Hong Kong University of Science and Technology
COMP 4211: Machine Learning
Spring 2020

Programming Assignment 1
Due: 26 March 2020, Thursday, 11:59pm

# Task 1: Calculating the Win Rate

**[Q1] When calculating the win rates of the Pokemons, you may notice that some of them have not participated in any battle. Explain how you deal with them.**

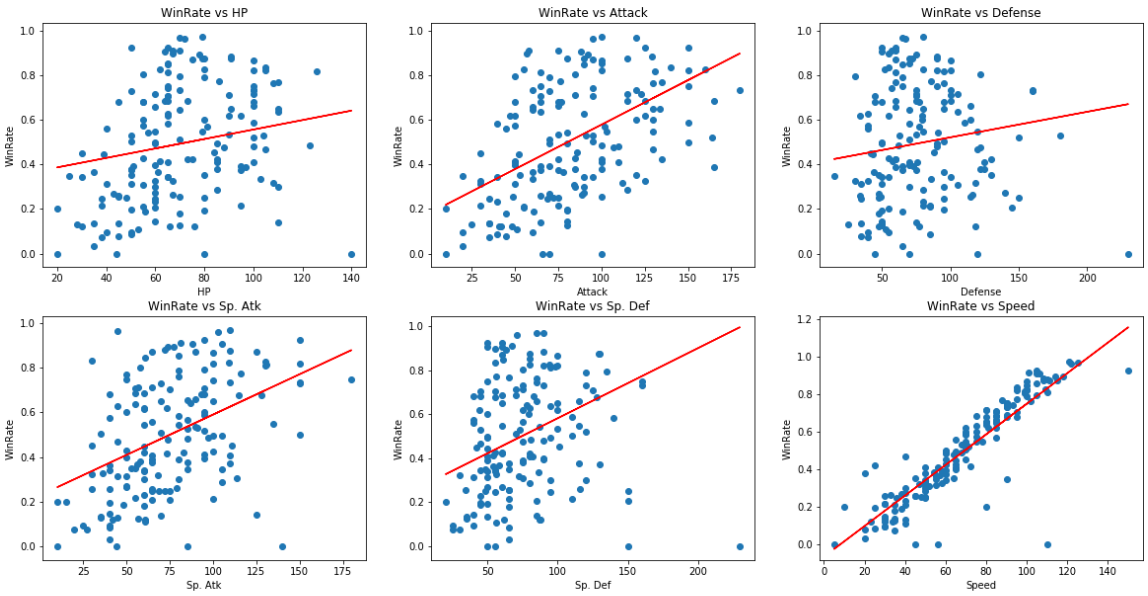
I simply treat their win rate as 0%.

# Task 2: Finding the Most Correlated Feature using Linear Regression

### [Q2] Report the validation $R^2$ score of each model to evaluate its prediction performance.

| | |
|---|---|
| R2 socre of HP: | 0.093010 |
| R2 socre of Attack: | 0.215038 |
| R2 socre of Defense: | -0.015525 |
| R2 socre of Sp. Atk: | 0.204860 |
| R2 socre of Sp. Def: | -0.015475 |
| R2 socre of Speed: | 0.805541 |

### [Q3] After training the models with the training set, use them to make prediction on the validation set. Then, plot the regression line and the data points of the validation set for each of the six models.




### [Q4] By looking at the regression lines of the six plots, find the feature that is most correlated to the win rate. Explain how you find it.

The most correlated feature is Speed. According to the graph above, we can observe that Speed are positively proportional to WinRate, and the regression line perform same as the scatter points.

# Task 3: Legendary Pokemon Classification using Logistic Regression and Single-hidden-layer Neural Networks

**[Q5] Report the model setting, training time, and performance of the logistic regression model. Since the solution found may depend on the initial weight values, you are expected to repeat each setting multiple times (e.g., three times) for the same hyperparameter setting and report the mean and standard deviation of the training time, accuracy, and F1 score for each setting.**

```
SGDClassifier(alpha=0.1, average=False, class_weight=None, early_stopping=False,
              epsilon=0.1, eta0=0.1, fit_intercept=True, l1_ratio=0.15,
              learning_rate='adaptive', loss='log', max_iter=500,
              n_iter_no_change=5, n_jobs=None, penalty='l2', power_t=0.5,
              random_state=0, shuffle=True, tol=0.001, validation_fraction=0.1,
              verbose=1, warm_start=False)
Mean Training Time:  0.03627816836039225
SD of Training Time:  0.008192214932326997
Mean Accuracy:  0.9187499999999998
SD of Accuracy:  1.1102230246251565e-16
Mean F1 score:  0.48
SD of F1 score:  0.0
```

**[Q6] Report the model setting, training time, and performance of the neural networks for each value of H. You are also expected to repeat each setting multiple times for the same hyperpa- rameter setting and report the mean and standard deviation of the training time, accuracy, and F1 score for each setting.**

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(64,), learning_rate='constant',
              learning_rate_init=0.001, max_fun=15000, max_iter=500,
              momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
              power_t=0.5, random_state=4211, shuffle=True, solver='sgd',
              tol=0.0001, validation_fraction=0.1, verbose=False,
              warm_start=False)
Mean Training Time:  0.5669942696889242
SD of Training Time:  0.06396841871022588
Mean Accuracy:  0.86875
SD of Accuracy:  0.0
Mean F1 score:  0.4615384615384615
SD of F1 score:  5.551115123125783e-17
```

## [Q7] Compare the training time, accuracy and F1 score of the logistic regression model and the best neural network model.

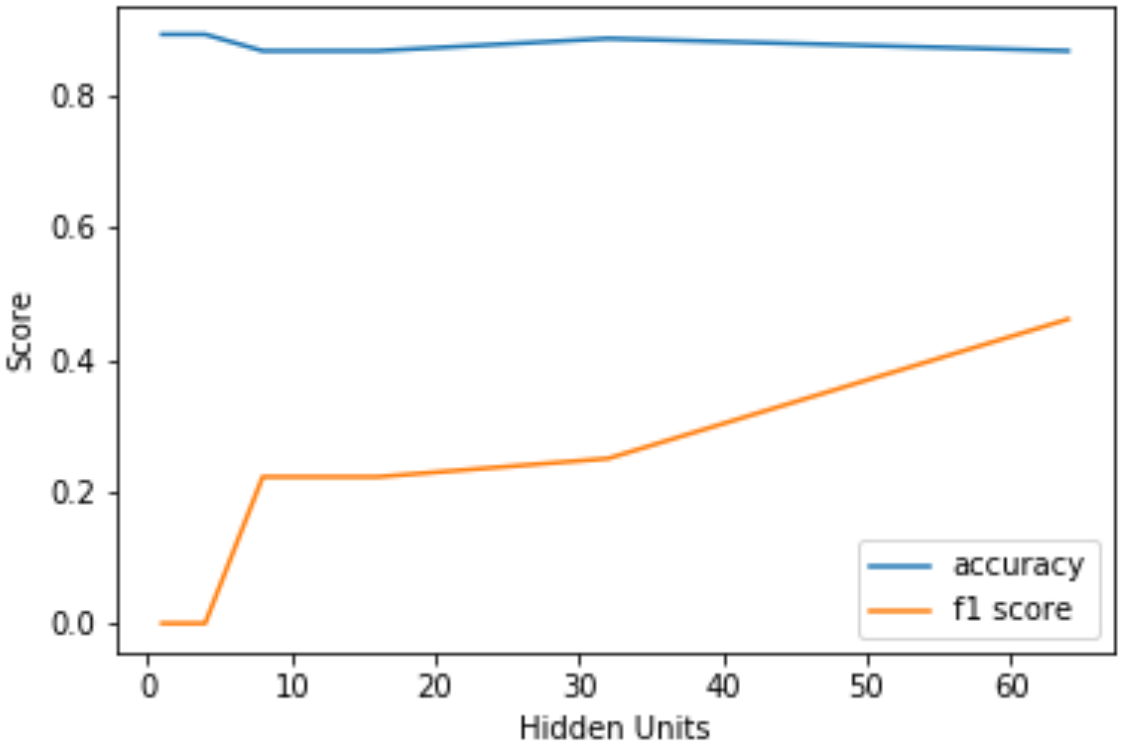
```
Linear Regression
SGDClassifier(alpha=0.1, average=False, class_weight=None, early_stopping=False,
          epsilon=0.1, eta0=0.1, fit_intercept=True, l1_ratio=0.15,
          learning_rate='adaptive', loss='log', max_iter=500,
          n_iter_no_change=5, n_jobs=None, penalty='l2', power_t=0.5,
          random_state=0, shuffle=True, tol=0.001, validation_fraction=0.1,
          verbose=1, warm_start=False)

Best Training Time;  0.04816102981567383
Best accuracy:  0.9187499999999998
Best f1 score:  0.48
Neural Network

MLPClassifier(activation='relu', alpha=0.0001, batch_size='auto', beta_1=0.9,
          beta_2=0.999, early_stopping=False, epsilon=1e-08,
          hidden_layer_sizes=(64,), learning_rate='constant',
          learning_rate_init=0.001, max_fun=15000, max_iter=500,
          momentum=0.9, n_iter_no_change=10, nesterovs_momentum=True,
          power_t=0.5, random_state=4211, shuffle=True, solver='sgd',
          tol=0.0001, validation_fraction=0.1, verbose=False,
          warm_start=False)

Best Training Time;  0.594632069269816
Best accuracy:  0.86875
Best f1 score:  0.4615384615384615
```

## [Q8] Plot the accuracy and the F1 score for different values of H.




## [Q9] Do you notice any trend when you increase the hidden layer size from 1 to 64? If so, please describe what the trend is.

With increasing the hidden layer size from 1 to 64, the f1 score increase while the accuracy score remain more or less the same.

## [Q10] Referring to your experiment results, comment on the gap between accuracy and the F1 score? Suggest a reason for this observation.

```
pokemons['Legendary'].value_counts()
False    735
True      65
Name: Legendary, dtype: int64
```

The gap between accuracy score and f1 score is quite large (~1% - 50%), I think the main reason is the skewed dataset. Most of the data are not Legendary(8.125%), so only a few data sample are Legendary(91.875%). The value of true-positive will be small, this is the factor that will lower the f1 score.

# Task 4: Predicting the Winners in the Pokemon Battles

## [Q11] Report 10 combinations of the hyperparameter setting.

```
 {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (512,),
'learning_rate': 'adaptive', 'learning_rate_init': 0.01, 'solver': 'sgd'}
 {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (512,),
'learning_rate': 'adaptive', 'learning_rate_init': 0.01, 'solver': 'sgd'}
 {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (128,),
'learning_rate': 'adaptive', 'learning_rate_init': 0.01, 'solver': 'sgd'}
 {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (256,),
'learning_rate': 'adaptive', 'learning_rate_init': 0.01, 'solver': 'sgd'}
 {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (256,),
'learning_rate': 'adaptive', 'learning_rate_init': 0.01, 'solver': 'sgd'}
 {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (128,),
'learning_rate': 'adaptive', 'learning_rate_init': 0.01, 'solver': 'sgd'}
 {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50,),
'learning_rate': 'adaptive', 'learning_rate_init': 0.01, 'solver': 'sgd'}
 {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (128,),
'learning_rate': 'adaptive', 'learning_rate_init': 0.001, 'solver': 'sgd'}
 {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50,),
'learning_rate': 'adaptive', 'learning_rate_init': 0.01, 'solver': 'sgd'}
 {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (128,),
'learning_rate': 'adaptive', 'learning_rate_init': 0.001, 'solver': 'sgd'}
```

## [Q12] Report the three best hyperparameter settings as well as the mean and standard deviation of the validation accuracy of the five random data splits for each hyperparameter setting.

| | mean | std | params | | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | activation | alpha | hidden layer sizes | learning rate | learn rate init | solver |
| **250** | 0.948050 | 0.006249 | tanh | 0.05 | 512 | adaptive | 0.01 | sgd |
| **124** | 0.947675 | 0.005045 | tanh | 0.0001 | 512 | adaptive | 0.01 | sgd |
| **214** | 0.947600 | 0.006041 | tanh | 0.05 | 128 | adaptive | 0.01 | sgd |

## [Q13] Use the best model to predict the instances in the test set (q4 test.csv). Report the accuracy.

```
              precision    recall  f1-score   support

           0       0.97      0.94      0.95      5282
           1       0.94      0.97      0.95      4718

    accuracy                           0.95     10000
   macro avg       0.95      0.95      0.95     10000
weighted avg       0.95      0.95      0.95     10000
```

## [Q14] Print the confusion matrix of the predictions on the test set.

```
Confusion matrix:
[[4972  310]
 [ 176 4542]]
```