedia contributors. Pokémon — Wikipedia, the free encyclopedia. `https://en.wikipedia.org/w/index.php?title=Pok%C3%A9mon&oldid=1071862894`, 2022. [Online; accessed 17-February-2022].

[2] Wes McKinney. Data Structures for Statistical Computing in Python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 56 – 61, 2010.

[3] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.