# CSCE633 Homework 02

Lu Sun

September 2020

## 1 Data exploration:

(a) Categorical attributes

(b) Continuous attributes

**Answer:**
From the data set in file "*hw2_data.csv*", finally results can be get:

- Categorical attributes: *primary_strength* (Column 8)
- Continuous attributes: *stamina*, *attack_value*, $defense\_value$, *capture_rate*, $flee\_rate$, *spawn_chance* (Column 2-7)

## 2 Data exploration:

(a) Plot 2-D scatter plots.

(b) Compute the Pearson's correlation coefficient.

(c) Which attributes would be the most predictive.

**Answer:**
The code for this part is as follow:

```
import matplotlib.pyplot as plt
import scipy.stats

y = combat_point
for i in range(6):
  x = NameMap[i][1]
  xlab = NameMap[i][0]
  plt.scatter(x, y)
  plt.xlabel('{0}'.format(xlab))
  plt.ylabel('combat_point')
  plt.show()

  PeaRate = scipy.stats.pearsonr(x,y)
  print('Pearsonr Correlation Rate for {0}: ...
      {1}'.format(xlab,PeaRate))
```

## 2.1 Plots

The results of plots are shown as follow:

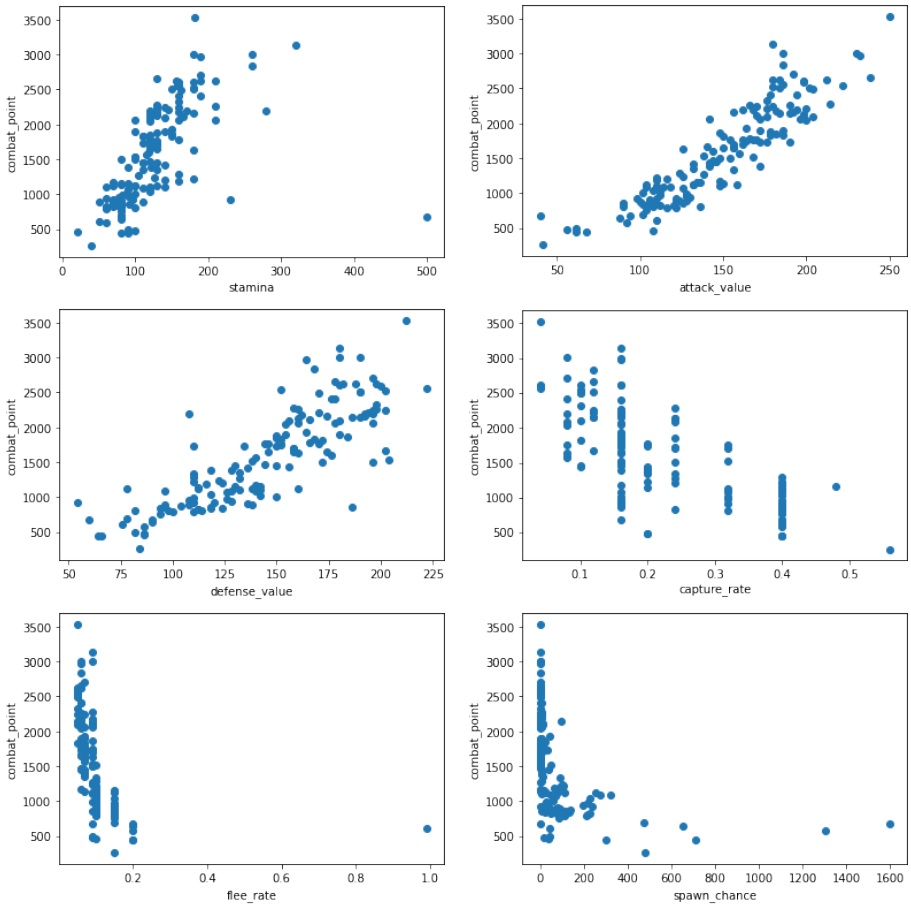



Figure 1: Scatter Plots with combat point

## 2.2 Pearson's correlation coefficient

The Pearson's correlation coefficients are shown in the following table:

| $stamina$ | $attack\_value$ | $defense\_value$ |
|---|---|---|
| 0.582831703222926 | 0.9075315401042733 | 0.8262293053572931 |
| $capture\_rate$ | $flee\_rate$ | $spawn\_chance$ |
| -0.7430078083529397 | -0.40703421142159646 | -0.42132699465983586 |

Table 1: Pearson's correlation coefficient with combat point

### 2.3 The most predictive attribute

From the table 1, *attack_value* is the most predictive attribute. The reason is as follow:

*attack_value* has the largest absolute value of Pearson's correlation coefficient, which is closest to 1 among all the other attributes. By the definition of Pearson's correlation coefficient, the closer distance it has to 1 or -1, the more predictable relation data set will get.

## 3 Data exploration:

(a) Plot scatter plots among continuous attributes only.

(b) Compute corresponding Pearson's correlation coefficient.

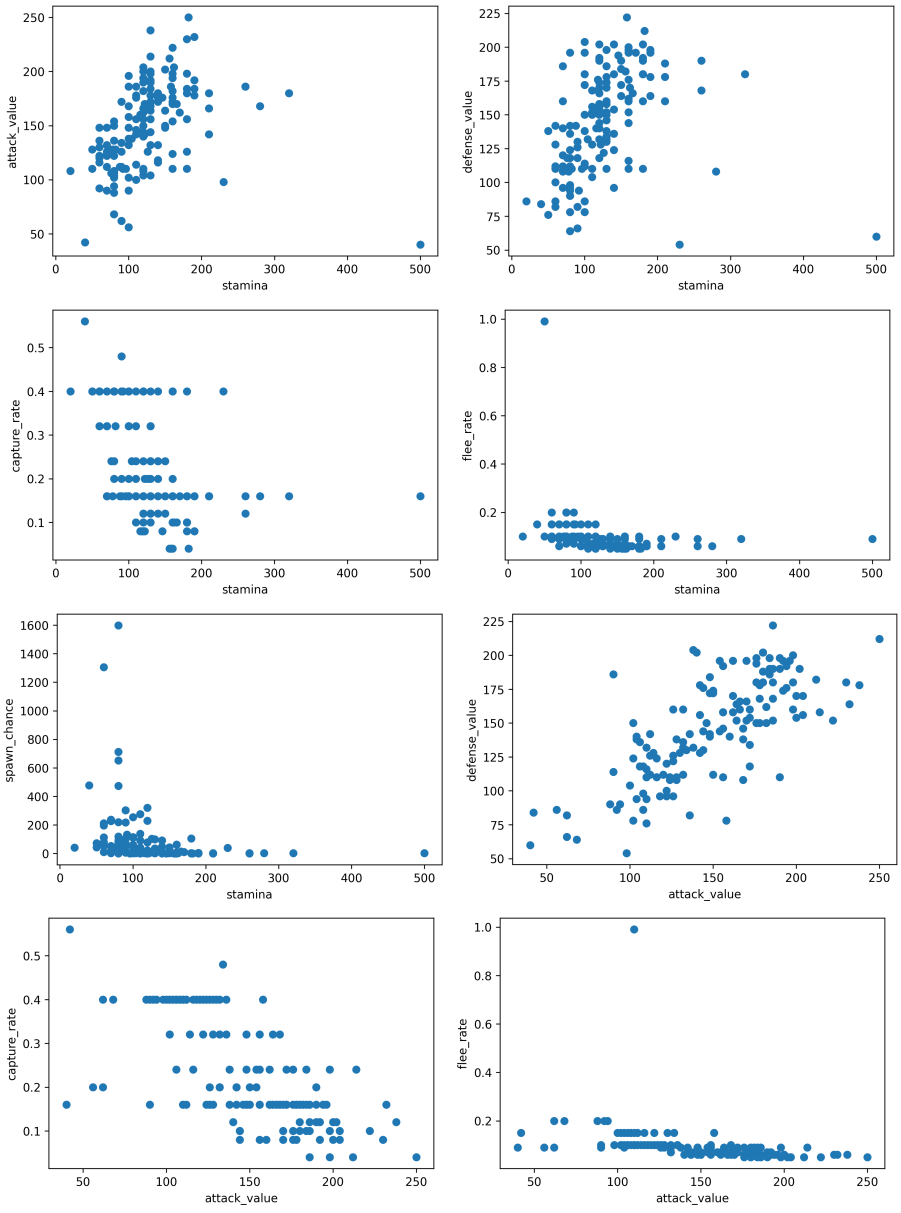
(c) Which variables are the most correlated to each other.

**Answer:**
The code for this part is as follow:

```
import matplotlib.pyplot as plt
%matplotlib inline

for i in range(6):
  j = i+1
  while j<6:
    filename = ','.join(['/content/drive/My ...
        Drive/2020_FALL/CSCE633/HomeWork/HW02/Figure/...
    prob3_'+str(i)+str(j)+'.png' ])

    x = NameMap[i][1]
    xlab = NameMap[i][0]
    y = NameMap[j][1]
    ylab = NameMap[j][0]

    plt.scatter(x, y)
    plt.xlabel('{0}'.format(xlab))
    plt.ylabel('{0}'.format(ylab))
    plt.savefig('{0}'.format(filename), dpi=300, ...
        bbox_inches='tight')  # Returns the list of datapoints

    plt.show()

    PeaRate = scipy.stats.pearsonr(x,y)
    print('Pearsonr Correlation Rate for ...
        ({0},{1}):{2}'.format(xlab,ylab,PeaRate))

    j += 1
```

## 3.1 Plots

The scatter plots are shown as follow:



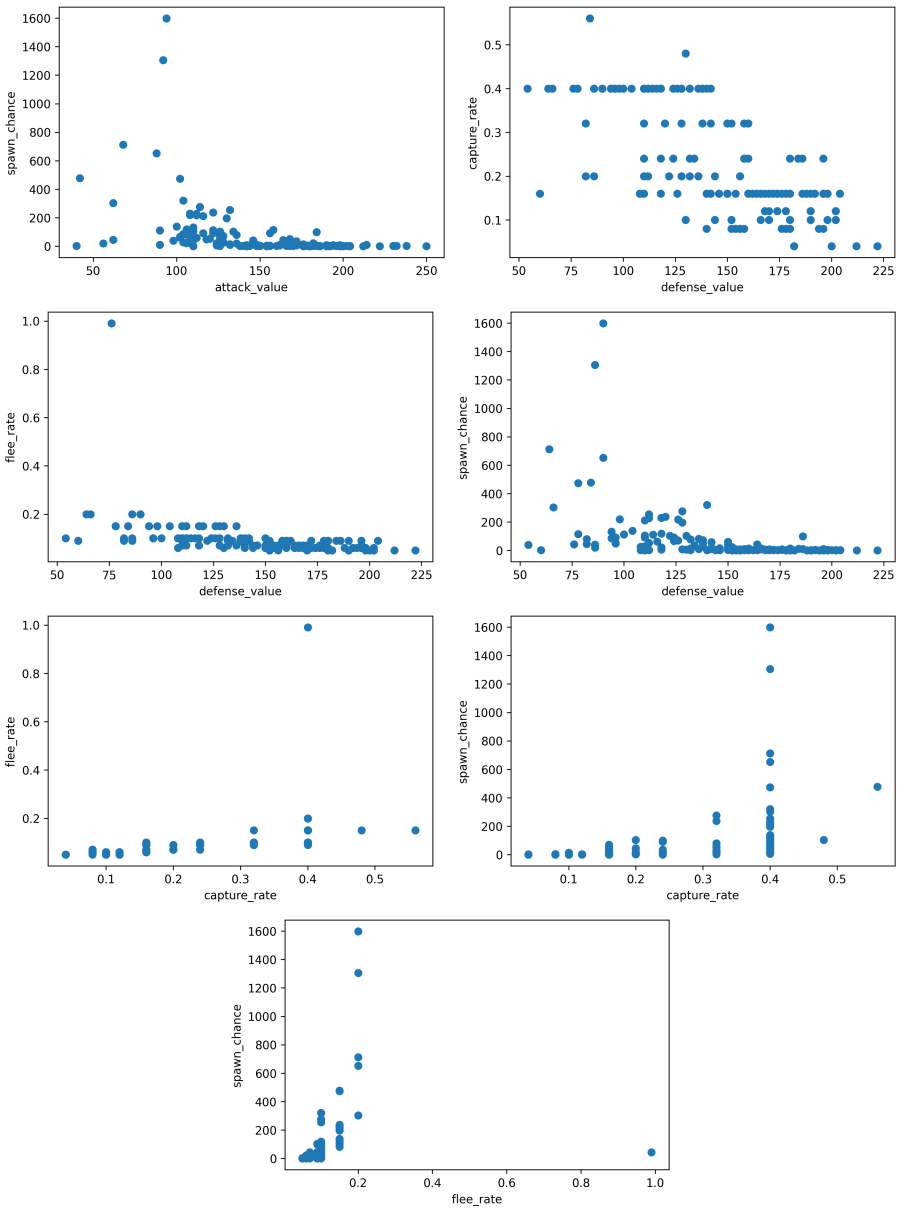



Figure 2: Scatter Plots among continuous attributes

### 3.2 Pearson's correlation coefficient

The Pearson's correlation coefficients among continuous attributes are shown in the following table.

| | | |
|---|---|---|
| $(stamina, attack\_value)$ | $(stamina, defense\_value)$ | $(stamina, capture\_rate)$ |
| 0.3029949826738916 | 0.30266333625368935 | -0.4468503047144601 |
| $(stamina, flee\_rate)$ | $(stamina, spawn\_chance)$ | $(attack\_value, defense\_value)$ |
| -0.2710475393248393 | -0.2764202078836037 | 0.7367766467515237 |
| $(attack\_value, capture\_rate)$ | $(attack\_value, flee\_rate)$ | $(attack\_value, spawn\_chance)$ |
| -0.6905726716022137 | -0.36906414197600723 | -0.4326484402010869 |
| $(defense\_value, capture\_rate)$ | $(defense\_value, flee\_rate)$ | $(defense\_value, spawn\_chance)$ |
| -0.6972657162131648 | -0.42385975623729333 | -0.43249856208332005 |
| $(capture\_rate, flee\_rate)$ | $(capture\_rate, spawn\_chance)$ | $(flee\_rate, spawn\_chance)$ |
| 0.44051150728059624 | 0.4727927266445679 | 0.29322169222082034 |

Table 2: Pearson's correlation coefficient among continuous attributes

### 3.3 The most related pair

From the above table 2, the most related pair is $(attack\_value, defense\_value)$ whose Pearson's correlation coefficient is 0.7367766467515237. The reason for this is total the same as that in problem 2.3.

## 4 Pre-processing of categorical variables:

(a) Count the number of different values.

(b) Implement the one hot encoding.

**Answer:**

- First, generate all the classes without duplicates by python's package *np.unique* and compute the total number of class as *onehot_length*
- Then, map each class to corresponding one hot vector as *onehot_mapTotal*
- Finally, use *combat_point_new* to record all the one hot vectors

The code for this problem is as follow:

```
combat_point = NameMap[6][1]
print(combat_point)

onehot_class = np.unique(combat_point)
onehot_length = len(onehot_class)
onehot_mapTotal = {}
```

```
for i in range(onehot_length):
  onehot = np.zeros(onehot_length,dtype=int)
  onehot[i] = 1
  onehot_map = {onehot_class[i]:onehot}
  onehot_mapTotal.update(onehot_map)
#print(combat_point)
#print(onehot_mapTotal)

combat_point_new = []
for i in range(len(combat_point)):
  combat_point_new.append(onehot_mapTotal[combat_point[i]])
#print(combat_point)
#print(combat_point_new)
```

# 5 Prediction combat points:

(a) How many parameters does OLS model have.

(b) Implement a linear regression model with OLS and implement RSS error computation code.

(c) Randomly split data and implement 5-fold cross-validation process.

(d) Compute square root of RSS for each fold and the average of all folds.

**Answer:**

## 5.1 Data process

From previous problems, $N_{features} = 22$ feature parameters are needed. The reason is as follow:

Here, feature parameter is represented by $\omega \in R^{N\times 1}$ and $\omega_0$ is the parameter corresponding to constant feature 1. Then, in the previous problem, the number of continuous features are 6, namely $stamina$,$attack\_value$,$defense\_value$,$capture\_rate$, $flee\_rate$,$spawn\_chance$. Then number for categorical feature $primary\_strength$'s class are 15, namely $Bu$,$Dragon$,$Electric$,$Fair$,$Fighting$,$Fire$,$Ghost$,$Grass$,$Ground$, $Ice$,$Normal$,$Poison$,$Psychic$,$Rock$,$Water$. Finally, $22 = 1 + 6 + 15$ can be got.

## 5.2 Linear regression and RSS algorithms

In this problem 5 only ordinary linear regression model(OLS) is needed to generate, while in problem 6, regularity linear regression model(RLS) is needed to generate. In this part, the two methods will be generated at the same time.

First, initialize $\omega(0) = \vec{0}$ and fix hyper parameters' value, learning rate and max iteration number.

Then, implement function *score* to compute RSS within whose process function *predict* is needed for computing $X\omega$. In function *score*, finally, $(X\omega - y)^2$ can be obtained.

In the process of implementing $OLS$ and $RLS$ algorithm, the only difference is the way to generate gradients. In my code, basic gradient descent method is used in both algorithm. Then the formulas of gradients for $OLS$ and $RLS$ are shown as follow: (Here, assume $X_0[samples] = 1$) The above two formula

$$\begin{aligned} \text{OLS:} \quad & \nabla_\omega J_{OLS}(\omega) = 2 \cdot X^T(X\omega - y) \\ \text{RLS:} \quad & \nabla_\omega J_{RLS}(\omega) = 2 \cdot X^T(X\omega - y) + 2\lambda \cdot \omega \end{aligned}$$

can be obtained in function $OLR\_gradients, RLR\_gradients$ respectively. And function $RLR\_gradients$ needs one more hyper parameter, regularity parameter $\lambda$, than function $OLR\_gradients$

Finally, function $fit\_GD\_OLR$ and $fit\_GD\_RLR$ are the corresponding gradient descent method by using the above gradients respectively.

The code is shown as follow:

```
1  class LinearRegression(object):
2
3    def __init__(self,W,learning_rate, max_iter):
4      self.learning_rate = learning_rate
5      self.max_iter = max_iter
6      self.W = W
7
8    def get_params(self):
9      return self.W
10
11   def predict(self, X):
12     return X.dot(self.W)
13
14   def score(self, X, y):
15     score_tep = self.predict(X) - y
16     score = np.transpose(score_tep).dot(score_tep)
17     return np.sqrt(score)
18
19   ############################Ordinary############################
20   def fit_GD_OLR(self, X, y):
21     sample_size, feature_size = X.shape
22     #self.W = np.zeros(feature_size)  # Initialize model ...
           parameters to zeros
23
24     losses = []  # Store loss values at each epoch
25     for t in range(self.max_iter):
26       losses_tep = self.score(X,y)
27       losses.append(losses_tep)
28
29       gt = 2*self.OLR_gradients(X,y)#/sample_size
30       self.W = self.W - self.learning_rate * gt
31     return self.W, losses
32
33   def OLR_gradients(self, _x, _y):
34     _g =  _x.T.dot(self.predict(_x) - _y)
35 #    _g = ( self.W.dot(_x) - _y) * _x
36     return _g
37   ############################Regularity############################
```

```
38    def fit_GD_RLR(self, X, y, labd):
39      sample_size, feature_size = X.shape
40
41      losses = []  # Store loss values at each epoch
42      for t in range(self.max_iter):
43        losses_tep = self.score(X,y)
44        losses.append(losses_tep)
45        #if losses_tep ≤2900:
46        #  print("stop ahead at iteration: {0}".format(t))
47        #  break
48        gt = 2*self.RLR_gradients(X,y,labd)
49        self.W = self.W - self.learning_rate * gt
50      return self.W, losses
51
52    def RLR_gradients(self, _x, _y, labd):
53      _g =  _x.T.dot(self.predict(_x) - _y) + labd * self.W
54      return _g
```

## 5.3 5-fold cross-validation

In this part, the 5-fold cross-validation algorithm is generated.

First, a set of random index is generated and stores in *random_index* whose length is the same as sample size.

Then, divide the training set into two part, training set and validation set according to the random index set in the first step. Namely, for example, select *random_index*[0]th,*random_index*[5]th,*random_index*[10]th,... number in original input $(X, y)$ as the first fold validation data set and the left as training data set. Repeat this part for 5 times to complete 5-fold cross-validation. Within each fold, use corresponding method *OLS* in this problem.

The code is as follow:

```
1
2   def cross_validation(X,y,W_ini,method,...
3           labd=None,K_fold=5,learning_rate = 1e-7,max_iter = 15000):
4   #MethodMap = {'OrdinaryLinearR': 0,
5   #              'RegularLinearR': 1,
6   #             'l2normLogisticR': 2,
7   #             'l1normLogisticR': 3}
8     #random divide data set into 5 folds
9     random_index = np.random.RandomState(seed = ...
10                                          0).permutation(X.shape[0])
11
12    Lg = len(random_index)
13    w_tol = []
14    rss_tol = []
15    losses = None
16
17    for i in range(K_fold):
18      # get: i,i+5,i+10,i+15,... for validation index
19      index_val = random_index[i::K_fold]
20      # get the remaining for training index
21      index_train = np.delete(random_index,np.arange(Lg)[i::K_fold])
```

```
22
23      X_val = X[index_val]
24      X_train = X[index_train]
25
26      y_val = y[index_val]
27      y_train = y[index_train]
28
29      if method == 0:
30        LRA = LinearRegression(W = W_ini,...
31                  learning_rate = learning_rate, max_iter = max_iter)
32        w_tep, losses = LRA.fit_GD_OLR(X_train,y_train)
33        rss_tep = LRA.score(X_val,y_val)
34
35      elif method == 1:
36        LRA = LinearRegression(W = W_ini,...
37                  learning_rate = learning_rate, max_iter = max_iter)
38        w_tep, losses = LRA.fit_GD_RLR(X_train,y_train,labd)
39        rss_tep = LRA.score(X_val,y_val)
40
41      elif method == 2:
42        clf = LogisticRegression(penalty='l2', C=labd,...
43                                      solver='saga',max_iter=max_iter)
44        clf.fit(X_train, y_train)
45        w_tep = clf.coef_
46        rss_tep = clf.score(X_val, y_val)
47
48      elif method ==3:
49        clf = LogisticRegression(penalty='l1', C=labd,...
50                                      solver='saga',max_iter=max_iter)
51        clf.fit(X_train, y_train)
52        w_tep = clf.coef_
53        rss_tep = clf.score(X_val, y_val)
54
55      if losses != None:
56        # Plot losses to make sure it converges with the fixed ...
              hyper parameter
57        plt.plot(losses,'+-')
58        plt.xlabel("number of epochs")
59        plt.ylabel("loss")
60        plt.show()
61
62      w_tol.append(w_tep)
63      rss_tol.append(rss_tep)
64      print('fold {0} rss:{1}'.format(i+1,rss_tep))
65    rss_ave = np.sum(rss_tol)/K_fold
66    print('average rss:',rss_ave)
67    return w_tol,rss_tol,rss_ave
```

Here, method '0' means ordinary linear regression which is used in this problem 5. The method '1' is corresponding to regularity linear regression which is needed in next problem 6. The method '2' and '3' represent logistic regression with l-2 norm and l-1 norm respectively, which will be used in the following problem 9. And finally, $10^{-7}$ is chosen as the learning rate which can get convergence results.

### 5.4 Compute RSS

The code for getting results is shown as follow:

```
#################Genearate basic form dataset##################
y = np.array(NameMap[7][1])
X_continuous = np.array([NameMap[i][1] for i in range(6)]).T
X_categorical = np.array(primary_strength_new)

X = np.hstack((X_continuous,X_categorical))
X_basis = np.hstack((np.ones([X.shape[0],1]),X))

W_ini = np.linalg.inv(np.transpose(X_basis).dot(X_basis))....
                                dot(np.transpose(X_basis).dot(y))
###################Cross Validation#############################
_,rss_tol,rss_ave = cross_validation(X_basis,y,0*W_ini,method = 0)
print('all rss:{},average rss:{}'.format(rss_tol,rss_ave))
```

The results are shown in the following table.

| | |
|---|---|
| fold 1 | 1663.962670802079 |
| fold 2 | 2193.803990273198 |
| fold 3 | 1262.3876327445237 |
| fold 4 | 2249.7536666025358 |
| fold 5 | 1374.2513634282927 |
| average | 1748.8318647701258 |

Table 3: square root of RSS for OLS

# 6 Prediction combat points:

(a) Implement ridge regression( linear regression with regularization)

(b) Experiment and report the results with different $\lambda$

**Answer:** The corresponding method has been mentioned in problem 5 and implemented in function $fit\_GD\_RLR$. Initialize $\lambda = 0.001, 0.01, 1, 10, 100$. Different $\lambda$s share the same way of random dividing data set into 5 folds. The code to get results is shown as follow:

```
labd_total = [0.01,0.1,1,10,100]
rss_tol = []
rss_ave = []
for labd in labd_total:
  _,rss_tol_tep,rss_ave_tep = ...
      cross_validation(X_basis,y,0*W_ini,method = 1,labd = labd)
  print('lambda:{}'.format(labd))
  print('all rss:{},average rss:{}'.format(rss_tol_tep,rss_ave_tep))

```

```
    rss_tol.append(rss_tol_tep)
    rss_ave.append(rss_ave_tep)

for i in range(len(labd_total)):
    print('lambad:{}, average rss:{}, all ...
        rss:{}'.format(labd_total[i],rss_ave[i],rss_tol[i]))
```

The results are shown in the following table:

| fold | $\lambda = 0.01$ | $\lambda = 0.1$ | $\lambda = 1$ | $\lambda = 10$ | $\lambda = 100$ |
|---|---|---|---|---|---|
| 1 | 1663.96298840 | 1663.96584647 | 1663.99439964 | 1664.27719642 | 1666.85083466 |
| 2 | 2193.80448491 | 2193.80893614 | 2193.85340296 | 2194.29355673 | 2198.27545329 |
| 3 | 1262.38796634 | 1262.39096840 | 1262.42095945 | 1262.71793643 | 1265.41489576 |
| 4 | 2249.75391616 | 2249.75616197 | 2249.77859782 | 2250.00074734 | 2252.01674105 |
| 5 | 1374.25164430 | 1374.25417190 | 1374.27942298 | 1374.52946509 | 1376.80025092 |
| Ave | 1748.83220002 | 1748.83521697 | 1748.86535657 | 1749.16378040 | 1751.87163514 |

Table 4: square root of RSS for RLS

# 7 Bonus:

(a) Use linear regression with different feature combinations.

**Answer:**
The code for this part is as follows:

```

#NameMap = {0:['stamina',stamina],
#           1:['attack_value',attack_value],
#           2:['defense_value',defense_value],
#           3:['capture_rate',capture_rate],
#           4:['flee_rate',flee_rate],
#           5:['spawn_chance',spawn_chance],
#           6:['primary_strength',primary_strength],
#           7:['combat_point',combat_point]}
def CombinFeature(X_new_basis,learning_rate = 1e-7,max_iter = ...
    15000):
  y = np.array(NameMap[7][1])
  W_ini = ...
      np.linalg.inv(np.transpose(X_new_basis).dot(X_new_basis))....
                              dot(np.transpose(X_new_basis).dot(y))
  ######################Cross Validation##########################
  _,rss_tol,rss_ave = cross_validation(X_new_basis, y, 0*W_ini,...
        method = 0,learning_rate=learning_rate,max_iter=max_iter)
  print('all rss:{},average rss:{}'.format(rss_tol,rss_ave))
  return rss_tol, rss_ave

rss_tol = []
rss_ave_tol = []
combain_features = []
for i in range(6):
```

```
    j = i + 1
    while j < 6:
      X_new = np.array([NameMap[i][1],NameMap[j][1]]).T
      X_new_basis = np.hstack((np.ones([X_new.shape[0],1]),X_new))
      name_tep = (NameMap[i][0],NameMap[j][0])
      print(name_tep)
      combain_features.append(name_tep)

      rss,rss_ave = CombinFeature(X_new_basis,learning_rate = 1e-9)
      rss_tol.append(rss)
      rss_ave_tol.append(rss_ave)
      j = j +1

for i in range(len(combain_features)):
    print('Combain features:{}, average rss:{}, all ...
        rss:{}'.format(combain_features[i],rss_ave_tol[i],rss_tol[i]))
```

The results is shown as follow:

| Combination feature | Average RSS |
|---|---|
| $(stamina, attack\_value)$ | 1761.7612002966048 |
| $(stamina, defense\_value)$ | 2105.0676869515 |
| $(stamina, capture\_rate)$ | 3312.81517445934 |
| $(stamina, flee\_rate)$ | 3312.8048894631384 |
| $(stamina, spawn\_chance)$ | 3305.896347648588 |
| $(attack\_value, defense\_value)$ | 1829.5275727386877 |
| $(attack\_value, capture\_rate)$ | 1865.0731029770945 |
| $(attack\_value, flee\_rate)$ | 1865.1272504360481 |
| $(attack\_value, spawn\_chance)$ | 1888.4128783248634 |
| $(defense\_value, capture\_rate)$ | 2237.345613547558 |
| $(defense\_value, flee\_rate)$ | 2237.386858013856 |
| $(defense\_value, spawn\_chance)$ | 2336.7613282436782 |
| $(capture\_rate, flee\_rate)$ | 2255.356812760387 |
| $(capture\_rate, spawn\_chance)$ | 9394.510236534275 |
| $(flee\_rate, spawn\_chance)$ | 9395.262608277268 |

Table 5: RSS with different combination features

In problem 2 and problem 3, the following observation can be get:

- The most related to output's features are $attack\_value$, $defense\_value$ ,$capture\_rate$,$stamina$ whose Pearson's correlation coefficient's absolute value larger than 0.5
- The following pairs likely have linear relation $(attack\_value, defense\_value)$, $(attack\_value, capture\_rate)$,$(defense\_value, capture\_rate)$ whose corresponding value also larger than 0.5

From the above table 5, similar results can be explained by the results from problem 2 and problem 3. For example, combination $(stamina, attack\_value)$ has the smallest average RSS. This is corresponding to the following two things:

- *attack_value* and *stamina* value show strong relation (¿0.5) with output value caused by problem 2's results.
- *attack_value* and *stamina* value show no relation (¡0.5) with each other caused by problem 3's results.

# 8 Logistic Regression:

(a) Use the mean sample value to binarize the data.

(b) Split the data into training and test.

(c) Report the accuracy of the classifier on the test data.

**Answer:**

## 8.1 Binary the data

In this part, first $y\_mean$ is used for storing mean value of output. Then label $y\_lab = +1$ if value $y \geq y\_mean$ and label $y\_lab = -1$ if value $y < y\_mean$. The correponding code is as follow:

```
y = np.array(NameMap[7][1])
y_lab = np.zeros(y.shape,dtype = int)
y_mean = y.mean()

y_lab[np.where(y ≥ y_mean)] = 1
y_lab[np.where(y <  y_mean)] = -1

print(y_mean)
print(y)
print(y_lab)
```

## 8.2 Split data

In this part, package $sklearn.model_selection$ is needed to import function $train\_test\_split$. Here, set the number of test set equals 0.15 multiply the number of total data set. The code is as follow:

```
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X_basis, ...
     y_lab, test_size=0.15, random_state=0)
```

### 8.3 Accuracy

In this part, build-in function *LogisticRegression* is used. The code is as follow:

```
from sklearn.linear_model import LogisticRegression

clf = LogisticRegression(penalty = ...
    'none',solver='saga',max_iter=15000)
clf.fit(X_train, y_train)
print(clf.predict(X_test))
print(y_test)
print('train acc:{},test ...
    acc:{}'.format(clf.score(X_train,y_train),clf.score(X_test, ...
    y_test)))
```

The result is as follow:

| | |
|---|---|
| train acc: | 0.8306451612903226 |
| test acc: | 0.9545454545454546 |

## 9 Regularization logistic regression:

(a) Find optimal $\lambda^*$ by 5-fold cross-validation on training data set.

(b) Use $\lambda^*$ in (a) and calculate accuracy on testing data set.

**Answer:**

Since there exist two kinds of norm, l-2 norm and l-1 norm, solving method $'saga'$ is used to handle this problem.

### 9.1 Find $\lambda^*$

The codes for l-2 norm and l-1 norm are really similar. So in this part, only the results of l-1 norm will be shown. Take l-2 norm as the example. The code is as follow:

```
################l2norm###############
labd_total = np.linspace(0.0001,0.1,12)
#labd_total = np.linspace(1000,100000,100)
rss_tol = []
rss_ave = []
for labd in labd_total:
  _,rss_tol_tep,rss_ave_tep = cross_validation(X_train,y_train, ...
      W_ini= None,method = 2,labd = labd,max_iter=10000)
  print('lambda:{}'.format(labd))
  print('all rss:{},average rss:{}'.format(rss_tol_tep,rss_ave_tep))

  rss_tol.append(rss_tol_tep)
  rss_ave.append(rss_ave_tep)
```

```
13
14  for i in range(len(labd_total)):
15    print('l2 penalty, lambad:{}, average rss:{}, all ...
          rss:{}'.format(labd_total[i],rss_ave[i],rss_tol[i]))
16
17  plt.plot(labd_total,rss_ave,'+-')
18  plt.xlabel('lambda')
19  plt.ylabel('Average Accuracy')
20  plt.title('l-2 norm')
```

Finally, the following results can be shown:

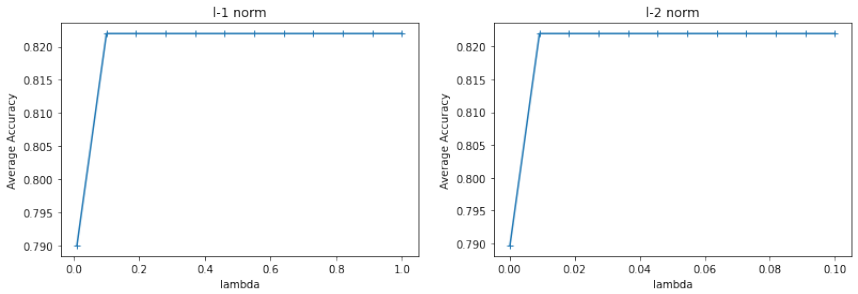



Figure 3: Find $\lambda^*$

From the above figure, choose $\lambda^* = 0.1$ for l-1 norm and $\lambda^* = 0.01$ for l-2 norm.

## 9.2 Calculate accuracy

The code for this part is as follow:

```
1  labd = 0.001
2  clf = LogisticRegression(penalty = ...
       'l2',C=labd,solver='saga',max_iter=15000)
3  clf.fit(X_train, y_train)
4  print(clf.predict(X_test))
5  print(y_test)
6  print('train acc:{},test ...
       acc:{}'.format(clf.score(X_train,y_train),clf.score(X_test, ...
       y_test)))
```

The final results are shown in the following table:

| norm | $\lambda^*$ | Train Acc | Test Acc |
|---|---|---|---|
| l-1 norm | 0.1 | 0.8306451612903226 | 0.9545454545454546 |
| l-2 norm | 0.01 | 0.8387096774193549 | 0.9545454545454546 |

Table 6: Results

# Appendix

### (i) Code for upload data

```
import numpy as np

class DataPoint(object):

  def __init__(self, feats):
    """
    Return a Datapoint object whose attributes are given by "feats"
    """
    self.stamina          = feats['stamina']
    self.attack_value     = feats['attack_value']
    self.defense_value    = feats['defense_value']
    self.capture_rate     = feats['capture_rate']
    self.flee_rate        = feats['flee_rate']
    self.spawn_chance     = feats['spawn_chance']
    self.primary_strength = feats['primary_strength']
    self.combat_point     = feats['combat_point']

  def feature_vector(self):
    """
    Return feature vector as a numpy array
    """
    return np.array([self.stamina,         self.attack_value,
                     self.defense_value,   self.capture_rate,
                     self.flee_rate,       self.spawn_chance,
                     self.primary_strength])


  def full_vector(self):
    """
    Return feature vector as a numpy array
    """
    return np.array([self.stamina,         self.attack_value,
                     self.defense_value,   self.capture_rate,
                     self.flee_rate,       self.spawn_chance,
                     self.primary_strength,self.combat_point])


def parse_dataset(filename):
  data_file = open(filename, 'r')  # Open File "to read"
  dataset = []  # List to hold Datapoint objects

  for index, line in enumerate(data_file):
    if index == 0:  # First line: "Height   Weight  Label" ...
        describes the datapoint, it's not an actual datapoint,
      continue  # do nothing, it will skip all the following lines
    #print(line)
    _,stamina, attack_value, defense_value, ...
        capture_rate,flee_rate, spawn_chance, primary_strength, ...
        combat_point = ...
        line.strip().split(',')#line.strip().split('\t')  # ...
        strip() removes '\n', and split('\t') splits the line at ...
        tabs
```

```
    dataset.append(DataPoint({'stamina':float(stamina),...
                      'attack_value':float(attack_value),...
                    'defense_value':float(defense_value),...
                      'capture_rate':float(capture_rate),...
                           'flee_rate':float(flee_rate),...
                      'spawn_chance':float(spawn_chance),...
                     'primary_strength':primary_strength,...
                   'combat_point':float(combat_point)}))
            # Create DataPoint object for the given data

  print("Total Number of Data Points: {0}".format(len(dataset)))
  print("Examples: 1 - {0} , 2 - {1}".format(dataset[0],...
                                             dataset[-1]))
  return dataset

datasetTrain = parse_dataset('/content/drive/My ...
    Drive/2020_FALL/CSCE633/HomeWork/HW02/hw2_data.csv')
# Returns the list of datapoints
Lg = len(datasetTrain)
stamina         =[datasetTrain[i].stamina for i in range(Lg)]
attack_value    =[datasetTrain[i].attack_value for i in range(Lg)]
defense_value   =[datasetTrain[i].defense_value for i in range(Lg)]
capture_rate    =[datasetTrain[i].capture_rate for i in range(Lg)]
flee_rate       =[datasetTrain[i].flee_rate for i in range(Lg)]
spawn_chance    =[datasetTrain[i].spawn_chance for i in range(Lg)]
primary_strength=[datasetTrain[i].primary_strength for i in ...
    range(Lg)]
combat_point    =[datasetTrain[i].combat_point for i in range(Lg)]
NameMap = {0:['stamina',stamina],
           1:['attack_value',attack_value],
           2:['defense_value',defense_value],
           3:['capture_rate',capture_rate],
           4:['flee_rate',flee_rate],
           5:['spawn_chance',spawn_chance],
           6:['primary_strength',primary_strength],
           7:['combat_point',combat_point]}
print(NameMap[0][1])
```

### (ii) Code for l-1 norm

```
################l1norm###############
labd_total = np.linspace(0.01,1,12)
#labd_total = np.linspace(1000,100000,100)
rss_tol = []
rss_ave = []
for labd in labd_total:
  _,rss_tol_tep,rss_ave_tep = cross_validation(X_train,y_train, ...
      W_ini= None,method = 3,labd = labd,max_iter=10000)
  print('lambda:{}'.format(labd))
  print('all rss:{},average rss:{}'.format(rss_tol_tep,rss_ave_tep))

  rss_tol.append(rss_tol_tep)
  rss_ave.append(rss_ave_tep)

for i in range(len(labd_total)):
  print('l2 penalty, lambad:{}, average rss:{}, all ...
```

```
        rss:{}'.format(labd_total[i],rss_ave[i],rss_tol[i]))

plt.plot(labd_total,rss_ave,'+-')
plt.xlabel('lambda')
plt.ylabel('Average Accuracy')
plt.title('l-1 norm')
#plt.ylim(0.82,0.825)
#plt.xlim(0.09,0.101)


labd = 0.1
clf = LogisticRegression(penalty = ...
    'l1',C=labd,solver='saga',max_iter=15000)
clf.fit(X_train, y_train)
print(clf.predict(X_test))
print(y_test)
print('train acc:{},test ...
    acc:{}'.format(clf.score(X_train,y_train),clf.score(X_test, ...
    y_test)))
```