Abhishek Sinha CSCE 633

Homework Assignment 2

Question 1

(a) $RSS(w_0, w_1) =$

$$\sum_{n=1}^{N}(y_n - w_0 - w_1 X_n)^2$$

Differentiating with respect to w_0 and w_1, we get:

$$2\sum_{n=1}^{N}(y_n - w_0 - w_1 X_n)x_n = 0 \tag{1}$$

$$2\sum_{n=1}^{N}(y_n - w_0 - w_1 X_n) = 0 \tag{2}$$

Solving equation (2) for w_0 we get:

$$w_0 = \frac{1}{N}\sum_{n=1}^{N} y_n - \frac{1}{N} w_1 \sum_{n=1}^{N} x_n \tag{3}$$

Putting Equation (3) back in equation (1) and solving for w_1 we get:

$$w_1 \sum_{n=1}^{N} x_n^2 = \sum_{n=1}^{N} y_n x_n - (\frac{1}{N}\sum_{n=1}^{N} y_n - w_1 \frac{1}{N}\sum_{n=1}^{N} x_n)\sum_{n=1}^{N} x_n$$

$$w_1 = \frac{\sum_{n=1}^{N} y_n x_n - \frac{1}{N}(\sum_{n=1}^{N} y_n \sum_{n=1}^{N} x_n)}{\sum_{n=1}^{N} x_n^2 - \frac{1}{N}\sum_{n=1}^{N} x_n \sum_{n=1}^{N} x_n}$$

which can also be written as :

$$w_1 = \frac{\sum_{n=1}^{N} x_n y_n - N(\frac{1}{N}\sum_{n=1}^{N} y_n \frac{1}{N}\sum_{n=1}^{N} x_n)}{\frac{1}{N}\sum_{n=1}^{N} x_n^2 - N(\frac{1}{N}\sum_{n=1}^{N} x_n)^2} \tag{4}$$

(b) Given $\bar{x} = \frac{1}{N}\sum_{n=1}^{N} x_n$ and $\bar{y} = \frac{1}{N}\sum_{n=1}^{N} y_n$, equation (3) can be written as :

$$w_0 = \bar{y} - w_1 \bar{x}$$

Given the following:

$$w_1^* = \frac{\sum_{n=1}^{N}(x_n - \bar{x})(y_n - \bar{y})}{\sum_{n=1}^{N}(x_n - \bar{x})^2} \tag{5}$$

On expanding and simplifying, we get :

$$w_1^* = \frac{\sum_{n=1}^{N}(x_n y_n - \bar{x}y_n - \bar{y}x_n + \bar{x}\bar{y})}{\sum_{n=1}^{N}(x_n)^2 + \sum_{n=1}^{N}\bar{x}^2 - 2\sum_{n=1}^{N}\bar{x}x_n}$$

Using the definition of $\bar{x}$ and $\bar{y}$ and upon simplifying we get:

$$w_1^* = \frac{\sum_{n=1}^{N} x_n y_n - \frac{1}{N}\sum_{n=1}^{N} x_n \sum_{n=1}^{N} y_n - \frac{1}{N}\sum_{n=1}^{N} y_n \sum_{n=1}^{N} x_n + \frac{1}{N}\sum_{n=1}^{N} y_n \frac{1}{N}\sum_{n=1}^{N} x_n \sum_{n=1}^{N} 1}{\sum_{n=1}^{N}(x_n)^2 + \frac{1}{N}\sum_{n=1}^{N} x_n \frac{1}{N}\sum_{n=1}^{N} x_n \sum_{n=1}^{N} 1 - 2\frac{1}{N}\sum_{n=1}^{N} x_n \sum_{n=1}^{N} x_n}$$

Cancelling 3rd and 4th term in numerator and subtracting 2nd and 3rd term in denominator, we get :

$$w_1^* = \frac{\sum_{n=1}^{N} x_n y_n - \frac{1}{N}\sum_{n=1}^{N} x_n \sum_{n=1}^{N} y_n}{\sum_{n=1}^{N}(x_n)^2 - \frac{1}{N}\sum_{n=1}^{N} x_n \sum_{n=1}^{N} x_n}$$

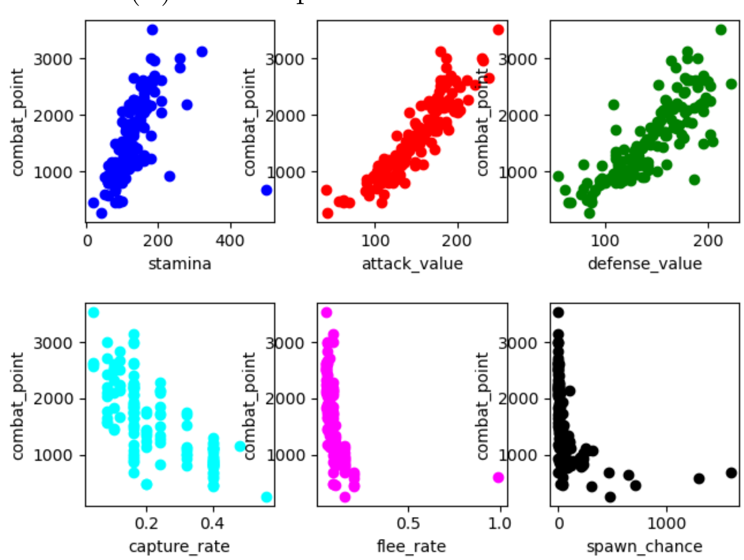
which can be written as equation 4. Thus both the equation (4) and (5) are equivalent.

(c) $\bar{x}$ and $\bar{y}$ are the sample means, while w_1 can be interpreted as the co-variance. w_0 can be interpreted as the intercept of line with slope as the covariance of the sample.

Question 2

(i) The categorical attributes is "*primary_strength*" while numerical attributes are "*stamina*" ,"*attack_value*","*defense_value*","*capture_rate*","*flee_rate*" and, "*spawn_chance*".

(ii) Scatter plots:



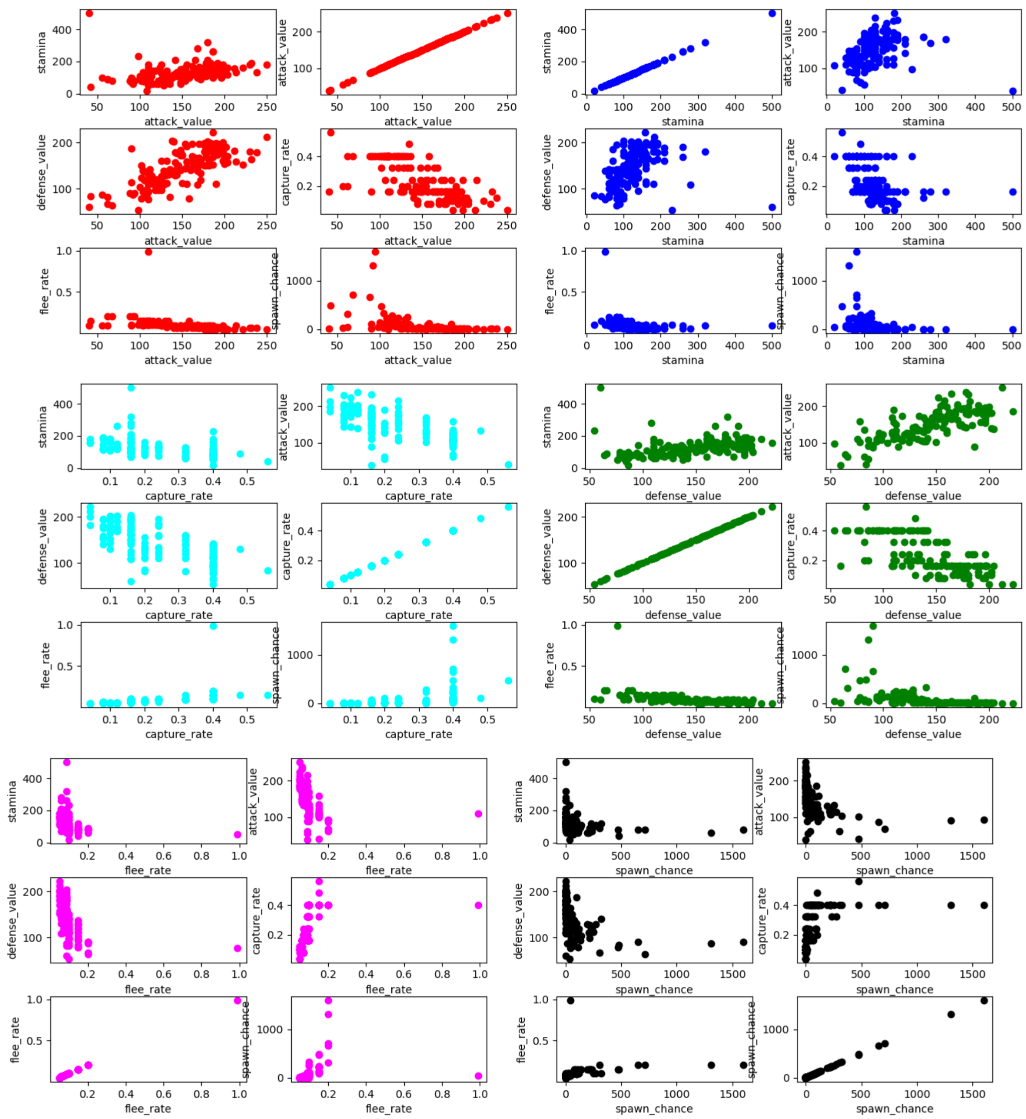
Homework Assignment 2

```
In [9]: correlation = {}
        for i in range(1,7):
            correlation['combat_point vs '+data.keys()[i]] = data['combat_point'].corr(data.iloc[:,i], method = 'pearson')
        correlation

Out[9]: {'combat_point vs stamina': 0.5828317032229261,
         'combat_point vs attack_value': 0.9075315401042738,
         'combat_point vs defense_value': 0.8262293053572938,
         'combat_point vs capture_rate': -0.7430078083529403,
         'combat_point vs flee_rate': -0.4070342114215966,
         'combat_point vs spawn_chance': -0.42132699465983603}
```

Based on scatter plots and correlation coefficients, it would seem that attack value, stamina and defense value would be most predictive of combat points.

(iii) Scatter of different numerical attributes wrt each other are as follows:



It can be observed that attack value, stamina and defense value are indeed correlated, which can be verified from the correlation coefficients.

In [8]:
```
final_df = data.iloc[:, 1:7]
final_df.corr(method = 'pearson')
```

Out[8]:

	stamina	attack_value	defense_value	capture_rate	flee_rate	spawn_chance
stamina	1.000000	0.302995	0.302663	-0.446850	-0.271048	-0.276420
attack_value	0.302995	1.000000	0.736777	-0.690573	-0.369064	-0.432648
defense_value	0.302663	0.736777	1.000000	-0.697266	-0.423860	-0.432499
capture_rate	-0.446850	-0.690573	-0.697266	1.000000	0.440512	0.472793
flee_rate	-0.271048	-0.369064	-0.423860	0.440512	1.000000	0.293222
spawn_chance	-0.276420	-0.432648	-0.432499	0.472793	0.293222	1.000000

(iv)
```
import numpy as np
from matplotlib import pyplot as plt
import pandas as pd
from sklearn import preprocessing
from sklearn.model_selection import KFold, train_test_split
import math

data = pd.read_csv("hw2_data.csv")

#add the one-hot encoding and remove the categorical column
df= pd.get_dummies(data['primary_strength'])
data = data.drop('primary_strength',axis=1)
data = data.join(df)
data.head()

 #Add bias term and pre-process the matrix X and Y
data['bias'] = 1
Y = data['combat_point']
data= data.drop('combat_point', axis=1)
X = data.iloc[:,1:]
main_cols = ['bias']+ X.columns.to_list()[:-1]
X = X[main_cols]
```

(v) KFold spliting and cross validation. Adding the one hot encoding, the model has 22 parameters.

```
#Store the split train and test data sets
kf = KFold(n_splits=5, shuffle=False)
X_ = X.to_numpy()
Y_ = Y.to_numpy()
X_train_set = []
X_test_set = []
Y_train_set = []
Y_test_set = []
split_data =  kf.split(X_)
for train_index, test_index in split_data:
    X_train_set.append(X_[train_index,:])
    Y_train_set.append(Y_[train_index].reshape(X_[train_index].shape
   [0],1))
```

```
    X_test_set.append(X_[test_index,:])
    Y_test_set.append(Y_[test_index].reshape(X_[test_index].shape
   [0],1))

perf =[]
for i in range (len(X_train_set)):
    X_train, X_test = X_train_set[i], X_test_set[i]
    Y_train, Y_test = Y_train_set[i], Y_test_set[i]
    X_train_transpose = X_train.T
    interm_matrix = np.matmul(X_train_transpose, X_train)
    #pseudo inverse
    inv_mat = np.linalg.pinv(interm_matrix)
    left_part =  np.matmul(X_train_transpose, Y_train)
    weights_star = np.matmul(inv_mat,left_part)
    perf.append(math.sqrt(1/(Y_test.shape[0])*sum((Y_test - np.matmul(
   X_test,weights_star))**2)))
perf
```

```
Out[16]: [238.0951048820434,
          98.24274381323976,
          143.9576211490135,
          276.4379767927157,
          170.3166248683674]
```

```
avg_error = sum(perf)/len(perf)
avg_error
```

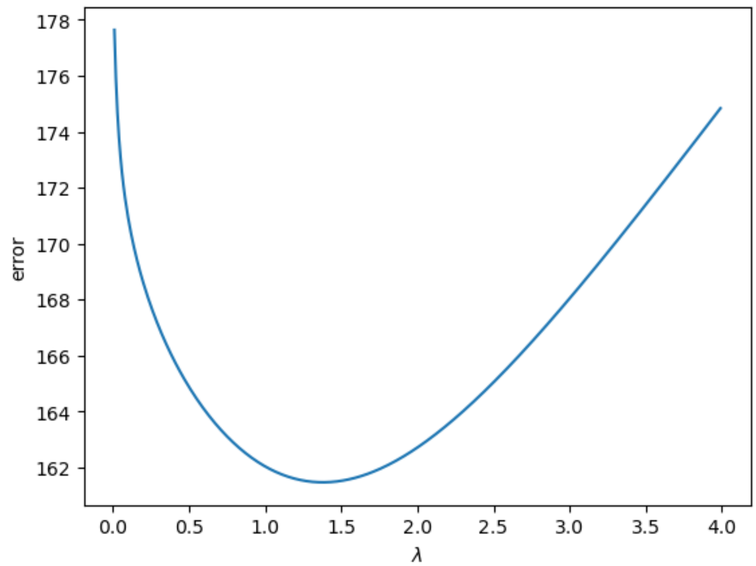
```
Out[17]: 185.41001430107593
```

(vi) KFold with regularization:

```
#L2 normalized least square solution
perf_reg = []
reglambda = np.arange (0.01, 4, 0.01)
error_mat = []
for reg_lambda in reglambda:
    for i in range (len(X_train_set)):
        X_train_reg, X_test_reg = X_train_set[i], X_test_set[i]
        Y_train_reg, Y_test_reg = Y_train_set[i], Y_test_set[i]
        X_train_transpose_reg = X_train_reg.T
        interm_matrix_reg = np.matmul(X_train_transpose_reg,
   X_train_reg) + reg_lambda*np.eye(X_train_transpose_reg.shape[0])
        #pseudo inverse
        inv_mat_reg = np.linalg.pinv(interm_matrix_reg)
        left_part_reg =  np.matmul(X_train_transpose_reg, Y_train_reg)
        weights_star_reg = np.matmul(inv_mat_reg,left_part_reg)
        perf_reg.append(math.sqrt(1/(Y_test_reg.shape[0])*sum((
   Y_test_reg - np.matmul(X_test_reg, weights_star_reg))**2)))
```

```python
    avg_error = sum(perf_reg)/len(perf_reg)
    error_mat.append(avg_error)

plt.plot(reglambda, error_mat)
plt.ylabel("error")
plt.xlabel('$\lambda$')
plt.show()
```




```python
#BEST LAMBDA = 0.83 (APPROX)
avg_error = sum(perf_reg)/len(perf_reg)
avg_error
```

```
Out[19]: 158.77773308988202
```

(vii) Different feature combination (Feature set - 1: Stamina, Attack and Defence):

```python
#Assuming best correlation is only if correlation factor is greater than approx 0.5 and scatter plot shows a definite trend
#Thus only attack, defense and stamina rates matter
perf_reg = []
for i in range (len(X_train_set)):
    X_train_reg, X_test_reg = X_train_set[i][:,:4], X_test_set[i][:,:4]
    Y_train_reg, Y_test_reg = Y_train_set[i], Y_test_set[i]
    X_train_transpose_reg = X_train_reg.T
    interm_matrix_reg = np.matmul(X_train_transpose_reg, X_train_reg)
    #pseudo inverse
    inv_mat_reg = np.linalg.pinv(interm_matrix_reg)
```

```
    left_part_reg =  np.matmul(X_train_transpose_reg, Y_train_reg)
    weights_star_reg = np.matmul(inv_mat_reg,left_part_reg)
    perf_reg.append(math.sqrt(1/(Y_test_reg.shape[0])*sum((Y_test_reg
   - np.matmul(X_test_reg, weights_star_reg))**2)))
perf_reg
```

```
Out[20]: [151.17337580469237,
          77.35685697094283,
          91.08787186310748,
          285.7501305347148,
          176.1470577045314]
```

```
avg_error = sum(perf_reg)/len(perf_reg)
avg_error
```

```
Out[21]: 156.3030585755978
```

(Feature set - 2: Attack and Defence only):

```
#Assuming best correlation is only if correlation factor is greater
   than approx 0.75 and scatter plot shows a definite trend
#Thus only attack and defense rates matter
perf_reg = []
for i in range (len(X_train_set)):
    X_train_reg, X_test_reg = X_train_set[i][:,[0,2,3]], X_test_set[i
   ][:,[0,2,3]]
    Y_train_reg, Y_test_reg = Y_train_set[i], Y_test_set[i]
    X_train_transpose_reg = X_train_reg.T
    interm_matrix_reg = np.matmul(X_train_transpose_reg, X_train_reg)
    #pseudo inverse
    inv_mat_reg = np.linalg.pinv(interm_matrix_reg)
    left_part_reg =  np.matmul(X_train_transpose_reg, Y_train_reg)
    weights_star_reg = np.matmul(inv_mat_reg,left_part_reg)
    perf_reg.append(math.sqrt(1/(Y_test_reg.shape[0])*sum((Y_test_reg
   - np.matmul(X_test_reg, weights_star_reg))**2)))
perf_reg
```

```
Out[22]: [160.34211936090574,
          216.32956433947814,
          201.75065066951126,
          287.46289183399915,
          333.2310056952472]
```

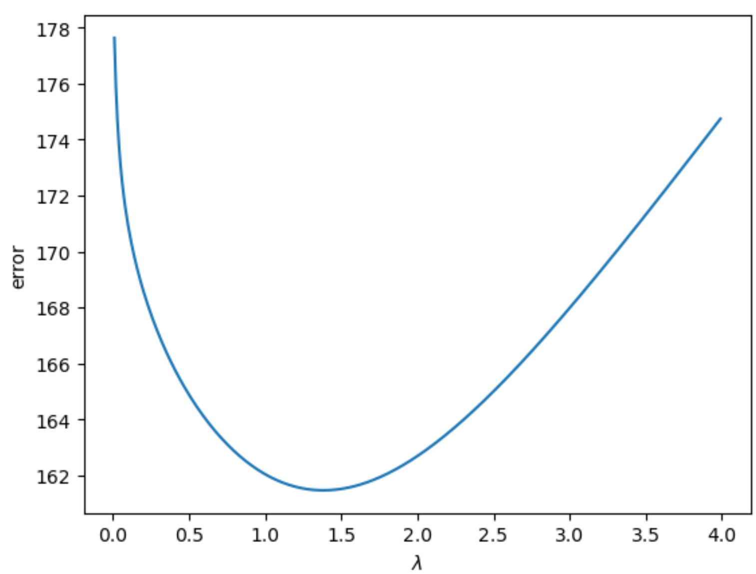
```
avg_error = sum(perf_reg)/len(perf_reg)
avg_error
```

```
Out[23]: 239.82324637982828
```

(**viii**) With L1 normalization:

```
#L1 normalization
perf_reg = []
reglambda = np.arange (0.01, 4, 0.01)
error_mat = []
for reg_lambda in reglambda:
    for i in range (len(X_train_set)):
        X_train_reg, X_test_reg = X_train_set[i], X_test_set[i]
        Y_train_reg, Y_test_reg = Y_train_set[i], Y_test_set[i]
        X_train_transpose_reg = X_train_reg.T
        interm_matrix_reg = np.matmul(X_train_transpose_reg,
    X_train_reg) + reg_lambda*np.eye(X_train_transpose_reg.shape[0])
        #pseudo inverse
        inv_mat_reg = np.linalg.pinv(interm_matrix_reg)
        left_part_reg =  np.matmul(X_train_transpose_reg, Y_train_reg)
     - reg_lambda*np.ones([X_train_transpose_reg.shape[0],1])
        weights_star_reg = np.matmul(inv_mat_reg,left_part_reg)
        perf_reg.append(math.sqrt(1/(Y_test_reg.shape[0])*sum((
    Y_test_reg - np.matmul(X_test_reg, weights_star_reg))**2)))
    avg_error = sum(perf_reg)/len(perf_reg)
    error_mat.append(avg_error)

plt.plot(reglambda, error_mat)
plt.ylabel("error")
plt.xlabel('$\lambda$')
plt.show()
```




```
#Best lambda = 0.875 (approx)
avg_error = sum(perf_reg)/len(perf_reg)
avg_error
```

```
Out[25]: 158.7525461244108
```

Seems L1 performs slightly better than L2 normalization.

(**ix**) Binarization of data and logistic regression:

```
Y_d = Y>Y.describe()['mean']
Y_d = Y_d.astype(int).to_numpy()

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
X_train, X_test, Y_train, Y_test = train_test_split(X_,Y_d, test_size
   = 0.2)

logisticRegr = LogisticRegression(penalty = 'none')
logisticRegr.fit(X_train, Y_train)
score = logisticRegr.score(X_test, Y_test)
score
```

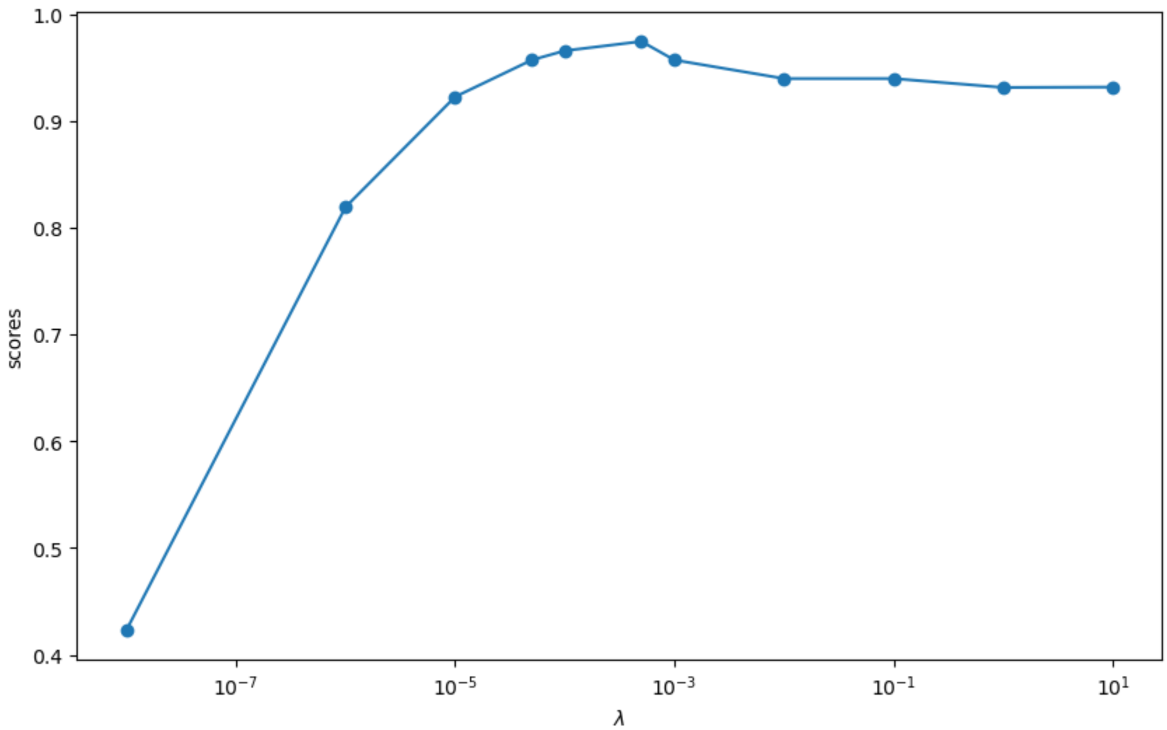
The score comes out to be 0.8 on test data

(**x**) Logistic Regression with regularization

```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
kf = KFold(n_splits=5, shuffle=True)
perf = []
X_train, X_test, Y_train, Y_test = train_test_split(X_,Y_d, test_size
   = 0.2)
logisticRegr = LogisticRegression(penalty = 'l2',solver = 'newton-cg',
    max_iter = 100000 )
parameters = {'C': [1e-8, 1e-6, 1e-5, 5e-5, 1e-4, 5e-4, 1e-3, 1e-2, 1e
   -1, 1.0, 10.0]}
gs = GridSearchCV(logisticRegr, param_grid=parameters, cv=kf, scoring=
   "accuracy")
gs.fit(X_train, Y_train)
gs.cv_results_

fit_lambdas = [d['C'] for d in gs.cv_results_['params']]
fit_scores = gs.cv_results_['mean_test_score']

fig, ax = plt.subplots(1,1, figsize=(10,6))
ax.plot(fit_lambdas, fit_scores, ls='-', marker='o')
ax.set_xscale('log')
ax.set_xlabel('$\lambda$')
ax.set_ylabel('scores');
plt.show()
```




```
index = np.where(fit_scores == max(fit_scores))
lamdas = fit_lambdas[index[0][0]]

logisticRegr = LogisticRegression(penalty = 'l2',solver = 'newton-cg',
    max_iter = 10000, C = lamdas)
logisticRegr.fit(X_train, Y_train)
score = logisticRegr.score(X_test, Y_test)
print(score)
```

Here the final score comes out to be 0.9667 on the test set