Assignment_1

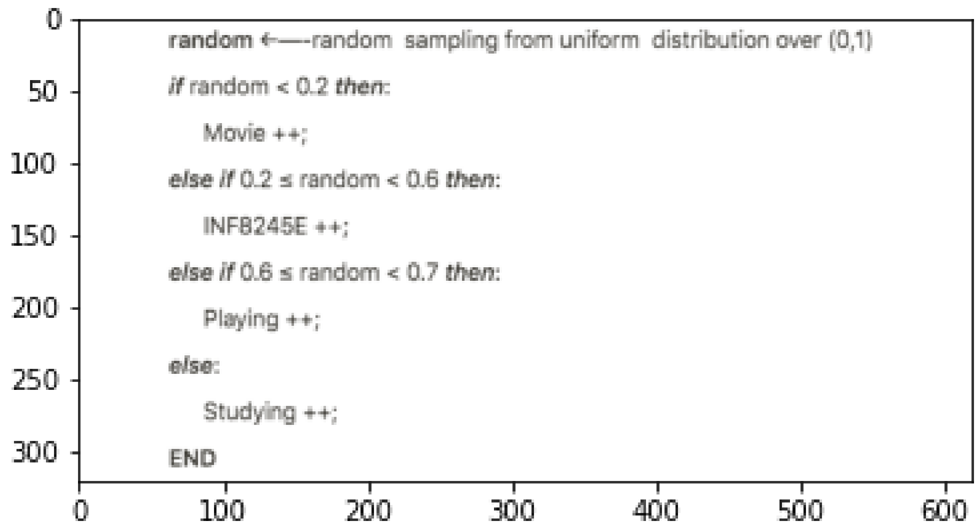
October 4, 2021

1 Zahra Parham 2122841

1.1 1 Sampling

1.1.1 1.1

```
[ ]: import matplotlib.pyplot as plt
import matplotlib.image as mpimg

img=mpimg.imread('1.jpg')
imgplot = plt.imshow(img)
```



1.1.2 1.2

```
[ ]: import numpy as np
test = np.zeros(4)
test[0]= 4
test

def prob(n):
```

```
    result = np.zeros(4)
    for i in range(n):
        random = np.random.uniform(0,1)
        if random < 0.2:
            result[0] = result[0] + 1
        if random >= 0.2 and random< 0.6:
            result[1] = result[1] + 1

        if random >= 0.6 and random<0.7:
            result[2] = result[2] + 1

        if random>=0.7 and random <=1:
            result[3] = result[3] + 1

    return result / n
```

[ ]:
```
result_100 = prob(100)
print(f"in 100 days - movie: {result_100[0]}, INF8245E:{result_100[1]}, playing:
↪{result_100[2]}, studying:{result_100[3]} ")
result_1000 = prob(1000)
print(f"in 1000 days - movie: {result_1000[0]}, INF8245E:{result_1000[1]},␣
↪playing:{result_1000[2]}, studying:{result_1000[3]} ")
```

```
in 100 days - movie: 0.22, INF8245E:0.34, playing:0.07, studying:0.37
in 1000 days - movie: 0.198, INF8245E:0.391, playing:0.079, studying:0.332
```

Based on comparing the fraction in 100 days and 1000 days, we can assume that by increasing the number of sampling, each activity will get closer to its probability

1.1.3 2 Model Selection

1.1.4 2.1

[ ]:
```
# first I split train data, test data and validation data to x and y:
import pandas as pd
dataset_test = pd.read_csv("Datasets/Dataset_1_test.csv", header = None)
test_x = dataset_test[0]
test_y = dataset_test[1]
```

[ ]:
```
dataset_train = pd.read_csv("Datasets/Dataset_1_train.csv", header = None)
train_x = dataset_train[0]
train_y = dataset_train[1]
```

[ ]:
```
dataset_valid =pd.read_csv("Datasets/Dataset_1_valid.csv", header = None)
valid_x = dataset_valid[0]
```

```
valid_y = dataset_valid[1]
```

1.1.5 1.a

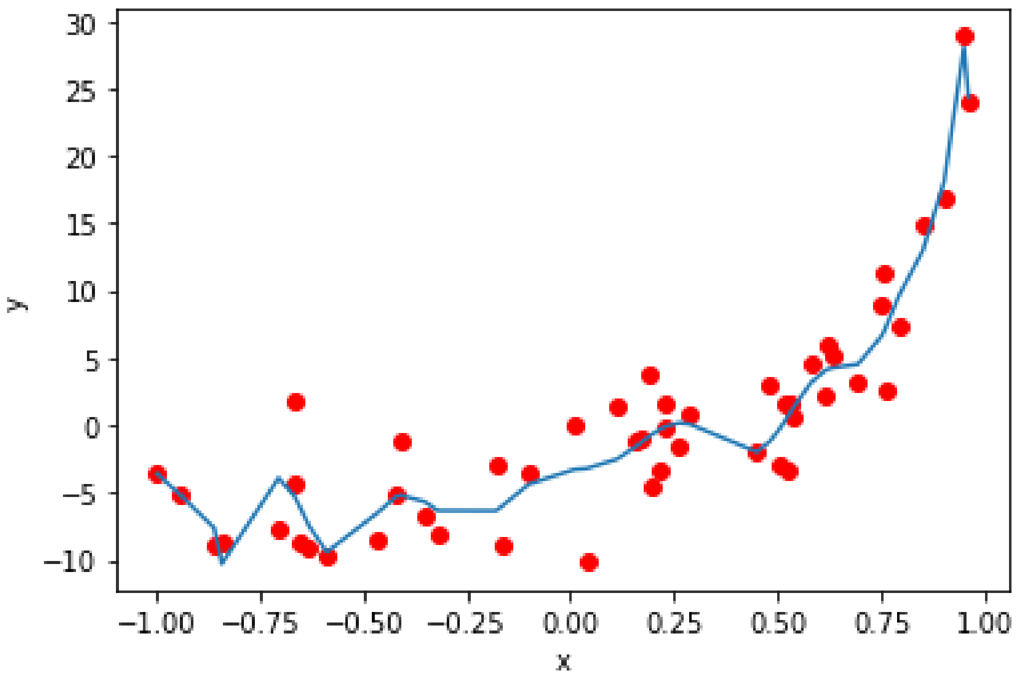
[ ]:
```
#cover x to a 50*20 matrix
def final_x(x):
    result = np.zeros((50,20))
    for i in range(50):
        for j in range(20):
            result[i][j] = x[i] ** j
    return result



#compute RSME:
def f_rmse(x, y, w):
    y_hat = np.dot(x, w)
    return np.sqrt(np.sum((y_hat - y) ** 2 )/ 50)

final_x_train = final_x(train_x)
final_x_valid = final_x(valid_x)

#Compute W:
def w(x , y):
    x_t = np.transpose(x)
    x_t_x = np.dot(x_t, x)
    x_t_x_inv = np.linalg.inv(x_t_x)
    x_t_y = np.dot(x_t, y)
    w = np.dot(x_t_x_inv, x_t_y)
    return w
w = w(final_x_train, train_y)
rmse_valid = f_rmse(final_x_valid, valid_y, w)
rmse_train = f_rmse(final_x_train, train_y, w)


print(f"Train RMSE:{rmse_train}, Validation RMSE{rmse_valid}")
```

```
Train RMSE:2.6774364810380065, Validation RMSE18.352463341124924
```

1.1.6 1.b

[ ]:
```
import itertools
new_x, new_y = zip(*sorted(zip(train_x, np.dot(final_x_train, w))))
import matplotlib.pyplot as plt

plt.plot(train_x, train_y, 'ro')
```

```
plt.plot(new_x, new_y)

plt.xlabel("x")
plt.ylabel("y")

plt.show()

#plot all of them after
```



1.1.7 1.c

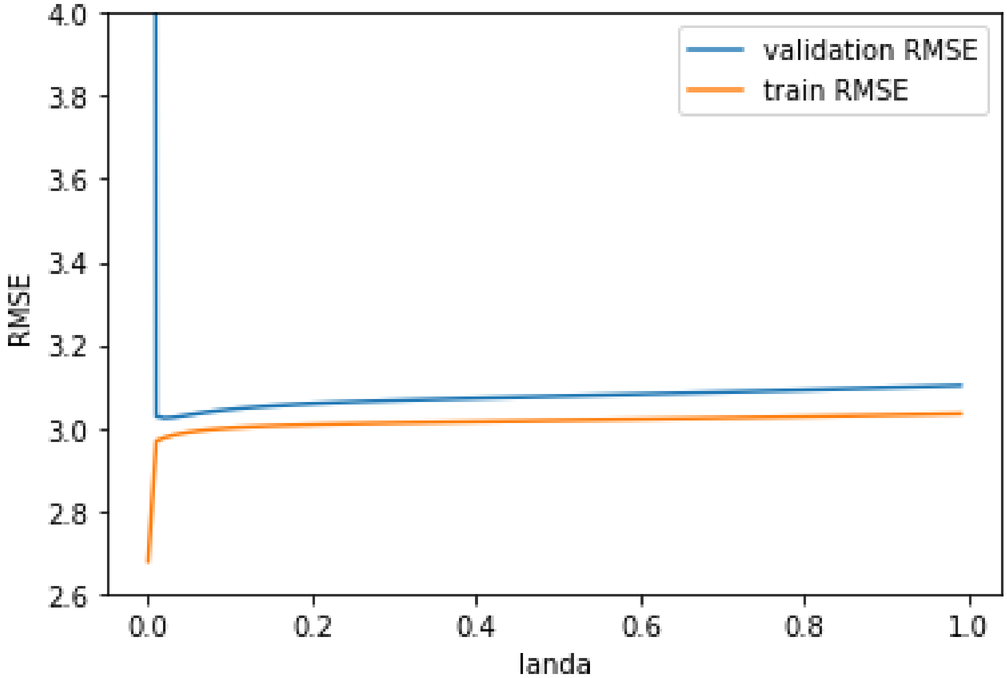
The model is overfitting since, based on the plot, it is fit to the train data too closely, and also based on comparing the RMSE for train data and test data, we can assume that the model is overfitting.

1.1.8 2

1.1.9 2.a

[ ]:
```
#cover x to a 50*20 matrix
def final_x(x):
    result = np.zeros((50,20))
    for i in range(50):
        for j in range(20):
            result[i][j] = x[i] ** j
    return result
```

```python
#compute RSME:
def f_rmse(x, y, w):
    y_hat = np.dot(x, w)
    return np.sqrt(np.sum((y_hat - y) ** 2 )/ 50)

final_x_train = final_x(train_x)
final_x_valid = final_x(valid_x)

#Compute W:
def w_reg(x , y, landa):
    I = np.identity(20)
    x_t = np.transpose(x)
    x_t_x = np.dot(x_t, x)
    x_t_landa =x_t_x  + np.dot(landa, I)
    x_t_x_inv = np.linalg.inv(x_t_landa)
    x_t_y = np.dot(x_t, y)
    w = np.dot(x_t_x_inv, x_t_y)
    return w



rmse_reg_valid = []
rmse_reg_train = []

for landa in np.arange(0, 1, 0.01):
    w = w_reg(final_x_train, train_y, landa)
    rmse_reg_valid.append(f_rmse(final_x_valid, valid_y, w))
    rmse_reg_train.append(f_rmse(final_x_train, train_y, w))



x= np.arange(0,1,0.01)
plt.plot(x, rmse_reg_valid, label = "validation RMSE")
plt.plot(x, rmse_reg_train, label = "train RMSE")
plt.xlabel("landa")
plt.ylabel("RMSE")
plt.ylim([2.6, 4])
plt.legend()
plt.show()
```



1.1.10 2.b

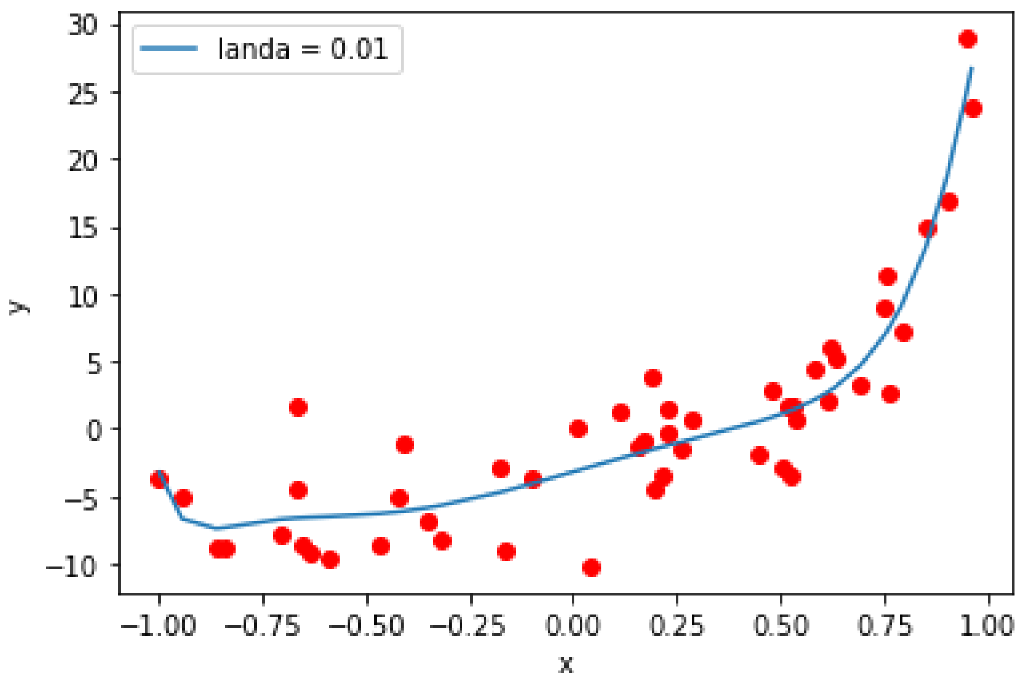
best landa based on the plot is 0.01

[ ]:
```
best_w =w_reg(final_x_train, train_y, 0.01)

rmse_reg_test = f_rmse(final_x(test_x), test_y, best_w)
print(f"RMSE for test data with the best landa = 0.01 is: {rmse_reg_test}")
```

```
RMSE for test data with the best landa = 0.01 is: 3.2925595397745555
```

1.1.11 2.c

[ ]:
```
import itertools
new_x, new_y = zip(*sorted(zip(train_x, np.dot(final_x_train, best_w))))
import matplotlib.pyplot as plt

plt.plot(train_x, train_y, 'ro')
plt.plot(new_x, new_y, label ="landa = 0.01")

plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```



1.1.12 2.d

This is a good fit for the data because:

1- based on the plot, it does not fit too closely or does not have enough flexibility in terms of line fitting(learning too much or too small)

2- based on the RMSE on test and train data, we can assume that the model is not overfitting or underfitting since it has good RMSE(result) on both the train and test dataset

1.1.13 3

it's hard to infer the exact degree from the last plot but from comparing the last plot with the other degree polynomial it can be a third-degree polynomial

1.1.14 3 Gradient Descent for Regression

1.1.15 3.1

[ ]:
```
import pandas as pd
dataset_train = pd.read_csv("Datasets/Dataset_2_train.csv", header = None)

train_x = dataset_train[0]
train_x = train_x.to_numpy()
train_y = dataset_train[1]
train_y= train_y.to_numpy()
```

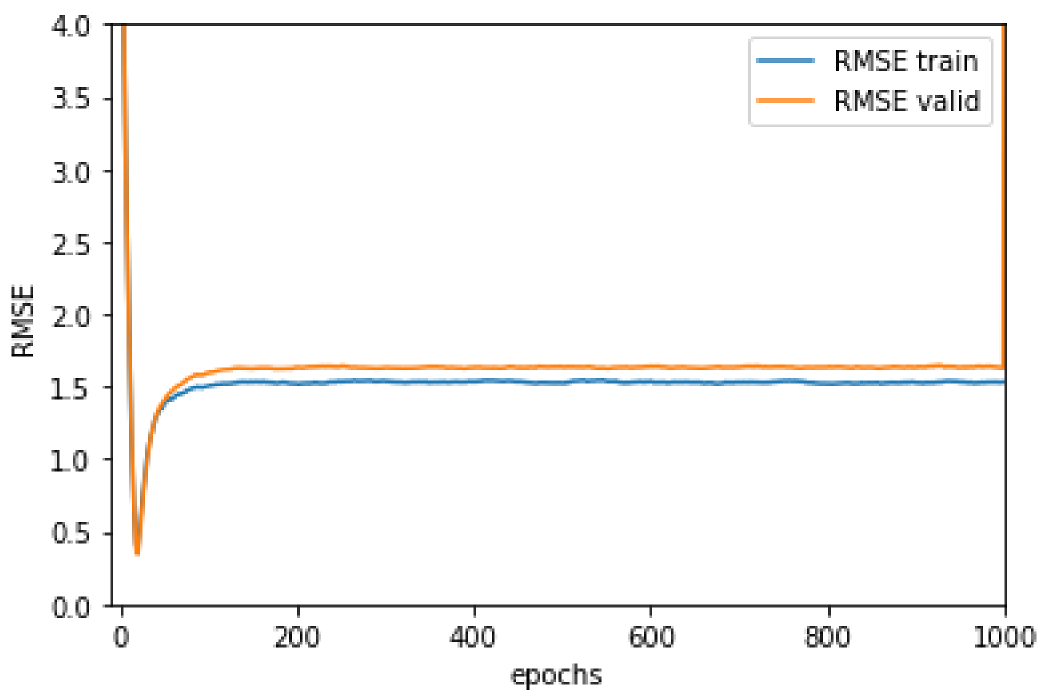
```python
dataset_test = pd.read_csv("Datasets/Dataset_2_test.csv", header = None)
test_x = dataset_test[0]
test_x = test_x.to_numpy()
test_y = dataset_test[1]
test_y = test_y.to_numpy()

dataset_valid = pd.read_csv("Datasets/Dataset_2_valid.csv", header = None)
valid_x = dataset_valid[0]
valid_x = valid_x.to_numpy()
valid_y = dataset_valid[1]
valid_y = valid_y.to_numpy()
```

[ ]:
```python
def SGD(x, y, epochs, learning_rate):
    theta = [0 for i in range(1)]
    b = np.zeros(1)
    result = []
    theta_ = []
    b_ = []
    for e in range(epochs):
        for i in range(len(x)):
            random = np.random.randint(len(x))
            rand_x = x[random:random+1]
            rand_y = y[random:random+1]

            gradient = 2 * np.dot(np.transpose(rand_x),(np.dot(rand_x, theta) -␣
 ↪rand_y))
            gradient_b = 2 * (((np.dot(rand_x, theta)) + b) - rand_y)
            theta = theta - learning_rate * gradient
            b = b - learning_rate * gradient_b

        theta_.append(theta[0])
        b_.append(b)
        #result.append(RMSE(x, y, theta[0], b))

    return  theta_, b_

def RMSE(x, y, theta, b):

    y_hat = np.dot(x, theta) + b
    rmse = np.sum((y - y_hat) ** 2) / int(len(y))
    return np.sqrt(rmse)
```

1.1.16 1.a

[ ]:
```
theta, b = SGD(train_x, train_y, 1000, 0.0001)

rmse_train = []
for i in range(len(theta)):
    rmse_train.append(RMSE(train_x, train_y, theta[i], b[i]))
    rmse_valid.append(RMSE(valid_x, valid_y, theta[i], b[i]))
```

[ ]:
```
plt.plot(rmse_train, label="RMSE train")
plt.plot(rmse_valid, label="RMSE valid")
plt.xlabel("epochs")
plt.ylabel("RMSE")
plt.ylim([0, 4])
plt.xlim([-10, 1000])


plt.legend()
plt.show()
```



1.1.17 2

1.1.18 2.a

[ ]:
```
rmse_2 = []
for alpha in [0.0001,0.001,0.01,0.1,0.5]:
    theta, b = SGD(train_x, train_y, 1000, alpha)
    rmse = RMSE(valid_x, valid_y, theta[-1], b[-1])
    rmse_2.append(rmse)
```

[ ]:
```
final = np.array([["step size", "rmse"],[0.0001, rmse_2[0]], [0.001,␣
↪rmse_2[1]], [0.01, rmse_2[2]], [0.1, rmse_2[3]], [0.5, rmse_2[4]]])
f = pd.DataFrame(final)
f
```

[ ]:

	0	1
0	step size	rmse
1	0.0001	1.63535537775918
2	0.001	1.62534367284661
3	0.01	1.685956558252589
4	0.1	1.7243225040018213
5	0.5	1.932250067612219

The best step size is 0.001

1.1.19 2.b

[ ]:
```
alpha_test = 0.001
theta = np.zeros(1)
b = np.zeros(1)
theta, b= SGD(train_x, train_y, 1000, 0.001)
rmse_test = RMSE(test_x, test_y, theta[-1], b[-1])

rmse_test
```

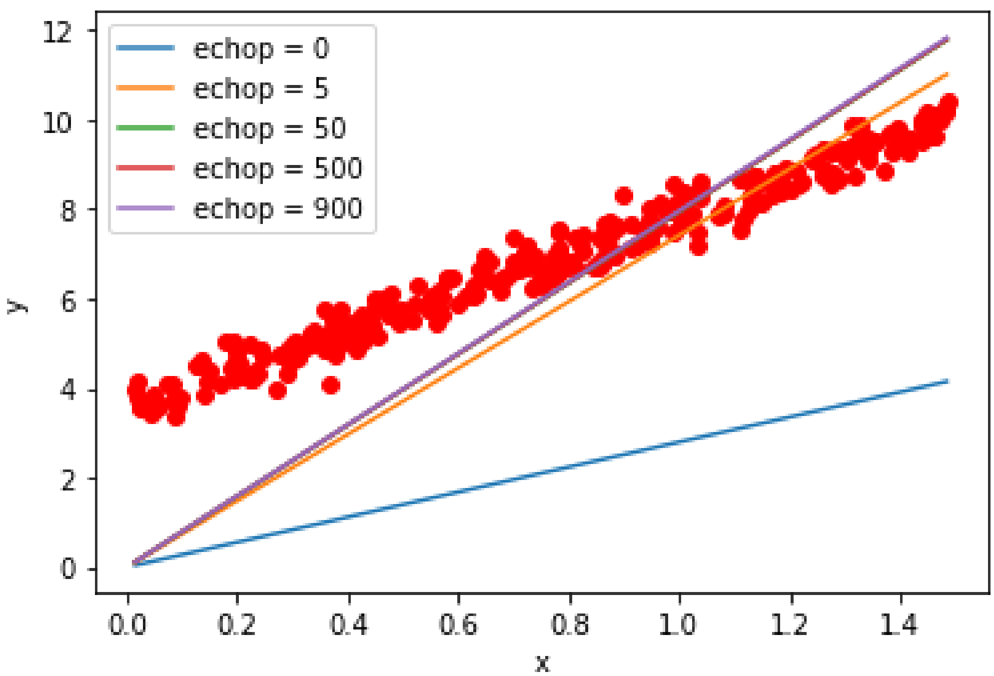
[ ]:
```
1.6147289304723258
```

1.1.20 3

[ ]:
```
rand = np.random.randint(0, high=1000, size=5)
print(rand)
new_x = []
new_y = []

rand = [0, 5, 50, 500, 900]
for i in rand:
```

```python
    print(theta[i])
    new_x_, new_y_ = zip(*sorted(zip(train_x, np.dot(train_x, theta[i]))))
    new_x.append(new_x_)
    new_y.append(new_y_)

plt.plot(train_x, train_y, 'ro')
for i in range(5):
    plt.plot(new_x[i], new_y[i], label = f'echop = {rand[i]}')
plt.xlabel("x")
plt.ylabel("y")
plt.legend()
plt.show()
```

```
[355 984 939 900 152]
2.794079206210347
7.417099464382808
7.934548518036762
7.943282729765617
7.967270346953026
```

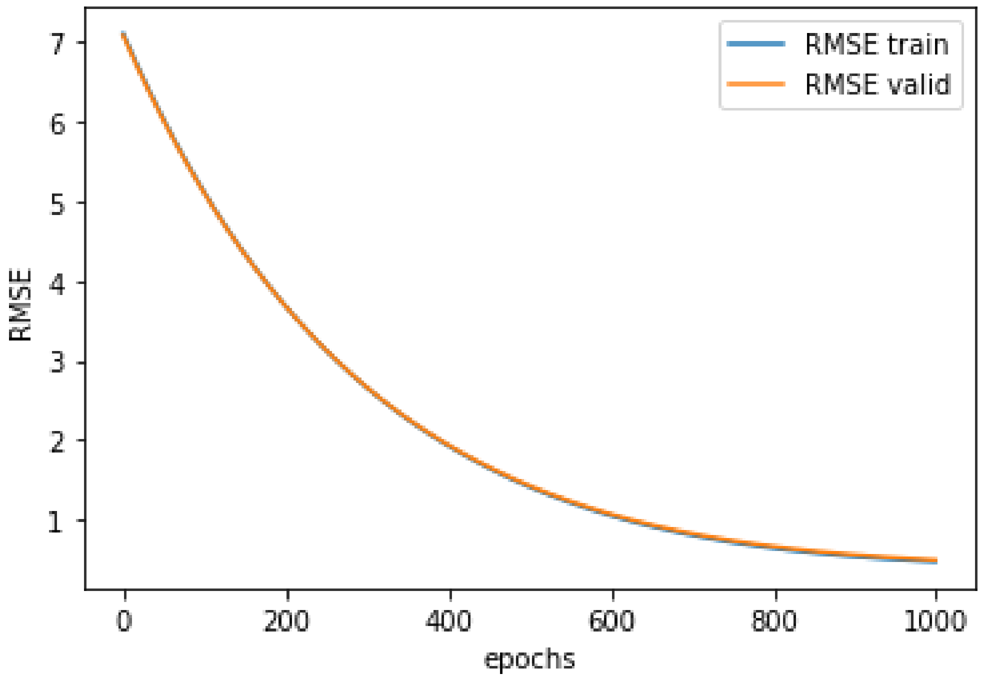


1.1.21 4

[ ]:
```
def RMSE(x, y, theta, b):
    x = x.reshape(len(x), 1)

    y_hat = np.dot(x, theta) + b
    rmse = np.sum((y - y_hat) ** 2) / int(len(y))
    return np.sqrt(rmse)
def LinearRegression(x, y, learning_rate, iteration):
    w_ = []
    b_ = []
    m = x.shape
    x= x.reshape((m[0],1))
    w = np.zeros(1)
    b = 0

    for i in range(iteration):
        y_hat = np.dot(x, w) + b
        gradient_w = 2 * np.dot(np.transpose(x), (y_hat - y)) / m
        gradient_b = 2 * np.sum((y_hat - y)) / m

        w = w - learning_rate * gradient_w
        b = b - learning_rate * gradient_b
        w_.append(w)
        b_.append(b)

    return w_, b_
```

[ ]:
```
w,b = LinearRegression(train_x, train_y, 0.001, 1000)

rmse_train_full = []
rmse_valid_full = []
for i in range(len(w)):
    rmse_train_full.append(RMSE(train_x, train_y, w[i], b[i]))
    rmse_valid_full.append(RMSE(valid_x, valid_y, w[i], b[i]))
```

[ ]:
```
plt.plot(rmse_train_full, label="RMSE train")
plt.plot(rmse_valid_full, label="RMSE valid")
plt.xlabel("epochs")
plt.ylabel("RMSE")
#plt.ylim([0, 4])
#plt.xlim([-10, 1000])


plt.legend()
plt.show()
```



1.1.22 5

1- on Full Gradient Descent we run the algorithm through all the data in your training set and do a single update for the parameters(w, b), on the other hand, in SGD we run the algorithm on just one sample or a subset of data in training data.

2- gradient descent (with a large number of data) can be too slow to converge but SGD is more faster since we update the parameter based on small number of samples

3- The error function in SGD is not as minimum as GD

1.1.23 4 Real life dataset

1.1.24 1.a

[80]:
```
import numpy as np
import pandas as pd

data = pd.read_csv("communities.data", header = None)

data_x = data.iloc[: , :-1]
data_y = data[:][127]



data_np = data.to_numpy()
```

[81]:
```
# replace ? with NaN in the dataset

for i in range(len(data)):
    for j in range(128):
        if(data_np[i,j] =='?'):
            data_np[i,j] =np.NaN


data_np
```

[81]:
```
array([[8, nan, nan, …, 0.32, '0.14', 0.2],
       [53, nan, nan, …, 0.0, nan, 0.67],
       [24, nan, nan, …, 0.0, nan, 0.43],
       …,
       [9, '9', '80070', …, 0.91, '0.28', 0.23],
       [25, '17', '72600', …, 0.22, '0.18', 0.19],
       [6, nan, nan, …, 1.0, '0.13', 0.48]], dtype=object)
```

[82]:
```
#compute mean for each column

data_np_mean = np.delete(data_np, obj = 3, axis =1)
data_np_mean = np.array(data_np_mean, dtype = 'float64')
mean = np.nanmean(data_np_mean, axis = 0)
mean_ = np.insert(mean, 3, 0)
```

[83]:
```
#replace the NaN with the mean in each column
data_ = pd.DataFrame(data_np)
for i in range(1994):
    for j in range(128):
        if j != 3:
            if (pd.isna(data_.iloc[i,j])):
                data_.iloc[i,j] = int(mean_[j])


data_.head()
```

[83]:

	0	1	2	3	4	…	123	124	125	126	127
0	8	58	46188	Lakewoodcity	1	…	0.9	0.5	0.32	0.14	0.2
1	53	58	46188	Tukwilacity	1	…	0	0	0	0	0.67
2	24	58	46188	Aberdeentown	1	…	0	0	0	0	0.43
3	34	5	81440	Willingborotownship	1	…	0	0	0	0	0.12
4	42	95	6096	Bethlehemtownship	1	…	0	0	0	0	0.03

[5 rows x 128 columns]

filling the missing data with the mean of each column maybe not be a good choice especially in skewed data and it can also reduce the variance of the data which it can produce bias in our model.

1.1.25 1.b

1- ignore the data that is missing which is not a good way since you might lose some valuable information(Dropping rows with null values, Dropping features with high nullity)

2-imputation using mean/median

3- imputation using the most frequent item

4-imputation using zero or constant

5-imputing using k-nn algorithm

6- linear/stochastic regression imputation

1.1.26 1.c

in regression imputation, we fill the missing data by predicting it by using the regression model. we will predict the missing data with the information of other variables.

In the first step, we fill the missing data with some trivial method like filling with mean of each column, and then the regression model is estimated in the information of other data and using the regression weights to predict the missing data.

1.1.27 1.d

[84]:
```
# drop the column= 3 since it's not numerical and also it does'nt have null␣
↪value!
# this dataset is filled it's missing value with mean
data_missing = data_.drop(3, 1)
data_missing = pd.DataFrame(data_missing, dtype='float64')
data_missing_x = data_missing.iloc[:,0:126]
data_missing_y = data_missing.iloc[:,126]
```

[85]:
```
def rmse(y, x, w, b):
    y_hat = x.dot(w) + b
    error = 0
    for i in range(len(y)):
        error = error + (y_hat[i] - y[i]) ** 2 / int(len(y))
    return np.sqrt(error)

def LinearRegression_(x, y, learning_rate, iteration):
    m, n = x.shape
    w = np.zeros(n)
    b = 0

    for i in range(iteration):
        y_hat = x.dot(w) + b
        gradient_w = 2 * x.T.dot(y_hat - y) / m
        gradient_b = 2 * np.sum(y_hat - y) / m
```

```
        w = w - learning_rate * gradient_w
        b = b - learning_rate * gradient_b

    return w, b
```

[86]:
```
import numpy as np
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(data_missing_x, data_missing_y)
print(reg.coef_, reg.intercept_)
```

```
[-5.81977810e-04 -1.54405177e-04 -1.79304298e-07 -1.53695737e-03
  2.94448220e-01 -1.19297032e-02  1.69653968e-01 -7.10430619e-02
 -2.99702134e-02  5.66919652e-02  1.32843690e-01 -2.47822371e-01
 -1.59427354e-01  2.57871586e-02 -3.66317624e-01  5.23098908e-02
 -1.98329937e-01 -1.91265748e-01  4.20903831e-02 -1.72333341e-01
  9.31914924e-02 -3.01743673e-03 -9.20090292e-02  3.01334928e-01
  1.43529236e-01 -3.89958880e-01 -3.58785844e-02 -3.20765021e-02
  1.94026257e-02  4.53519511e-02  3.33077478e-02  7.02750645e-02
 -1.80409134e-01 -7.85461693e-02  4.01752144e-02  4.15890018e-02
 -6.46675641e-03  2.53116570e-01 -6.18505867e-02 -1.14657419e-02
  7.09742450e-02  1.15030897e-01  4.56208926e-01  2.28338301e-01
  1.09652235e-01 -5.33832041e-01 -1.79604611e-01  1.36458465e-02
 -3.33735421e-01 -2.73818067e-02  1.62978006e-03  5.15593699e-02
 -1.84623491e-01 -1.42710303e-01  1.20733392e-01 -2.24767158e-01
  2.32318291e-02  1.89032948e-02 -7.34842093e-02  4.50460251e-02
 -3.98726710e-02 -1.79029822e-01  3.86926841e-01 -1.56573471e-01
 -1.87248212e-02 -1.64474515e-01  6.74401372e-03 -1.20676946e-01
  6.36339497e-01 -1.14047536e-01 -2.37083013e-01 -6.44240606e-01
  1.81560577e-01  8.11500277e-02  3.13990007e-02  1.26762762e-01
 -4.73122550e-02  4.87391690e-01  5.26036037e-02 -7.59392155e-02
 -1.60035315e-02  2.91125074e-02 -1.10211187e-02 -3.73363579e-01
  2.44436033e-01  2.77142124e-02 -2.04525380e-01 -2.91155671e-02
 -5.18427770e-02  3.31989436e-01  4.13472185e-02 -4.37191788e-02
 -8.07973136e-02  9.91326164e-02  1.55416833e-01  1.46309399e-01
  2.45742623e-02  2.16698230e-03 -4.74148186e-03  1.91601820e-02
 -1.81366556e-01 -2.08944278e+01  9.67481695e-02  5.01472906e-01
 -1.64109637e-01 -7.43403744e-02  2.03679171e-01  2.07048382e+01
 -9.28892440e-02 -3.88822921e-02  1.73578079e-02  5.92345004e-02
  7.77352485e-02 -4.67989838e-02 -9.15497856e-04 -2.08935083e-02
 -2.46552604e-02  2.79930383e-02 -2.61931330e-03 -4.01581919e-02
  1.21790756e-01  4.36672188e-01 -5.31849554e-02  3.50910471e-02
 -4.80297606e-02 -2.42434238e-01] 0.6792274604295954
```

[88]:
```
w, b = LinearRegression_(data_missing_x, data_missing_y, 0.01, 100)
rmse = rmse(data_missing_y, data_missing_x, w, b)

data_n = pd.DataFrame(data_np)
```

```
/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:19: RuntimeWarning:
invalid value encountered in double_scalars
```

[89]:
```
data_n = data_n.drop(3,1)
pred = reg.predict(data_missing_x)
pred = pred.round(1)

for i in range(1994):
    for j in range(127):
        if j != 3:
            if (pd.isna(data_n.iloc[i,j])):
                data_n.iloc[i,j] =  (pred[i])

data_n.head()
#data_n is the entire data base which is filled it's missing data with␣
 ↪regression imputation
#I used scikit learn to just get the w and b because my laptop and google colab␣
 ↪could not handle this matrix
```

[89]:

	0	1	2	4	5	6	7	…	121	122	123	124	125	126	127
0	8	0.2	0.2	1	0.19	0.33	0.02	…	0.06	0.04	0.9	0.5	0.32	0.14	0.2
1	53	0.3	0.3	1	0	0.16	0.12	…	0.3	0.3	0.3	0.3	0	0.3	0.67
2	24	0.4	0.4	1	0	0.42	0.49	…	0.4	0.4	0.4	0.4	0	0.4	0.43
3	34	5	81440	1	0.04	0.77	1	…	0.3	0.3	0.3	0.3	0	0.3	0.12
4	42	95	6096	1	0.01	0.55	0.02	…	-0	-0	-0	-0	0	-0	0.03

[5 rows x 127 columns]

1.1.28 2

[90]:
```
#data_ = data_.drop(3,1)
test_size = int(data.shape[0] * 0.2)
test_data = data_n.iloc[:test_size, : ]
test_data_x = test_data.iloc[:,0:126]
test_data_y = test_data.iloc[:, 126]


test_data.shape
```

[90]: (398, 127)

[12]:
```python
from google.colab import drive
drive.mount('/content/drive')
```

```
---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
/usr/local/lib/python3.7/dist-packages/ipykernel/kernelbase.py in␣
↪_input_request(self, prompt, ident, parent, password)
    728             try:
--> 729                 ident, reply = self.session.recv(self.stdin_socket, 0)
    730             except Exception:

/usr/local/lib/python3.7/dist-packages/jupyter_client/session.py in recv(self,␣
↪socket, mode, content, copy)
    802         try:
--> 803             msg_list = socket.recv_multipart(mode, copy=copy)
    804         except zmq.ZMQError as e:

/usr/local/lib/python3.7/dist-packages/zmq/sugar/socket.py in␣
↪recv_multipart(self, flags, copy, track)
    624         """
--> 625         parts = [self.recv(flags, copy=copy, track=track)]
    626         # have first part already, only loop while more to receive

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket.Socket.recv()

zmq/backend/cython/socket.pyx in zmq.backend.cython.socket._recv_copy()

/usr/local/lib/python3.7/dist-packages/zmq/backend/cython/checkrc.pxd in zmq.
↪backend.cython.checkrc._check_rc()

KeyboardInterrupt:

During handling of the above exception, another exception occurred:

KeyboardInterrupt                         Traceback (most recent call last)
<ipython-input-12-d5df0069828e> in <module>()
      1 from google.colab import drive
----> 2 drive.mount('/content/drive')

/usr/local/lib/python3.7/dist-packages/google/colab/drive.py in␣
↪mount(mountpoint, force_remount, timeout_ms, use_metadata_server)
    111       timeout_ms=timeout_ms,
    112       use_metadata_server=use_metadata_server,
--> 113       ephemeral=ephemeral)
    
    114
```

```
    115

/usr/local/lib/python3.7/dist-packages/google/colab/drive.py in␣
↪_mount(mountpoint, force_remount, timeout_ms, use_metadata_server, ephemeral)
    290       with _output.use_tags('dfs-auth-dance'):
    291         with open(fifo, 'w') as fifo_file:
--> 292           fifo_file.write(get_code(auth_prompt) + '\n')
    293       wrote_to_fifo = True
    294     elif case == 5:

/usr/local/lib/python3.7/dist-packages/ipykernel/kernelbase.py in␣
↪raw_input(self, prompt)
    702             self._parent_ident,
    703             self._parent_header,
--> 704             password=False,
    705         )
    706

/usr/local/lib/python3.7/dist-packages/ipykernel/kernelbase.py in␣
↪_input_request(self, prompt, ident, parent, password)
    732             except KeyboardInterrupt:
    733                 # re-raise KeyboardInterrupt, to truncate traceback
--> 734                 raise KeyboardInterrupt
    735             else:
    736                 break

KeyboardInterrupt:
```

[91]:

```
remaining = data.shape[0] - test_size
train_size = int(remaining * 0.8)
data_n = pd.DataFrame(data_n, dtype='float64')
train_data = data_n.iloc[test_size : test_size + train_size, :]
train_data.shape
train_data_x = train_data.iloc[:,0:126]
train_data_y = train_data.iloc[:, 126]

train_data.head()
```

[91]:

```
        0      1        2    4     5     6   …   122   123  124   125   126
127
398  51.0  630.0  29744.0  2.0  0.01  0.39  …  0.40  0.40  0.4  0.00  0.40
0.18
399  12.0    0.7      0.7  2.0  0.03  0.00  …  0.70  0.70  0.7  0.00  0.70
0.80
400  39.0  139.0  47138.0  3.0  0.07  0.36  …  0.30  0.30  0.3  0.00  0.30
1.00
401  55.0   79.0  53000.0  3.0  0.99  0.42  …  0.38  0.57  0.5  0.24  0.25
```

```
0.40
402   22.0     0.3       0.3   3.0   0.00   0.45   …   0.30   0.30   0.3   0.00   0.30
0.05

[5 rows x 127 columns]
```

[92]:
```python
remaining_valid = data.shape[0] - test_size - train_size
valid_data = data_n.iloc[test_size+ train_size  :, :]
valid_data.shape
```

[92]:
```
(320, 127)
```

1.1.29 2.a

[93]:
```python
w, b = LinearRegression_(train_data_x, train_data_y, 0.01, 2)
print(w, b)
```

```
0      -7.460795e+05
1      -1.232640e+06
2      -1.310525e+09
4      -1.231994e+05
5      -1.058334e+03
            …
122    -2.288254e+03
123    -4.295925e+03
124    -2.941331e+03
125    -1.686733e+03
126    -2.662016e+03
Length: 126, dtype: float64 -21638.017965280207
```

[94]:
```python
# here I ran regression with sklearn since my laptop is not strong enough to␣
↪run my regression on it but I implement it as well
import numpy as np
from sklearn.linear_model import LinearRegression
reg = LinearRegression().fit(train_data_x, train_data_y)
print(reg.coef_, reg.intercept_)
```

```
[-4.88459384e-04 -1.01316758e-04 -3.95169334e-07 -1.70598373e-03
  6.42890822e-02 -5.94103134e-03  2.65511413e-01 -6.84958353e-03
 -5.33527327e-02  2.66285906e-02  1.74222422e-01 -3.02003879e-01
 -2.16012727e-01 -5.22323133e-02 -4.62932289e-01  6.48483983e-02
 -2.89986828e-01 -1.80919125e-01  1.34050797e-02 -1.74549218e-01
  2.15087021e-01  6.44209402e-02 -1.16368403e-01  3.11404951e-01
 -2.18784129e-03 -3.03580698e-01 -2.91557936e-02 -4.62209880e-02
  3.89972078e-02  3.99431714e-02  2.72645496e-02  2.89080712e-01
 -2.62850572e-01 -6.53259402e-02  3.66004975e-02  1.85071886e-01
  5.57641041e-02  3.41594743e-01 -8.36664117e-02 -5.40558523e-02
```

```
 1.15391211e-01  9.56910642e-02  4.81549731e-01  2.60415851e-01
-4.51782579e-02 -4.59523052e-01 -4.11631538e-01 -5.20290525e-02
-2.85458797e-01  2.73102787e-02  1.57932849e-02  4.51551548e-02
-1.91378981e-01 -2.06380679e-01  1.49248114e-01 -7.90682889e-02
 5.59584142e-02 -2.23227449e-02  6.40755601e-02 -8.53342135e-02
-1.62733717e-02 -6.67689185e-02 -2.60590928e-01  3.68705237e-01
-1.67556450e-01 -2.85406968e-01  2.79111026e-01 -3.74910463e-01
 8.24898134e-01 -7.37712981e-02 -2.88962157e-01 -8.41471225e-01
 2.34676303e-01  8.74450051e-02  5.29880045e-02  2.23245277e-01
-3.75056593e-02  6.48468238e-01  7.72399263e-02 -8.05476419e-02
-2.69679146e-02  2.80837451e-02 -4.52972541e-02 -5.09006924e-01
 4.87209540e-01 -4.81100920e-02 -1.07513039e-01 -2.00342422e-01
-1.89840696e-02  3.67640148e-01  6.58225188e-02 -5.68951079e-02
-7.51682977e-02  1.42669825e-01  1.82159198e-01  1.11276934e-01
 2.14912815e-02 -5.39521740e-02  3.13392117e-02  3.99312837e-02
 1.70533479e-02 -2.30624767e+01  1.35027418e-01  6.33969356e-01
-2.54437427e-01 -2.07761937e-01  2.40177628e-01  2.26492464e+01
-2.54395214e-02 -1.31899601e-01 -3.42586793e-01 -2.92825402e-01
 9.68289577e-02  2.01959255e-01 -6.34545792e-03 -2.02217852e-02
-2.51151032e-02  4.70571120e-02 -1.38738471e-02 -2.78264838e-02
 1.23164121e-01  2.72001891e-01 -7.12029217e-02  1.55913222e-02
-4.61172392e-02 -3.84260048e-02] 0.7202570773031803
```

[95]:
```python
from random import randrange

def cross_validation(x, n):
    k_fold = []
    fold_size = int(x.shape[0] / n)
    for i in range(n):
        fold = []
        while len(fold) < fold_size:
            index = randrange(x.shape[0])
            fold.append(x.iloc[index,:])
        k_fold.append(fold)
    return k_fold
```

[96]:
```python
def rmse(y, x, w, b):
    #w = w.values.reshape((126,1))
    y_hat = x.dot(w) + b
    error = 0
    for i in range(len(y)):
        error = error + (y_hat[i] - y[i]) ** 2 / int(len(y))
    return np.sqrt(error)


k_fold = cross_validation(valid_data, 5)
fold_rmse= []
```

```
for d in (k_fold):
    d = pd.DataFrame(d)
    data_x = d.iloc[:,:126]
    data_y = d.iloc[:,-1]
    y_hat = data_x.dot(reg.coef_) + reg.intercept_
    error_rmse = np.sum((y_hat - data_y)**2) / 1994
    fold_rmse.append(error_rmse)

print(f"5-fold cross-validation average RMSE is :{np.average(fold_rmse)} ")
```

```
5-fold cross-validation average RMSE is :0.0005378357434981447
```

1.1.30 2.b

[97]:
```
#print(test_data_x.head())
test_x =test_data_x.to_numpy(dtype = 'float64')
y_hat = np.dot(test_x, reg.coef_) + reg.intercept_
#y_hat = test_x.dot(reg.coef_) + reg.intercept_
error_rmse = np.sum((y_hat - test_data_y)**2) / 1994
print(f"test RMSE is :{error_rmse} ")
```

```
test RMSE is :0.003976107921357373
```

1.1.31 3

[98]:
```
def rigid(x, y, learning_rate, landa, iteration):
    w_ = []
    b_ = []
    m, n = x.shape
    w = np.zeros(n)
    b = 0

    for i in range(iteration):
        y_hat = np.dot(x, w) + b
        gradient_w = 2 *( np.dot(np.transpose(x), (y_hat - y))) + ( 2 * landa *␣
↪w )   / m
        gradient_b = 2 * np.sum((y_hat - y)) / m

        w = w - learning_rate * gradient_w
        b = b - learning_rate * gradient_b
        w_.append(w)
        b_.append(b)

    return w, b
```

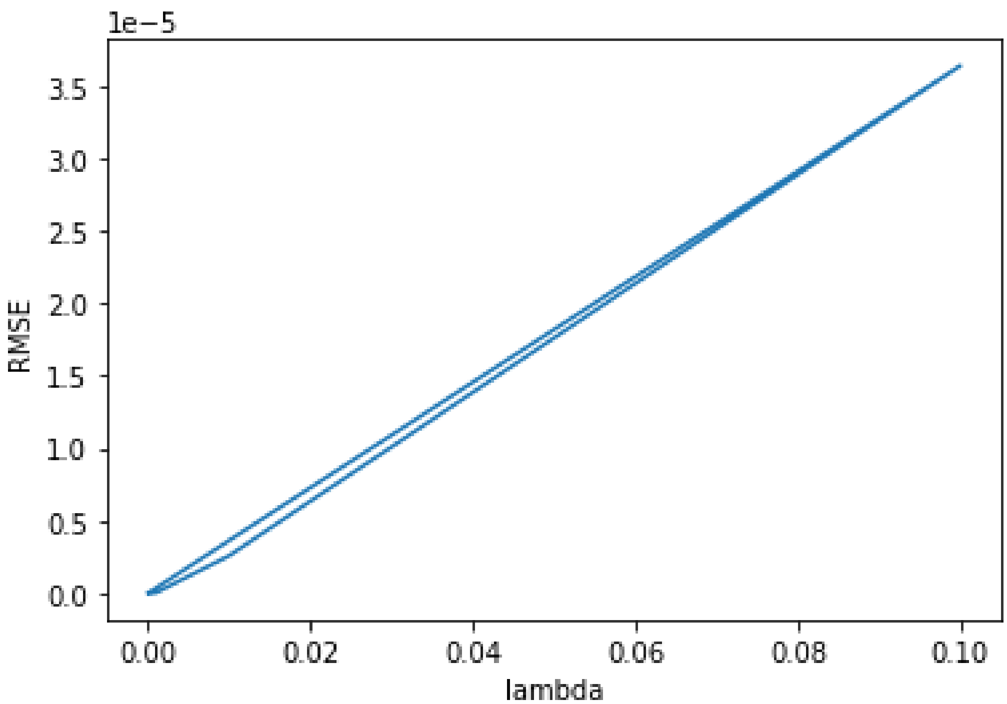
1.1.32 3.a

[110]:
```python
from sklearn import linear_model

landa = [ 0,0.1, 0.01, 0.001, 0.0001]
ave = []
for l in landa:

    kfold = cross_validation(valid_data, 5)
    fold_rmse= []
    for d in (k_fold):
        d = pd.DataFrame(d)
        data_x = d.iloc[:,:126]
        data_y = d.iloc[:,-1]
        reg = linear_model.Ridge(alpha=l)
        reg.fit(data_x, data_y)
        #w, b =rigid(data_x, data_y, 0.5, l, 10)
        y_hat = data_x.dot(reg.coef_) + reg.intercept_
        error_rmse = np.sum((y_hat - data_y)**2) / 1994
        fold_rmse.append(error_rmse)
    ave.append(np.average(fold_rmse))


#print(f"5-fold cross-validation average RMSE is :{np.average(fold_rmse)} ")
```

```
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_ridge.py:190:
UserWarning: Singular matrix in solving dual problem. Using least-squares
solution instead.
  warnings.warn("Singular matrix in solving dual problem. Using "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_ridge.py:190:
UserWarning: Singular matrix in solving dual problem. Using least-squares
solution instead.
  warnings.warn("Singular matrix in solving dual problem. Using "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_ridge.py:190:
UserWarning: Singular matrix in solving dual problem. Using least-squares
solution instead.
  warnings.warn("Singular matrix in solving dual problem. Using "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_ridge.py:190:
UserWarning: Singular matrix in solving dual problem. Using least-squares
solution instead.
  warnings.warn("Singular matrix in solving dual problem. Using "
/usr/local/lib/python3.7/dist-packages/sklearn/linear_model/_ridge.py:190:
UserWarning: Singular matrix in solving dual problem. Using least-squares
solution instead.
  warnings.warn("Singular matrix in solving dual problem. Using "
```

[111]:
```
import matplotlib.pyplot as plt

print(ave)
plt.plot(landa, ave)
plt.xlabel("lambda")
plt.ylabel("RMSE")
plt.show()
```

```
[1.5139859888902382e-12, 3.640739382861271e-05, 2.591611817328552e-06,
4.413862662326075e-08, 4.766207503574767e-10]
```



3.b

lambda = 0.1 is the best fit

3.c

[115]:
```
test_data_x = test_data_x.to_numpy(dtype = 'float64')

y_hat = test_data_x.dot(reg.coef_) + reg.intercept_
error_rmse = np.sum((y_hat - test_data_y)**2) / 1994
print(f"test RMSE is :{error_rmse} ")
```

```
test RMSE is :0.022755918237488515
```

3.d

yes we can use the information for feature selecting. we can omit(delete) the features where w=0.

3.e

[116]:
```
print(reg.coef_)
```

```
[ 5.65865967e-04 -2.82236753e-04 -6.98564691e-07  2.61420397e-02
  7.05586721e-02  1.03353318e-02  3.41345646e-01 -2.53442805e-01
  6.29959156e-02  1.25647709e-01  2.54962851e-01  1.80872925e-01
  7.50500570e-02 -1.85418966e-01  3.38351145e-02  3.63273049e-02
  4.21032481e-02  6.90213501e-03  9.16223299e-02 -3.99559656e-02
  4.02172356e-01 -1.16249431e-01 -2.06618629e-01  2.50030026e-02
  2.33358211e-02  7.72969120e-02 -1.26431757e-01  5.56742378e-02
  1.33892714e-02  4.50354255e-02  9.08033054e-02  1.16338869e-01
 -3.45404684e-02 -6.43565860e-03  3.87357918e-02 -6.93751850e-02
  6.11844671e-02 -2.78609007e-01  2.18493896e-02  2.32389364e-01
  1.12448219e-01  5.84045004e-01  9.55573575e-02 -4.88761618e-01
 -3.14016175e-02  2.57903056e-02  9.35259096e-02 -2.18632663e-01
 -1.60892065e-01 -3.23276434e-02  2.82965648e-01  1.74794321e-01
 -1.32623889e-01  1.11693320e-01  4.18695050e-01  1.05047910e-02
  1.44188769e-02  5.61661133e-02 -2.23463469e-01 -2.06509971e-01
 -9.89959325e-03  2.62407426e-02 -6.92048720e-02  3.69316464e-02
 -1.92374273e-01  3.71754530e-02 -2.85232653e-01 -2.25211889e-01
 -4.82725714e-02 -5.60886949e-02  1.84487431e-01  5.31771233e-02
  3.01929352e-02  1.38008809e-03 -1.94369824e-01  1.34835633e-01
  4.77443112e-02  5.53680607e-02  2.05534977e-02 -9.68574094e-02
 -2.41955772e-01 -1.66507710e-01  8.05435979e-02 -2.55349562e-01
 -1.23043286e-01  1.66877340e-01  1.89956565e-01  2.23676884e-01
 -1.03269471e-01 -9.35228197e-02 -8.11473760e-02  2.42905163e-01
  1.84042693e-02  1.69810211e-02 -8.38490284e-02  6.92382873e-02
  1.53280847e-01 -2.60667278e-01  2.14490692e-01 -1.70788000e-01
 -1.23655532e-02  1.79428940e-03 -6.82360587e-02  2.06187162e-03
 -6.57058025e-03  2.39539779e-02  2.35932409e-02  1.79428940e-03
 -3.92044374e-02 -5.03204106e-02  5.90991993e-02 -3.06119339e-02
 -4.79184218e-02  1.31430281e-02 -2.23638946e-02 -5.61375074e-02
 -1.23367350e-01 -4.07860322e-02 -1.38968806e-01 -7.55864917e-02
 -6.53453154e-02 -4.12127657e-02 -4.12680869e-02  1.53890176e-01
  7.63216162e-04 -4.25833873e-02]
```

3.f

by reducing the feature we will reduce the computational complexity of the model and just consider the feature that is most important for our prediction. so it will decrease the RMSE error as well.