CS584 – machine learning

fall 2016

Legendary Pokemo Prediction

Group Members: Kai Yao

Table of Contents

[Task 2](#_Toc467172373)

[Dataset 2](#_Toc467172374)

[Data source 2](#_Toc467172375)

[Target variable 2](#_Toc467172376)

[Features 2](#_Toc467172377)

[Data size 2](#_Toc467172378)

[Preprocessing 2](#_Toc467172379)

[Visualization 2](#_Toc467172380)

[Target 2](#_Toc467172381)

[Features 3](#_Toc467172382)

[Evaluation 3](#_Toc467172383)

[Performance Measure 5](#_Toc467172384)

[Classifiers 5](#_Toc467172385)

[Evaluation Strategy 6](#_Toc467172386)

[Performance Results 6](#_Toc467172387)

[Top Features 6](#_Toc467172388)

[Discussion 6](#_Toc467172389)

[Interesting/Unexpected Results 7](#_Toc467172390)

[Contributions of Each Group Member 7](#_Toc467172391)

[Conclusion 7](#_Toc467172392)

[References 7](#_Toc467172393)

Legendary Pokemon Prediction

Group Members: Kai Yao

# Task

# I am going to use the dataset that I downloaded from kaggle. This is a pokemon dataset with attributes like the HP, attack, attack speed, defense speed, type, generation of the pokemon. I decided to use these attributes to predict the Legendary for the pokemons. In general, the higher HP or attack a pokemon has, the higher its Legendary Degree is. We can use project to verify this assumption.

# Dataset

## This dataset contains 801 pokemons with attributes like name, ids, attack, type 1, type 2, speed and so on.

## Because there are some missing values for “Type 2” attribute, so I just use python to clean the data and add “?” for missing values.

## Data source

I collect the data from Kaggle, There is no additional data and I didn’t manually label any data.

## Target variable

The target variable is “Legendary”

## Features

Input features are Total, HP, Attack, Defense, Sp. Atk, Sp. Def, Speed, Generation, 8 attributes in total.

## Data size

There are 801 instances in total.

# Preprocessing

All the missing values are labeled as “?”

# Visualization

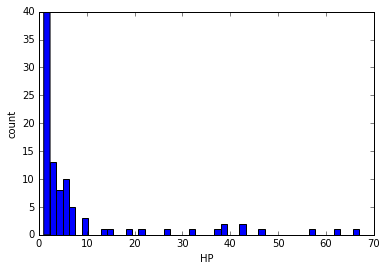
## Target

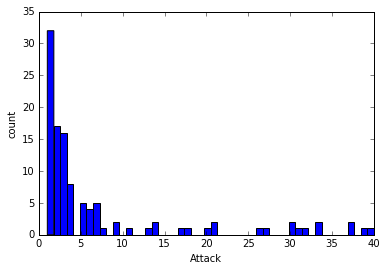
False: 735, True:65

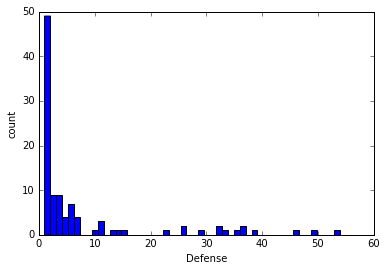
## 

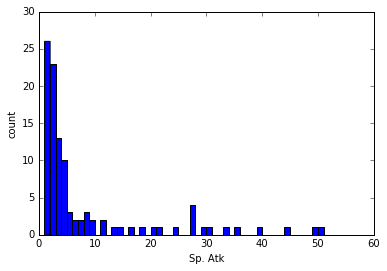
## Features

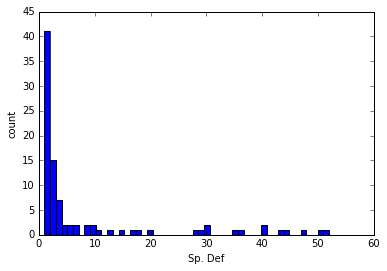
# 

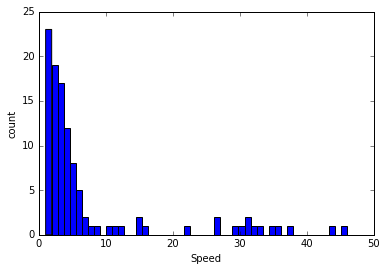


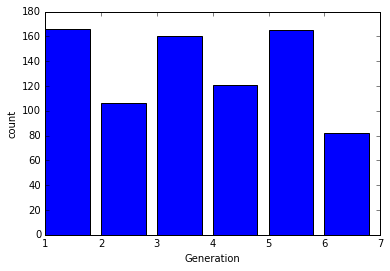












# Evaluation

## Performance Measure

I choose accuracy and F1 as my performance measure, because if we do majority guess, then the accuracy will be 0.918, so accuracy cannot tell us anything, so we also use F1 as performance measure. So that we can focus on the instances that are predicted as True and are actually True.

## Classifiers

support vector machine:

C = [1., 20., 25., 50., 100.]

kernels = ['rbf', 'linear', 'poly']

degrees = [2, 3, 4, 5, 6], this is just for polynomial kernel

I use these parameters to find out if C have an impact on accuracy and which combination of settings can give me the best performance.

logistic regression:

reg = ['l1', 'l2']

solver = ['liblinear', 'lbfgs', 'sag', 'newton-cg']

C = [1., 20., 25., 50., 100.]

intercept = [True, False]

I tried these parameter settings combination because I want to see if L1 regularization can have a better performance than L2 regularization.

## Gaussian Naive Bayes:

Because all the attributes are continuous, so I can use Gaussian Naive Bayes to do classification.

## Evaluation Strategy

I do the cross-validation, because I don’t have separate training and testing dataset. And if I use cross-validation, then I can use the whole dataset as training set and test set.

## Performance Results

|  |  |  |
| --- | --- | --- |
| Model | Parameters | Performance |
| Baseline | Random Guess | Acc: 0.473 F1: 0.102 |
|  | Majority guess | Acc: 0.918 |
| Logistic regression | penalty = l1, C=20 | Acc:0.943 F1: 0.508 |
|  | penalty = l2, C=20 | Acc:0.943 |
|  | … | … |
| SVM | kernel:=“rbf”, C = 100 | Acc:0.943 F1: 0.544 |
|  | kernel:=“rbf”, C = 50 | Acc:0.942 |
|  | kernel:=“rbf”, C = 1 | Acc:0.941 |
|  | … | … |
| Gaussian NB |  | Acc:0.932 F1: 0.627 |

## Top Features

## Based on information gain. The top features are:

## 'Total'

## ' Generation'

## 'HP‘

## 'Sp. Atk'

## Discussion

The best classifier doesn’t have the best accuracy but have the best F1 over all the other classifier. I expected that the best model has both the best accuracy and the best F1 over all the other classifier. I think that the reason for the accuracy is worse than other model is that Gaussian Naive Bayes have a higher bias and low variance. In these classifiers, the logistic regression has worse performance, even though it has a high accuracy, but its F1 is the much worse than other model, so even if we classify one instance as True, we cannot be confident that this instance is actually true, that is the main reason that I pick F1 as my performance measure to classification problem.

# Interesting/Unexpected Results

# 'Total', 'HP', 'Attack', 'Defense', 'Sp. Atk', 'Sp. Def', 'Speed', 'Generation'

# Here are two pokemons that are legendary:

# [ 770. 100. 180. 160. 150. 90. 90. 3.] 1 1

# [ 680. 105. 150. 90. 150. 90. 95. 3.] 1 1

Here are two pokemons that are not predicted as legendary but actually legendary

[ 600. 50. 150. 50. 150. 50. 150. 3.] 1 0

[ 600. 50. 180. 20. 180. 20. 150. 3.] 1 0

From those two pairs of examples we can see that the reason for Gaussian Naive Bayes misclassifying the pokemons is that it considers 'Total', ' Generation', 'HP‘, 'Sp. Atk' as the most important features and if 'Total', ' Generation', 'HP‘, 'Sp. Atk' are high, then it will be classified as legendary pokemons, but actually that is not the case for some pokemons.

# Conclusion

From this project I can see that most legendary pokemons have high ‘total’, ' Generation', 'HP‘ and 'Sp. Atk' values, but some legendary pokemons don't have that high values, so they are misclassified. This fact implies that the legendary pokemons may not related to their values in ‘total’, ' Generation', 'HP‘ and 'Sp. Atk'. So maybe there are other factors that lead to legendary pokemons.

# References

<http://scikit-learn.org/0.17/modules/classes.html>