EECE 5698 Assignment 2

Zifeng Wang

Question 0

The comments (docstring) part of the code in PR.readData causes this output to be printed.

A line in data/small.test is a list of floating numbers separated by comma, the first 50 values represents x and the last value represents y

An element in rdd is in the format of [numpy array of floats, float], the array represents x and the single float represents y.

Question 1

(a)The code snippet is below

```
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
```

(b)The code and the result I get in pyspark

```
>>> reload(PR)
<module 'ParallelRegression' from 'ParallelRegression.py'>
>>> import numpy as np
>>> x = np.array([np.cos(t) for t in range(5)])
>>> beta =  np.array([np.sin(t) for t in range(5)])
>>> PR.predict(x, beta)
'0.4312188399711047'
```

Question 2

(a)The gradient w.r.t β is as below

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

(b) The code snippet is below

```
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

    """
    y_hat = predict(x, beta)
    return (y - y_hat) ** 2
```

(c) The code snippet is below

```
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 - predict(x, beta)) * x
```

(d)The code and the result I get in pyspark

```
>>> y = 1.0
>>> x = np.array([np.cos(t) for t in range(5)])
>>> beta = np.array([np.sin(t) for t in range(5)])
>>> PR.localGradient(x, y, beta)
array([-1.13756232, -0.61462754,  0.47339296,  1.12617816,  0.74356035])
>>> delta = 0.000001
>>> func = lambda beta: PR.f(x, y, beta)
>>> PR.estimateGrad(func, beta, delta)
array([-1.13756132, -0.61462725,  0.47339313,  1.12617914,  0.74356078])
```

From checking the estimateGrad and the localGradient, we can conclude that the results agrees with each other so that the functions are correct.

Question 3

(a)The code snippet is below

```
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).

    """
    num_of_points = data.count()
    sum_of_errors = data.map(lambda (x, y): f(x, y, beta)) \
                        .reduce(lambda x, y: x+y)
    mse = sum_of_errors / float(num_of_points) + lam * np.sum(np.square(beta))
    return mse
```

(b) The code snippet is below

```
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.

    """
    num_of_points = data.count()
    sum_of_gradients = data.map(lambda (x, y): localGradient(x, y, beta)) \
                           .reduce(lambda x, y: x+y)
    gradient = sum_of_gradients / num_of_points + 2 * lam * beta
    return gradient
```

(c) The script is below

```python
from pyspark import SparkContext
import numpy as np
import ParallelRegression as PR

if __name__ == "__main__":
    input_file = "data/small.test"
    sc = SparkContext("local[40]", 'regression')
    data = PR.readData(input_file, sc)
    beta =  np.array([np.sin(t) for t in range(50)])
    delta = 0.000001
    func = lambda beta : PR.F(data, beta, lam=1.0)
    mse = PR.F(data, beta, lam=1.0)
    print("MSE is ", mse)
    grad = PR.gradient(data, beta, lam = 1.0)
    print("Output of gradient is ", grad)
    check_grad = PR.estimateGrad(func, beta, delta)
    print("Output of estimateGrad is ", check_grad)
    diff = grad - check_grad
    print("Difference of 2 gradients is ", diff)
```

And the results are below

```
('MSE is ', '280.2594439897265')
('Output of gradient is ', array([ 4.39945352,  2.86050932,  4.50982604, -1.85170994, -6.19540742,
       -3.84463142, -2.80309467,  3.58914049,  5.23949536,  3.20436283,
       -1.13257134, -4.00822426, -3.97652387,  1.62077657,  8.53774366,
        3.76708615, -3.56305719, -6.7752212 , -7.21234235,  2.42757438,
        2.0236413 ,  5.2434361 , -1.23645832, -1.93962279, -4.89414425,
        0.33872292,  2.67334964,  5.74150933,  3.16590298, -0.2020467 ,
       -1.27129553, -1.11336197,  2.52013787,  1.58421622, -0.24287307,
       -2.71253267, -1.66387657, -1.93504448,  6.16984068,  2.62113744,
        5.2980497 ,  2.08366788, -6.20624771,  0.16714202, -0.13615823,
        2.54079899, -0.05016347,  1.36447249, -5.48897702, -1.47100687]))
('Output of estimateGrad is ', array([ 4.39945558,  2.86051124,  4.5098281 , -1.85170762, -6.19540538,
       -3.84462913, -2.80309274,  3.5891423 ,  5.23949751,  3.20436504,
       -1.13256931, -4.00822233, -3.97652184,  1.62077885,  8.53774566,
        3.76708812, -3.56305492, -6.77521905, -7.21234028,  2.42757636,
        2.02364328,  5.24343818, -1.23645646, -1.93962086, -4.89414231,
        0.33872487,  2.67335173,  5.74151142,  3.16590513, -0.20204465,
       -1.27129363, -1.11336004,  2.5201399 ,  1.58421807, -0.24287107,
       -2.71253066, -1.66387474, -1.93504235,  6.16984278,  2.62113917,
        5.29805175,  2.08366998, -6.20624576,  0.16714409, -0.13615613,
        2.54080101, -0.05016147,  1.36447449, -5.48897503, -1.47100485]))
('Difference of 2 gradients is ', array([-2.05503552e-06, -1.92088323e-06, -2.06445047e-06, -2.32094584e-06,
       -2.03891679e-06, -2.28882426e-06, -1.93305919e-06, -1.80380712e-06,
       -2.15025418e-06, -2.21446600e-06, -2.02950293e-06, -1.92541192e-06,
       -2.03468157e-06, -2.27524644e-06, -1.99312309e-06, -1.96579176e-06,
       -2.27654767e-06, -2.15274580e-06, -2.07487460e-06, -1.98397038e-06,
       -1.97380514e-06, -2.07997366e-06, -1.85964608e-06, -1.92873854e-06,
       -1.93935536e-06, -1.94935267e-06, -2.08648140e-06, -2.08291410e-06,
       -2.14254940e-06, -2.04919474e-06, -1.90472593e-06, -1.92879148e-06,
       -2.03549063e-06, -1.84768210e-06, -2.00140529e-06, -2.01416132e-06,
       -1.83117309e-06, -2.13374861e-06, -2.09470579e-06, -1.73580866e-06,
       -2.04783892e-06, -2.10545840e-06, -1.95254035e-06, -2.07775250e-06,
       -2.09556456e-06, -2.01898490e-06, -1.99100156e-06, -1.99352576e-06,
       -1.98052779e-06, -2.01685989e-06]))
```

The difference of every elements in 2 gradients are of the order 1e-6, which means they agree and the functions are correct.

Question 4

(a)The code snippet is below

```
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).

    """
    return F(data, beta, lam = 0)
```

(b)The code snippet is below, for clarity, I also print out the “iteration 0”, which is the initial value of the required outputs.

```
def train(data,beta_0, lam,max_iter,eps):
    """ Perform gradient descent:


        to  minimize F given by

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

    where
            - data: an rdd containing pairs of the form (x,y)
            - beta_0: the starting vector β
        - lam:  is the regularization parameter λ
            - max_iter: maximum number of iterations of gradient descent
            - eps: upper bound on the l2 norm of the gradient
            - a,b: parameters used in backtracking line search


    The function performs gradient descent with a gain found through backtracking
        line search. That is it computes


            β_k+1 = β_k - γ_k ∇F(β_k)

    where the gain γ_k is given by

        γ_k = lineSearch(F,β_κ,∇F(β_k))

    and terminates after max_iter iterations or when ||∇F(β_k)||_2<ε.

    The function returns:
         -beta: the trained β,
         -gradNorm: the norm of the gradient at the trained β, and
            -k: the number of iterations performed
    """
    start_time = time()
    k = 0
    gradNorm = 0
    beta = beta_0
    fun = lambda beta_input : F(data, beta_input, lam=lam)
    while (k <= max_iter):
        grad = gradient(data, beta, lam = lam)
        gradNorm = np.sqrt(np.sum(np.square(grad)))
        func_value = fun(beta)
        time_now = time() - start_time
        print("Present iteration: {}, time elapsed: {}, function value: {}, gradient norm: {}" \
            .format(k, time_now, func_value, gradNorm))
        if gradNorm < eps:
            break
        # here we shoud print the current values
        k += 1
        gamma = lineSearch(fun, beta, grad)
        beta = beta - gamma * grad

    return beta, gradNorm, k
```

(c) –silent is for reducing the verbosity of program's runtime output,

```
verbosity_group = parser.add_mutually_exclusive_group(required=False)
verbosity_group.add_argument('--verbose', dest='verbose', action='store_true')
verbosity_group.add_argument('--silent', dest='verbose', action='store_false')
parser.set_defaults(verbose=True)
```

This part of the code causes the result.

And the printout is as below:

```
Reading training data from data/small.train
Training on data from data/small.train with λ = 0.0 , ε = 0.01 , max iter =  100
Present iteration: 0, time elapsed: 0.274766921997, function value: 220.564648376, gradient norm: 10.7326121454
Present iteration: 1, time elapsed: 0.810680150986, function value: 196.950858491, gradient norm: 5.21787375028
Present iteration: 2, time elapsed: 1.4482460022, function value: 192.202962818, gradient norm: 0.981922719761
Present iteration: 3, time elapsed: 1.85959601402, function value: 192.030079227, gradient norm: 0.695480877873
Present iteration: 4, time elapsed: 2.36791014671, function value: 191.9525186, gradient norm: 0.168629000552
Present iteration: 5, time elapsed: 2.77675104141, function value: 191.947034509, gradient norm: 0.126634425216
Present iteration: 6, time elapsed: 3.30131292343, function value: 191.944461359, gradient norm: 0.0342334207718
Present iteration: 7, time elapsed: 3.64363312721, function value: 191.944201927, gradient norm: 0.0250179753358
Present iteration: 8, time elapsed: 4.11494708061, function value: 191.944098559, gradient norm: 0.00750286425664
Algorithm ran for 8 iterations. Converged: True
Saving trained β in beta_small
Reading test data from data/small.test
Reading beta from beta_small
Computing MSE on data data/small.test
MSE is: 255.1211477462913
```

Problem 5

(a)For doing that, I write a bash script called prob4.sh as below:

```
#!/bin/bash
for i in {0..20..1}
do
spark-submit --master local[40] --driver-memory 100G ParallelRegression.py \
--train data/small.train --test data/small.test \
--beta beta_small_${i} --lam ${i} --silent
done
```

And then redirect all the out put to a text file called prob4.result by running:

```
$ bash prob4.sh > prob4.result
```

And the results are in the table below:

λ	0	1	2
MSE	255.1211477462913	233.00781602522457	229.53768265062874

λ	3	4	5
MSE	228.31209430820527	227.73080516970134	227.40651879196403
λ	6	7	8
MSE	227.2029925015651	227.06696834740447	226.972325809258
λ	9	10	11
MSE	226.90041128233875	226.84698286794222	226.80431584090084
λ	12	13	14
MSE	226.77126881314143	226.74352118111076	226.72069946607627
λ	15	16	17
MSE	226.7016523704868	226.68490897129584	226.67085247567687
λ	18	19	20
MSE	226.65916447421898	226.64830964243257	226.6391722534933

We can see from the table that when $\lambda = 20$, we get least MSE on test data.

The corresponding β is below:

-0.017402507353,0.00797474217258,-0.0274240947991,0.0159458185151,-0.022518811717,-0.0239316217949,0.0364235799041,-0.0227455624674,-0.0311777331392,-0.0135229518377,0.0064162340321,0.0339924770738,0.0196718740877,0.0423553130949,-0.0435532651886,0.0156584152513,-0.0285058292549,-0.0158214447749,0.0726666691996,0.0583111537736,0.0874434829294,-0.0365586513321,-0.00343575922875,0.0424288008633,-0.0216981748667,0.00463211961379,-0.052438977233,-0.0151045998635,0.044389229654,0.0479043633361,0.0251447153654,-0.0053569524134,0.0606309128847,-0.0122600504909,-0.00918442247061,-0.0361993979921,0.0492520143848,-0.0159157038413,0.0602865151194,-0.0761269591162,-0.0281943631548,0.00396622694321,-0.0173751453971,0.00250281759287,0.0290921683854,0.022211985252,0.0116091612846,0.0358589509738,-0.0521946977874,0.0304654818825

(b)Here we also have a bash script called prob4b.sh as below:

```
#!/bin/bash
for i in {0..10..1}
do
spark-submit --master local[40] --driver-memory 100G ParallelRegression.py \
--train data/big.train --test data/big.test \
--beta beta_big_${i} --lam ${i} --silent
done
```

By doing similar redirecting trick:

```
$ bash prob4b.sh > prob4b.result
```

And the results are in the table below:

λ	0	1	2
MSE	4151.493711621748	4000.0993020234955	3977.838485799316
λ	3	4	5
MSE	3971.3275160962307	3968.868207617222	3967.832654269264
λ	6	7	8
MSE	3967.3861594271334	3967.213968075196	3967.176583535701
λ	9	10	
MSE	3967.207135497181	3967.2723069311182	