EECE 5645 HOMEWORK 2 Solutions

Question 1 : Refer the pdf **hw2question1.pdf** attached if you don't get my 1st question below

Part A)

Given a vector x ∈ Rd and a real number y ∈ R, define the function f : Rd → R as

$$f(\beta; x, y) = \left(y - x^{\top}\beta\right)^2 = \left(y - \sum_{k=1}^{d} \beta_k x_k\right)^2.$$

To find the partial derivates of f with respect to β_k , so expand the square term in the numerator

numerator

so expand $\left(y - \sum_{k=1}^{d} B_k x_k\right)^2$

$$= y^2 - \left(\sum_{k=1}^{d} B_k x_k\right)^2$$

$$= (y - x^T B)^2$$

It can be written as

$$= (y - b_1 x_1 - b2 x_2 - \dots - bd xd)^2$$

$$= y^2 - 2y\, x^T B + (x^T B)^2$$

$$= y^2 - 2y\, x^T B + (x^T B)(B^T x^T)$$

$$f(B;, x y) = y^2 - 2y x^T B_K + x^{T^2} (B_K B_K^T)$$

$$= y^2 - 2 x^T y B_K + (x_K^T)^2 (B_K)^2,$$

Hence β^T is a scalar quantity and β_k β_k^T is a d*d matrix so it can be written as $(\beta_k)^2$
Hence by taking partial derivatives

$$\frac{df}{dB_k} = -2yx_k + 2\,\cancel{B_k}\,\beta_k x_k^2$$

also $B_k^T B_k$ $d \times d$ can be expandd as

$$= (b_1^2\ b_1b_2 \ldots b_1bd,\ b_2^2\ b_1b_2 \ldots b_2bd \ldots\ b_1bd,\ b_2bd \ldots bd^2)$$

hence the

$$\frac{df}{dB_k} = -2yx_k + 2B_K\,\cancel{B_k}^T (x_K)^2 \quad \text{for } K = 1 \ldots d$$

where $\beta_k\ \beta_k$^T k denotes the K^{th} component of the vector βk βk^Tx . Therefore the partial derivatives of f with respect to β_k is -2 times the k^{th} component of x times y plus 2 times k^{th} component of vector $\beta_k X_k$.

Ans B)

Question 2 : The gradient of f as a d-dimensional column vector with components given by the partial derivatives with respect to each element of β .

so from $\dfrac{df}{dB_k} = -2yx_k + 2B_K\,\cancel{B_k}\,(x_K)^2$

$\nabla f(B, x, y) = [df/dB_1, df/B_2 \ldots df/B_d]^T$

using dot product of B and x as

$x^T B = [x_1, x_2 \ldots x_d][B_1, B_2, \ldots B_d]$

taking derivates of the expression we get w.r.t B_K

$$\frac{d}{dB_K}(x^T B) = x_K$$

$$\frac{df}{dB_K} = -2y x_K + (2x^T B) x_K$$

$$= 2(y - x^T B) x_K$$

Hence $\nabla f(B, x, y) = [-2(y - x^T B)x_1, -2(y - x^T B)x_2, \ldots -2(y - x^T B)x_d]^T$

to solve it further

$\nabla f(B, x, y) = -2(y - x^T B)(x_1, x_2, \ldots; x_d]^T$

using matrix multiplication we get

$\nabla f(B, x, y) = -2(y - x^T B)x$

using the transpose of a column vector is a row vector

$\nabla f(B, x, y) = -2(y - x^T B)x^T$

. The gradient descend can be given as.

$$= -2(y x_K - B_K B_K^T x_K)$$

$$= -2(y - x^T B_K)x^T \quad //$$

Part C)

Ans c Given:

$$F(\beta) = \frac{1}{n} \sum_{i=1}^{n} f(\beta; x_i, y_i) + \lambda \|\beta\|_2^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \beta)^2 + \lambda \beta^T \beta$$

Here $\lambda \geq 0$ and $\beta \in R^d$,

To prove

$$\nabla F(\beta) = -\frac{2}{n} \sum_{i=1}^{n} (y_i - x_i^T \beta) x_i + 2\lambda\beta$$

hence when we take partial derivatives

$$\frac{dF}{d\beta} = \frac{1}{n} \sum_{i=1}^{n} \frac{df}{d\beta} + 2\lambda$$

$$= \left(\frac{1}{n}\right) \sum_{i=1}^{n} \left[-2(x_i k)(y_i - x_i^T)\right] + 2\lambda$$

$$= \frac{-2}{n} \sum_{i=1}^{n} (x_i k(y - x_i^T)) + 2\lambda.$$

hence

$$\nabla F(\beta) = (\partial F/\partial \beta_1, \partial F/\partial \beta_2, \ldots, \partial F/\partial \beta_d)$$

d dimension of vector x_i and β . So partial derivatives can be given as

$$\partial F/\partial \beta_k = \partial/\partial \beta_k \left(1/n \times \sum (y_i - x_i^T \beta)^2 + \lambda \|\beta\|^2 \right)$$

$$= \frac{1}{n} \sum_{i=1}^{n} \partial/\partial \beta_k (y_i - x_i^T \beta)^2 + \partial/\partial \beta_k \lambda \|\beta\|^2$$

$$= \frac{1}{n} \sum_{i=1}^{n} 2 (y_i - x_i^T \beta)(-x_{ki}) + 2\lambda \beta_k$$

using chain rules $\|\beta\|^2 = \beta^T \beta$

$$= \frac{-2}{n} \sum_{i=1}^{n} (y_i - x_i^T \beta) x_{ki} + 2\lambda \beta_k$$

$$\nabla F(\beta) = (-2/n) \sum (y_i - x_i^T \beta) x_i + 2\lambda \beta$$

which is desired result.

Part D)
Given that

$$h(\gamma) = F(\beta_1 + \gamma \beta_2),$$

$$h(\gamma) = a\gamma^2 + b\gamma + c,$$

Substituting $F(\beta_1 + \gamma\beta_2)$ with its definition we get

$$h(\gamma) = \frac{1}{n} \sum (y_i - x_i^T (\beta_1 + \gamma \beta_2))^2 + \lambda (\beta_1 + \gamma \beta_2)^2$$

Expanding the squared terms we get is

$$h(\gamma) = \frac{1}{n} \sum \left(y - x^T \beta_1 - \gamma\, x_i^T \beta_2\right)^2 + \lambda \|\beta_1\|^2 + 2\lambda\gamma \beta_1^T \beta_2 + \lambda \gamma^2 \|\beta_2\|^2$$

On expanding further we get

$$h(\gamma) = \frac{1}{n} \sum \Big(y_i^2 - 2 y_i x_i^T \beta_1 + (x^T \beta_1)^2 + 2\gamma x_i^T \beta_2 y_i - (2\gamma x_i^T \beta_1) \times (x_i^T \beta_2) + (\gamma^2 x_i^T \beta_2)^2 + \lambda \|\beta_1\|^2 + 2\lambda\gamma \beta_1^T \beta_2 + \lambda\gamma^2 \|\beta_2\|^2$$

On simplifying we get

$$h(\gamma) = \frac{1}{n} \sum_{i=1}^{n} (x_i^T \beta_1)^2 - 2 y_i x_i^T \beta_1 + y_i^2 + 2\gamma x_i^T \beta_2 y_i - 2\gamma (x_i^T \beta_1)(x_i^T \beta_2) + \gamma^2 (x_i^T \beta_2)^2 + \lambda \|\beta_1\|^2 + 2\lambda\gamma \beta_1^T \beta_2 + \lambda\gamma^2 \|\beta_2\|^2$$

Collecting the terms having same power we get

$$h(\gamma) = \frac{1}{n} \sum_{i=1}^{n} (x_i^T \beta_1)^2 - 2 y_i x^T \beta_1 + y_i^2 +$$

$$\lambda \|\beta_1\|^2 + 2 x_i^T \beta_2 y_i - 2 (x_i^T \beta_1)(x_i^T \beta_2)$$

$$+ 2 \lambda \beta_1^T \beta_2) \gamma + \gamma^2 2 (x_i^T \beta_2)^2 + \lambda \gamma^2 \|\beta_2\|^2$$

Comparing it in general from we get

$$h(\gamma) = a\gamma^2 + b\gamma + c$$

we can see that

$$a = \frac{1}{n} \left[\sum_{i=1}^{n} (x_i^T \beta_2)^2 + \lambda \beta_2^T \beta_2 \right.$$

$$b = \frac{-2}{n} \sum_{i=1}^{n} (x_i^T \beta_2)(y_i - x_i^T \beta_1)$$

$$+ 2 \lambda \beta_1^T \beta_2$$

$$c = \frac{1}{n} \sum_{i=1}^{n} (y_i - x_i^T \beta_1)^2 + \lambda \beta_1^T \beta_1$$

Ans E)

<u>Ans</u> To show $a \geq 0$

Here we need to show coefficient of γ^2
a is +ve

since $(x_i^T \beta_2)^2$ and $\lambda \|\beta_2\|^2$ are both non-negative since $\lambda \geq 0$, their sum is also non negative.

$\therefore a \geq 0$ as required.

Ans f :

As per the convex function definition we need to show that 2nd derivate for all values of gamma . So taking derivative of gamma we get .

$$h'(\gamma) = 2a\gamma + b$$

2nd derivate $\quad h''(\gamma) = 2a$

Since a is a>0 and is a non negative so h''(γ) is also non negative for all value of gamma . Hence h convex function when a is not equal to 0

TO find min(h), $\frac{df}{dh} \approx 0$

$$h'(\gamma^+) = 2a\gamma + b = 0$$

$\gamma^* = -b/2a$ when u solve for γ^+

Since h is a convex function, this is a critical point and is minimum.

$\therefore$ h is minimized at $\gamma^* = -b/2a$.

Ans G)

Given that

$$\begin{aligned} \text{Minimize:} \quad & h(\gamma) \\ \text{subject to:} \quad & \gamma \in [0, 1]. \end{aligned}$$

Assume that $a \neq 0$. Find the optimal $\gamma \in [0, 1]$ is each of these cases:

(a) $\gamma^* \in [0, 1]$,

(b) $\gamma^* < 0$,

(c) $\gamma^* > 1$,

(a) $\gamma \in [0,1]$

$\frac{-b}{a} \in [0,1]$

minimum of h is achieved at

$\underline{\underline{\gamma^* = \frac{-b}{2a}}}$ since it is already in a feasible range

(b) $\frac{-b}{2a} < 0$, then minimum h's is achived at

$\underline{\underline{\gamma^* = 0}}$

since it is smallest range value within the feasible range.

Part b

If γ^* is less than 0, then the optimal solution is

$\gamma^* = 0$, since γ cannot be less than 0.

Part c

If γ^* is greater than 1, then the optimal solution is $\gamma^* = 1$, since γ cannot be greater than 1

(c) $\frac{-b}{2a} > 1$, min h is achieved at $\underline{\underline{\gamma^* = 1}}$ since ~~it~~ is the largest value in the feasible range.

when $a = 0$ we can see optimal problem such as optimization becomes linear function over a bounded interval $[0,1]$ and the min is achieved at one of the endpoints.

If $b \geq 0$ the $\min(h(\gamma))$ is at $\gamma^* = 0$
If $b \leq 0$ the $\min(h)$ is at $\gamma^* = 1$

then any value in the interval $[0,1]$ is optimal.

<u>Ans</u> given $|z_{k^*}| \geq |z_k|$

$$\min \quad z^T \beta$$
$$\text{subject to} \quad \sum_{k=1}^{d} |\beta_k| \leq K$$

Part(a) To prove if β is feasible.

$$|\beta_k| = |-K\,\text{sign}(z_{k^*})| = K \text{ and}$$

$|\beta_k| = |0| = 0$ for all $k \neq k$

so we have

$$\sum_{k=1}^{d} |\beta_k^*| = |\beta_k| + \sum_{\substack{k=1 \\ (k \neq k^*)}}^{d} |\beta_k|$$

$$= K + 0$$
$$= K$$

Hence, β^* satisfies the constraints

$\sum_{k=1}^{d} |\beta_k| \leq K$ and is feasible

Part b proving optimally of β^*:

Let β be any feasible solution satisfying the constraint

$$\sum_{k=1}^{d} |\beta_k| \leq K$$

so $z^\wedge T\beta \geq z^T \beta^*$

Let k^* be coordinate

$z^T \beta^* = z_{k^{**}}(-K \operatorname{sign}(z_{k^*})) = -K|z_{k^*}|$

$$z^T\beta = \sum_{k=1}^{d} z_k \beta_k \leq \sum_{k=1}^{d} |z_k||\beta_k|$$

As $|z_{k^*}| \geq z|k|$, we have

$$|z_{k^*}||\beta_k| \geq |z_k||\beta_k|, \text{ and}$$

$$|z_{k^*}||\beta_k| \geq \sum_{k=1}^{d} |z_k||\beta_k|$$

$\therefore$

$$|z_{k^*}||\beta_k| + \sum_{k=1}^{d} |z_k||\beta_k| \geq$$

$$\sum_{k=1}^{d} |z_k||\beta_k| \geq z^T\beta.$$

multiplying both size $-k \operatorname{sign}(z_{k^*})$

$$\beta_k - \sum_{k=1}^{d} k \operatorname{sig} z_k |z_k| \beta_k \geq -k z^T\beta$$

since $\beta_k = 0$ for all $k \neq k^*$, the sum reduce to:

$$-k |z_{k^*}| \operatorname{sign}(z_{k^*}) \beta_k \geq -k z^T\beta$$

divide both sides by $-k$ and $\times$ by z_{k^*}

$|z_{k^*}| (\text{sign } z_{k^*}) \beta_k \leq z^T \beta$

~~sign~~ sign (z_{k^*}) and sign(z_k)

are bothe either +1 or -1,

so when we multiply by $|z_{k^*}|$ to get

$z_{k^*} \beta_k \leq z_k \beta_k$

since $|z_{k^*}| \geq |z_k|$

$$|z_{k^*}| \, \|\beta_k\| \geq |z_k| \, |\beta_k| \geq \sum_{k=1}^{d} |z_k| \, |\beta_k|$$

$\longrightarrow$ 1

The

$1 > z^T \beta$

Hence we can say that

$$z^T \beta^* = -K |z_{k^*}| \leq z^T \beta$$

for any feasible β, proving the optimality of β^*

Question 2 :

After reading Help(PR.readData) it prints this :

```
Help on function readData in module ParallelRegression:

readData(input_file, spark_context)
    Read data from an input file and return rdd containing pairs of the form:
                    (x,y)
    where x is a numpy array and y is a real value. The input file should be a
    'comma separated values' (csv) file: each line of the file should contain x
    followed by y. For example, line:

    1.0,2.1,3.1,4.5

    should be converted to tuple:
```

After reading Help(PR.F) it prints this :

```
Help on function f in module ParallelRegression:

f(x, y, beta)
    Given vector x containing features, true label y,
    and parameter vector β, return the square error:

            f(β;x,y) =  (y - <x,β>)^2
```

First the argument for the whole program in spark is defined as below

```
args = parser.parse_args()

sc = SparkContext(appName='Parallel Regression')
sc.setLogLevel('warn')
```

Then there are various options that are stored within the argument. It describes the format on which the input needs to be given in terms of choosing an option for , input files , options to choose like test data or train data . Hence it describes the use cases of the positional argument.

The part of the code that helps in reading and printing the above commands is highlighted as below:

```
if __name__ == "__main__":
    parser = argparse.ArgumentParser(description = 'Parallel Regression.', formatter_class=argparse.ArgumentDefaultsHelpFormatter)
    parser.add_argument('--traindata',default=None, help='Input file containing (x,y) pairs, used to train a linear model')
    parser.add_argument('--testdata',default=None, help='Input file containing (x,y) pairs, used to test a linear model')
```

This helps us in selecting the right option from the menu and then it goes within the function as below :

```
if args.testdata is not None:
    # Read beta from args.beta, and evaluate its MSE over data
    print('Reading test data from',args.testdata)
    data = readData(args.testdata,sc)
```

Once the values goes to the respective functions then in each function whatever are the comments written within it for us to help in coding and to make us understand what code does , these comments are the printed when we call it using help(pr.readdata) or help(pr.r). This is shown below :

```
def readData(input_file,spark_context):
    """  Read data from an input file and return rdd containing pairs of the form:
                        (x,y)
         where x is a numpy array and y is a real value. The input file should be a
         'comma separated values' (csv) file: each line of the file should contain x
         followed by y. For example, line:

         1.0,2.1,3.1,4.5

         should be converted to tuple:

         (array(1.0,2.1,3.1),4.5)
    """
```

```
def f(x,y,beta):
    """ Given vector x containing features, true label y,
        and parameter vector β, return the square error:

                f(β;x,y) =  (y - <x,β>)^2
    """
```

After the values are passed inside the function it helps us to generate the data for the f and When you store it in a another rdd and take inputs you get output of the rdd as below :

The format of a line in the “data/small.test “ file sis expected to be a comma- separated list of numerical values representing the independent variables , followed by a single numerical value representing the dependent variables . For example below is the screenshot attached of the file :

```
-1.33645193442268417e+00,3.600848542167133792e-01,-1.516961555037868170e-01,1.786103773022134522e+00,-4.8123
2.106503876368362338e+00,-4.329014446182832265e-01,9.442626357254089164e-01,4.437358581154937198e+00,-9.11908
-3.245733207805587828e-01,-7.704172635517650969e-01,1.193850392195123122e+00,1.053478405625195152e-01,2.50056
5.438613463532162573e-01,-3.034140805865944701e-01,1.984081610357717684e+00,2.957851640571330654e-01,-1.65015
1.609649379872065411e-01,-1.066439888547795034e+00,-7.392464836862415734e-01,2.590971126122524823e-02,-1.7165
-2.757574043496156535e-01,-2.513288265715642122e+00,-9.158789076489609604e-01,7.604214605363743273e-02,6.9305
-6.091680874067831875e-01,-1.088233294499143938e-01,4.632135168995548113e-01,3.710857587148382319e-01,6.62916
-1.490040323249569587e+00,2.853970370486333064e+00,1.684770351679401035e+00,2.220220164909681770e+00,-4.25253
-2.382511830481278459e+00,-1.765204666428647484e-01,8.257640118836888643e-01,5.676362622383251733e+00,4.20562
-5.010526314787675517e-01,-5.673718789863278072e-02,-1.787057719772146225e+00,2.510537395117976578e-01,2.8428
```

The format of an element in the resulting RDD is a tuple of two elements , where the first element is a list of numerical values representing the independent variables , and the second element is a single numerical value representing the dependent variable . When you store it in a another rdd and take inputs you get output of the rdd as below :

```
>>> dataRDD = PR.readData("data/small.test",sc)
>>> dataRDD.take(10)
[(array([-1.33645193,  0.36008485, -0.15169616,  1.78610377, -0.4812361 ,
        0.20273462,  0.1296611 , -0.05462349,  0.02301172]), -9.256246212617082), (array([ 2.10650388, -0.43290144,  0.94426264,  4.43
735858, -0.91190857,
        1.9890929 ,  0.18740366, -0.40877266,  0.89163193]), 6.516306329260258), (array([-0.32457332, -0.77041726,  1.19385039,  0.105
34784,  0.25005689,
       -0.38749199,  0.59354276, -0.91976295,  1.42527876]), -15.3755309545626), (array([ 0.54386135, -0.30341408,  1.98408161,  0.295
78516, -0.16501519,
        1.0790653 ,  0.0920601 , -0.6019983 ,  3.93657984]), -11.25378216714633), (array([ 0.16096494, -1.06643989, -0.73924648,  0.02
590971, -0.17165943,
       -0.11899276,  1.13729404,  0.78836194,  0.54648536]), -31.206919348193306), (array([-0.2757574 , -2.51328827, -0.91587891,  0.0
7604215,  0.69305785,
        0.25256039,  6.31661791,  2.30186771,  0.83883417]), -30.607265124710928), (array([-0.60916809, -0.10882333,  0.46321352,  0.3
7108576,  0.0662917 ,
       -0.28217489,  0.01184252, -0.05040844,  0.21456676]), -16.368846462420912), (array([-1.49004032,  2.85397037,  1.68477035,  2.2
2022016, -4.25253093,
       -2.51037576,  8.14514688,  4.80828466,  2.83845114]), 40.60689774644796), (array([-2.38251183, -0.17652047,  0.82576401,  5.676
36262,  0.4205621 ,
       -1.96739253,  0.03115948, -0.14576425,  0.6818862 ]), -18.468112600246144), (array([-0.50105263, -0.05673719, -1.78705772,  0.2
5105374,  0.02842832,
        0.89540997,  0.00321911,  0.10139263,  3.19357529]), -18.676303559340088)]
>>>
```

Question 3 :

Part A) Predict function is spark

```
def predict(x,beta):
    """ Given vector x containing features and parameter vector β,
        return the predicted value:

                        y = <x,β>
    """
    return np.dot(x, beta)
    pass
```

Part B) Output after executing the function is as follows

```
>>> import ParallelRegression as PR
>>> import numpy as np
>>> x = np.array([np.cos(t) for t in range(-5,5)])
>>> beta = np.array([np.sin(t) for t in range(-5,5)])
>>> y_pred = PR.predict(x, beta)
>>> print(y_pred)
0.2720105544468494
```

Output to reload the program using the functions in the pyspark . This a library obtained from the spark documentation Whenever you change thecontents of ParallelRegression.py, you can update the function definitions in pyspark by using it as below

```
>>> import importlib
>>> import ParallelRegression as PR
>>> importlib.reload(PR)
<module 'ParallelRegression' from '/home/agnihotri.ak/EECE5645/ParallelRegression.py'>
```

Question 4 :

Answer : The code for function f

Part A)

```
def f(x,y,beta):
    """ Given vector x containing features, true label y,
        and parameter vector β, return the square error:

                f(β;x,y) =  (y - <x,β>)^2
    """
    return (y - predict(x, beta)) ** 2
```

Part B) The code for local gradient

```
def localGradient(x,y,beta):
    """ Given vector x containing features, true label y,
        and parameter vector β, return the gradient ∇f of f:

              ∇f(β;x,y) =  -2 * (y - <x,β>) * x

        with respect to parameter vector β.

        The return value is  ∇f.
    """
    return -2 * (y - np.dot(x, beta)) * x
```

Part C)

Testing local_gradient and estimated_gradient . Tested with at least 100 random values of x,y, and β (using, e.g., a for loop), and appropriate assert statements when δ(delta) value is extremely small .

First the rand vector function generates the random values we need for test as we need 100 random values

```
def rand_vector():
    x1 = np.random.rand(100)
    beta = np.random.rand(100)
    y1 = random()
    return x1, y1, beta
```

The function to test :

```
def gradient_test(rddout):
    d = 0.0001 # this is delta
    x1, y1, beta = rand_vector()
    gradvalue = PR.estimateGrad(lambda beta: PR.f(x1,y1,beta),beta, d)
    localgradvalue = PR.localGradient(x1, y1, beta)
    print("Local Gradient = ", localgradvalue)
    print("Estimated Gradient = ", gradvalue)
    for i in range(len(gradvalue)):
        sub= abs(localgradvalue[i] - gradvalue[i])
        print("Difference = ", sub)
        assert sub < d, f"Assert failed"
```

The main function to call the test

```
if __name__ == "__main__":
    sc = SparkContext(appName='test code')
    sc.setLogLevel('warn')
    input="data/small.test"
    rddout=PR.readData(input,sc)
    print(rddout.take(10))
    gradient_test(rddout)
    print("All tests passed 1!")
```

Output of the test that tell it has passed the cases well :

```
Difference =  6.018818527309122e-05
Difference =  1.8257841173152656e-05
Difference =  9.816530379680444e-06
Difference =  4.1149819615782235e-05
Difference =  6.366398400459161e-06
Difference =  5.994151408117432e-05
Difference =  8.75310360157755e-05
Difference =  1.7132199712222018e-06
Difference =  6.055665768833762e-07
Difference =  8.12521435911151e-06
Difference =  3.9173080024212936e-07
Difference =  1.9874672865682896e-05
All tests passed 1!
```

Question 5 :

Part A) Function F code

```
def F(data,beta,lam = 0):
    """  Compute the regularized mean square error:

            F(β) = 1/n Σ_{(x,y) in data}    f(β;x,y)  + λ ||β ||_2^2
                 = 1/n Σ_{(x,y) in data} (y- <x,β>)^2 + λ ||β ||_2^2

        where n is the number of (x,y) pairs in RDD data.

        Inputs are:
           - data: an RDD containing pairs of the form (x,y)
           - beta: vector β
           - lam:  the regularization parameter λ

        The return value is F(β).
    """

    n = data.count()
    mean_squared_error = data.map(
        lambda element: f(element[0], element[1], beta) / n).sum()
    regulization_term = lam * np.dot(beta, beta)
    return mean_squared_error + regulization_term
```

Part B) The gradient code

```
def gradient(data,beta,lam = 0):
    """ Compute the gradient  ∇F of the regularized mean square error
                F(β) = 1/n Σ_{(x,y) in data} f(β;x,y) + λ ||β ||_2^2
                     = 1/n Σ_{(x,y) in data} (y- <x,β>)^2 + λ ||β ||_2^2

        where n is the number of (x,y) pairs in data.

        Inputs are:
            - data: an RDD containing pairs of the form (x,y)
            - beta: vector β
            - lam:  the regularization parameter λ

        The return value is an array containing ∇F.
    """
    n = data.count()
    mse_gradient = data.map(
        lambda element: localGradient(element[0], element[1], beta) / n
        ).sum()
    weights_gradient = 2 * lam * beta
    return mse_gradient + weights_gradient
```

Part C)

Testing gradient actual value and gradient calculated value . Tested with multiple random values of β and λ (using, e.g., a for loop), and appropriate assert statements. Using ‘data/small.test ‘as a dataset. This defined in the function below

```
def gradient_file_test(rddout):
    d = 0.0001 # this is delta
    beta = np.random.rand(9)
    lam = random() * 5.0 + 0.1
    gradientactualvalue = PR.gradient(rddout, beta, lam)
    gradientcalculatedvalue = PR.estimateGrad(lambda beta: PR.F(rddout, beta, lam), beta, 0.0000001)
    print("Actual Gradient = ", gradientactualvalue)
    print("Estimated Gradient = ", gradientcalculatedvalue)
    for i in range(len(gradientactualvalue)):
       sub = abs(gradientcalculatedvalue[i] - gradientactualvalue[i])
       print("Difference = ", sub)
       assert sub < d, f"Assert failed"
```

The main function to test the gradient_file_test function()

```
if __name__ == "__main__":
    sc = SparkContext(appName='test code')
    sc.setLogLevel('warn')
    input="data/small.test"
    rddout=PR.readData(input,sc)
    print(rddout.take(10))
    #gradient_test(rddout)
    #print("All tests passed 1!")
    gradient_file_test(rddout)
    print("All tests passed 2!")
```

Output : It's the output for the above test it says that all test cases are passed . Also an rdd of 10 is printed to verify well

```
Actual Gradient =  [ -4.58383292 -45.33856758 -14.07694946  13.27115949  32.86146581
  15.93930387  -2.7398814  -17.97736514  21.83277825]
Estimated Gradient =  [ -4.5838334  -45.33856782 -14.07694981  13.2711591   32.86146523
  15.93930278  -2.73988121 -17.97736445  21.83277843]
Difference =  4.848518111444378e-07
Difference =  2.3960751605045516e-07
Difference =  3.4630211054320625e-07
Difference =  3.8445504202400116e-07
Difference =  5.77279145375087e-07
Difference =  1.0810062214261507e-06
Difference =  1.8549331182171613e-07
Difference =  6.90951150517094e-07
Difference =  1.843307373405878e-07
All tests passed 2!
```

Question 6 :

Using the equations below

$a=\lambda \beta_2^T \beta_2 + 1/n \sum(X_i^T \beta_2)^2$

$b= 2\lambda \beta_1^T \beta_2 - 2/n \sum(X_i^T \beta_2)(Y_i- X_i^T \beta_1)$

$c= 1/n \sum(Y_i-X_i^T \beta_1)^2 + \lambda \beta_1^T \beta_1$

for the hcoeff functions

```python
def hcoeff(data,beta1, beta2, lam = 0):
    """ Compute the coefficients a,b,c of quadratic function h, defined as :
                    h(γ) = F(β_1 + γβ_2) = aγ^2 + bγ + c
        where F is the reqularized mean square error function.

        Inputs are:
            - data: an RDD containing pairs of the form (x,y)
            - beta1: vector β_1
            - beta2: vector β_2
            - lam: the regularization parameter λ

        The return value is a tuple containing (a,b,c).
    """
    n = data.count()
    a1 = lam * np.dot(np.transpose(beta2),beta2)
    a2 = data.map(lambda xy:pow(predict(xy[0],beta2),2))\
                .sum()
    a = a1 + (a2 / n)
    b1 = 2 * lam * np.dot(np.transpose(beta1),beta2)
    b2 = data.map(lambda xy: predict(xy[0],beta2) * (xy[1] - predict(xy[0],beta1)))\
                .sum()
    b = b1 - ((2 / n) * b2)
    c1 = lam * np.dot(np.transpose(beta1),beta1)
    c2 = data.map(lambda x:f(x[0],x[1],beta1))\
                 .sum()
    c = c1 + (c2 / n)
    return (a,b,c)
```

Testing the coeff well :

```python
def test_hcoeff(rddout):
    beta1=np.random.rand(9)
    beta2=np.random.rand(9)
    gamma = np.random.randn()
    tol = 1e-1
    lamda = 0.01
    a, b, c = PR.hcoeff(rddout,beta1,beta2,lamda)
    f1 = PR.F(rddout, beta1 + gamma*beta2, lamda)
    f2 = a*((gamma)**2)+ b*(gamma) + c
    print("f1",f1)
    print("f2",f2)
    assert abs(f1 - f2) < tol ,f"Assert failed"
```

The main function to call the test test_coeff()

```python
if __name__ == "__main__":
    sc = SparkContext(appName='test code')
    sc.setLogLevel('warn')
    input="data/small.test"
    rddout=PR.readData(input,sc)
    print(rddout.take(10))
    #gradient_test(rddout)
    #print("All tests passed 1!")
    #gradient_file_test(rddout)
    #print("All tests passed 2!")
    test_hcoeff(rddout)
    print("All the test passed 3!")
```

Output : The output tells us that the value of f1 and f2 matches with each other . Also all the test cases are passed . Hence the equation written for the l2 norm regularization are correct .

```
f1 430.8098672273337
f2 430.8098672273337
All the test passed 3!
```

Question 7 :

Part A : The function for the test

```
def test(data,beta):
    """ Compute the mean square error

            MSE(β) =  1/n Σ_{(x,y) in data} (y- <x,β>)^2

        of parameter vector β over the dataset contained in RDD data, where n is the size of RDD data.

        Inputs are:
            - data: an RDD containing pairs of the form (x,y)
            - beta: vector β

        The return value is MSE(β).
    """
    n = data.count()
    mse = data.map(lambda element: f(element[0], element[1], beta) / n).sum()
    return mse
```

Part B: The function for exactLineSearch referring the 1(g) question logic

```
def exactLineSearch(data,beta,g,lam = 0):
    """ Given  data, a vector x, and a direction g, return
                γ = argmin_{γ} F(data, β-γg)

        The solution is found by first computing the coefficients of the quadratic
        polynomial
                h(γ) = F(data, β-γg) = aγ^2 + bγ + c
        The return value is γ* = -b/(2*a)

        Inputs are:
            - data: an RDD containing pairs of the form (x,y)
            - beta: vector β
            - g: direction vector g
            - lam: the regularization parameter λ

        The return value is γ*

    """
    # Compute the coefficient of the quadratic function for gamma
    #here cof is the coefficient stores tuples cof=(a,b,c) obtained from hcoeff function
    cof = hcoeff(data, beta,-g, lam)
    if cof[0] == 0:
        gamma= -cof[2]/cof[1]
    else:
        gamma = -cof[1] / (2 *cof[0])  # this nothing but  gamma = -b/2a
    return gamma
```

Part C:

Output for small test and train

```
Reading training data from data/small.train
Gradient descent training on data from data/small.train with λ = 10.0 , ε = 0.01 , max iter =  100
k = 0   t = 1.1679270267486572  F(β_k) = 475.80205650084224     ||∇F(β_k)||_2= 48.36642222745724       γ_k = 0.034644340508364575
k = 1   t = 2.241664409637451   F(β_k) = 435.28011656165035     ||∇F(β_k)||_2= 6.369684533296867       γ_k = 0.041392870179372175
k = 2   t = 3.2633073329925537  F(β_k) = 434.4404025625204      ||∇F(β_k)||_2= 1.0347072406178048      γ_k = 0.03515640336219431
k = 3   t = 4.279298543930054   F(β_k) = 434.42158300451774     ||∇F(β_k)||_2= 0.15479527474513602     γ_k = 0.04185252366896443
k = 4   t = 5.280001163482666   F(β_k) = 434.42108157828164     ||∇F(β_k)||_2= 0.02789933943570143     γ_k = 0.035332834868740795
k = 5   t = 6.218413352966309   F(β_k) = 434.42106782721686     ||∇F(β_k)||_2= 0.004340132957768057    γ_k = 0.041958276278279694
Algorithm ran for 6 iterations. Converged: True Training time: 6.218591928482056
Saving trained β in beta_small_gd
Reading test data from data/small.test
Reading β from beta_small_gd
Computing MSE on data data/small.test
MSE is: 399.47612435238216
```

Part D :
Output for large test and train

```
Reading training data from data/large.train
Gradient descent training on data from data/large.train with λ = 10.0 , ε = 0.01 , max iter =  100
k = 0   t = 1.2748327255249023  F(β_k) = 449.5326978248175      ||∇F(β_k)||_2= 34.179638533361754      γ_k = 0.03983855923496276
k = 1   t = 2.375239372253418   F(β_k) = 426.2620454198274      ||∇F(β_k)||_2= 10.024501896219903      γ_k = 0.023775791674214469
k = 2   t = 3.4401891231536865  F(β_k) = 425.06742317950113     ||∇F(β_k)||_2= 1.8399312668497296      γ_k = 0.040062729653947055
k = 3   t = 4.49951434135437    F(β_k) = 424.9996100573415      ||∇F(β_k)||_2= 0.5704360138258547      γ_k = 0.023829714168459224
k = 4   t = 5.536103248596191   F(β_k) = 424.9957329956614      ||∇F(β_k)||_2= 0.1068094270498083      γ_k = 0.040179708024953915
k = 5   t = 6.5032265186309814  F(β_k) = 424.99550380550977     ||∇F(β_k)||_2= 0.033774449038017844    γ_k = 0.023865589344711436
k = 6   t = 7.448167562484741   F(β_k) = 424.9954901936109      ||∇F(β_k)||_2= 0.00640751543983564     γ_k = 0.04027129289545917
Algorithm ran for 7 iterations. Converged: True Training time: 7.448390245437622
Saving trained β in beta_large_gd
Reading test data from data/large.test
Reading β from beta_large_gd
Computing MSE on data data/large.test
MSE is: 438.2794383647582
```

Question 8 :

Part A : Function for solveLin()

```
def solveLin(z,K):
    """ Solve problem
          Minimize:  z^T β
          subject to:  ||β||_1 <=K
        The return value is the optimal β*.
    """
    # Find coordinate k* that maximizes |zk|
    k_star = np.argmax(np.abs(z))
    #Compute optimal beta
    beta_star = np.zeros_like(z)
    beta_star[k_star] = -K*np.sign(z[k_star])
    return beta_star
```

Part B : The function for exact line search FW

```
def exactLineSearchFW(data,beta,s):
    """ Given  data, a vector x, and a direction g, return
                γ' = argmin_{γ in [0,1]} F(data, (1-γ)β+γs)

        The solution is found by first computing the coefficients of the quadratic
        polynomial
                h(γ) = F(data, (1-γ)β + γ s) = aγ^2 + bγ + c

        Inputs are:
            - data: an RDD containing pairs of the form (x,y)
            - beta: first interpolation vector β
            - s: second interpolation vector s

        The return value is γ'

    """
    # here cof stores tuples cof=(a,b,c) obtained from hcoeff function
    cof = hcoeff(data, beta, s - beta)
    gamma = -cof[1] / (2 * cof[0])
    gamma = max(0, min(1, gamma))
    return gamma
```

Part C:

```
def train_FW(data,beta_0, K,max_iter,eps):
    """ Use the Frank-Wolfe algorithm   minimize F_0 given by

            F_0(β) = 1/n Σ_{(x,y) in data} f(β;x,y)

        Subject to:
            ||β||_1 <= K

        Inputs are:
            - data: an rdd containing pairs of the form (x,y)
            - beta_0: the starting vector β
            - K:  the bound K
            - max_iter: maximum number of iterations
            - eps: upper bound on the convergence criterion

        The function runs the Frank-Wolfe algorithm with a step-size found through
        exact line search. That is, it computes

                s_k =  argmin_{s:||s||_1<=K} s^T ∇F_0(β_k)
                β_k+1 = (1-γ_κ)β_k + γ_k s_k

        where the gain γ_k is given by

                γ_k = argmin_{γ in [0,1]} F_0((1-γ_κ)β_k + γ_κ s_k))

        and terminates after max_iter iterations or when (β_k-s_k)^T∇F(β_k)<ε.
```

```
    The function returns:
        -beta: the trained β,
        -criterion: the condition (β_k-s_k)^T ∇F(β_k)
        -k: the number of iterations performed
"""
beta = beta_0
criterion = np.inf
k = 0
while k < max_iter and criterion > eps:
    grad = gradient(data, beta)
    s = solveLin(grad, K)
    gamma = exactLineSearchFW(data, beta, s)
    beta_next = (1 - gamma) * beta + gamma * s
    criterion = np.dot(beta - s, grad)
    beta = beta_next
    k += 1
return beta, criterion, k
```

Question 9 :
Part A : The following is the table of lambda with various values :

Lambda Value	MSE
0.125	187.4876923
0.25	180.9585623
0.5	184.0926789
1.0	207.4430064
2.0	251.8238198
4.0	304.6125705
8.0	350.7712854

The vector β for the smallest mse for λ=0.25 is

7.021374726806946,5.39530071697655,3.8687673552936532,7.992597130017712,9.049287437081869,1.6732785745748397,1.026298936238485,-0.8204970313544742,-0.18889666571174063,0.04677936033184126,-1.3354586513286817,-0.08615551747222905,-0.4373865737555297,0.35018289152900106,-0.22752018461561122,-0.6747107884658557,-0.6922443487619483,-1.1130039307621327,-0.6086923938363226,-0.9539084800499947,0.2090231391648798,0.5901512726555541,-0.21683279507970948,0.5248652600462588,-0.21167886242530973,0.06495615683450194,-0.6022594810140172

Part B:

K value	MSE
1.0	408.71093
5.0	361.7065
10.0	303.660
20.0	229.5094
30.0	188.8182
40.0	180.3845
50.0	184.1069

The vector β for the smallest MSE for k=40.0 is

7.195908105527798,5.251560791266931,3.9150181134985504,9.01336708401366,10.0909500276657 25,0.8477219404948313,0.1630402404570907,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,- 0.5528261548856925,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0,0.0

Part C :

The difference between the two optimal solution is that in **b** there are more number of zeros present as compared to **a** . The reason can be due to following reasons such as convergence , usage of exact line search gradient and forward algorithm and Frank-Wolfe algorithm