EECE 5698
Assignment 2: Parallel Regression

Preparation. Create a folder on discovery named after your username under the directory `/scratch`. You have done this alredy for HW1; if not you can do so by logging in to discovery and typing

```
mkdir /scratch/$USER
```

Copy the directory

```
/scratch/EECE5698/Assignment2
```

to the folder you just created, by typing:

```
cp -r /scratch/EECE5698/Assignment2 /scratch/$USER/
```

Make the contents of this directory private, by typing:

```
chmod -R go-rx /scratch/$USER/Assignment2
```

The directory contains (a) a python file called `ParallelRegression.py`, (b) a directory called `data` containing 4 files: two training datasets and two test datasets. In this assignment, you are asked to modify the provided code `ParallelRegression.py` and use it to train and test a linear model over the data using ridge regression. You must:

1. Provide a report, in `pdf` format, outlining the answers of the questions below. The report should be type-written in a word processor of your choice (e.g., MS Word, Latex, etc.).
2. Provide the final code in `ParallelRegression.py` you wrote, that implements the full functionality described below.

The report along with your final code should be uploaded to Blackboard. Executions of the code to generate the requested output can be run on "local" mode, on single compute node which you have reserved on the Discovery cluster. Use `sr5698` to reserve a node.

Question 0: Go to the directory that contains `ParallelRegression.py` and start the `pyspark` interpreter:

```
pyspark --master local[40]
```

Within the interpreter, type:

```
import ParallelRegression as PR
help(PR.readData)
help(PR.f)
```

Which part of the code in `PR.readData` causes this output to be printed? Try to use the function `PR.readData` to read file `data/small.test` and load its contents into an RDD. Describe what is the format of a line in `data/small.test`, and what is the format of an element of the resulting RDD.

Question 1: In linear regression, the predicted score of a vector of $x \in \mathbb{R}^d$ under parameter vector $\beta \in \mathbb{R}^d$ is given by:

$$\hat{y} = \beta^\top x$$

1(a) Modify the code in `ParallelRegression.py` so that function `predict`

- receives as input an x and a β represented as numpy arrays, and
- returns the predicted score $\hat{y}$, i.e., their inner product $\beta^\top x$.

Add the corresponding code snippet in your report.

1(b) Import your code within `pyspark` to compute the predicted score under the following two arrays

```
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)])
```

Add the code that you entered in `pyspark` to execute this, as well as the result, in your report. **Tip.** Whenever you change the contents of `ParallelRegression.py`, you can update the function definitions in `pyspark` by typing, e.g., `reload(PR)`.

Question 2: The given a feature vector $x \in \mathbb{R}^d$, a true value $y \in \mathbb{R}^d$, and a parameter vector $\beta \in \mathbb{R}^d$, the *square error* of a prediction is given by

$$f(\beta; x, y) = (y - \beta^\top x)^2.$$

2(a) Treating $f(\beta) = f(\beta; x, y)$ as a function of variable β only, derive the formula of its gradient $\nabla f(\beta)$ w.r.t. β.

2(b) Modify the program `ParallelRegression.py` so that the function `f`:

- receives as input an x and a β represented as numpy arrays, and a float y, and
- returns the square error $f(\beta; x, y)$.

Include the code snippet you wrote in the report.

2(c) Similarly, modify `ParallelRegression.py` so that the function `localGradient`

- receives as input an x a β represented as numpy arrays, and a float y, and
- returns the gradient $\nabla f(\beta; x, y)$ w.r.t. β.

Include the code snippet you wrote in the report.

2(d) Use the `python` or `pyspark` interpreter, or write a `python` script, that verifies that your computation of the gradient is correct–or, at least, is consistent with your implementation of f. The script should import `f` and `localGradient` from `ParallelRegression.py`, *as well as* function `estimateGrad`. Using these three functions you should confirm the correctness of the gradient computed by `localGradient` for the following inputs:

```
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)])
```

Correctess can be tested by confirming that `localGradient` agrees with the estimate produced by `estimateGrad` when δ is small. In writing this verification code/script, you should accomplish this **without modifying the definitions/arguments** of any of these three functions, even though `estimateGrad` assumes that `fun` has a single argument! Include the verification code/script that you wrote, as well as its output, in your report.

Question 3: Given a dataset $\mathcal{D}$ of pairs $(x_i, y_i) \in \mathbb{R}^d \times \mathbb{R}$, $i = 1, \ldots n$, *ridge regression* attempts to find a β that fits the dataset by minimizing the following ℓ_2*-regularized mean square error*:

$$\begin{aligned} F(b) &= \frac{1}{n}\sum_{i=1}^{n} f(\beta; x_i, y_i) + \lambda \sum_{j=1}^{d} \beta_j^2 \\ &= \frac{1}{n}\sum_{i=1}^{n} (y_i - \beta^\top x_i)^2 + \lambda \sum_{j=1}^{d} \beta_j^2 \\ &= \frac{1}{n}\|y - X\beta\|_2^2 + \lambda\|\beta\|_2^2 \end{aligned}$$

where $X \in \mathbb{R}^{n\times d}$ is a matrix whose rows comprise the feature vectors in the dataset $\mathcal{D}$, and $y \in \mathbb{R}^n$ is the vector of target/ground truth values. Parameter λ is called the *regularization parameter*. When $\lambda = 0$, ridge regression corresponds precisely to least squares estimation.

3(a) Modify the program `ParallelRegression.py` so that the function `F`:

- receives as input an RDD comprising (x, y) pairs, a β represented as a numpy array, and a float λ, and
- returns $F(\beta)$, the regularized MSE.

The function should make use of `f`, the per-data point square error function f. Include the code snippet you wrote in the report.

3(b) Modify the program `ParallelRegression.py` so that the function `gradient`:

- receives as input an RDD comprising (x, y) pairs, a β represented as a numpy array, and a float λ, and
- returns the gradient $\nabla F(\beta)$ of the regularized MSE F.

The function should make use of `localGradient`, the per-data point square error function F. Include the code snippet you wrote in the report.

3(c) Again, either by using the `pyspark` interpreter, or by writing a script, use the above definitions as well as `estimateGrad` to test whether your gradient estimate is correct. Use `data/small.test` as a dataset, and, e.g., $\lambda = 1.0$ and

```
beta =  np.array([np.sin(t) for t in range(50)])
```

as a β value for your tests. Include your code and verification results in your report. Again, you should perform this verification without altering the argument list of any of the aforementioned functions, including `estimateGrad`.

Question 4: You are asked to solve the problem $\min_{\beta \in \mathbb{R}^d} F(\beta)$ through gradient descent. In particular, starting from $\beta_0 = 0 \in \mathbb{R}^d$, you are asked to perform the following iterations: For $k = 0, 1, 2, \ldots$,

$$\beta_{k+1} = \beta_k - \gamma_k \nabla F(\beta_k).$$

In the above iterative process, the gain γ_k is determined through backtracking line search (see "Convex Optimization", Boyd and Vandenberghe, page 464). Code for performing backtracking line search has been provided in `ParallelRegression.py`: given function F, a current value $\beta_k \in \mathbb{R}^d$, and gradient $\nabla F(\beta_k)$, the function `lineSearch` returns the gain γ_k to be used a gradient descent step when called as:

$$\gamma_k = \texttt{lineSearch}(F, \beta_k, \nabla F(\beta_k)).$$

Note that the code anticipates that F has only a single argument. Gradient descent should be repeated until the norm $\|\nabla F(\beta_k)\|_2$ becomes less than some ε, or a maximum number of iterations is reached.

A trained parameter vector $\beta \in \mathbb{R}^d$, trained over a dataset $\mathcal{D}$.train, can be *tested* over a dataset $\mathcal{D}$.test by computing the *mean square error* of predictions over the test set; that is, if $\mathcal{D}$.test contains m datapoints (x_i, y_i), $i = 1, \ldots, m$, then

$$\text{MSE}(\beta) = \frac{1}{m} \sum_{i=1}^{m} f(\beta; x_i, y_i) = \frac{1}{m} \sum_{i=1}^{m} (y_i - \beta^\top x_i)^2.$$

4(a) Modify the program `ParallelRegression.py` so that the function `test`:

- receives as input an RDD comprising (x, y) pairs, a β represented as a numpy array, and
- returns $\text{MSE}(\beta)$.

Hint: This should be extremely short.

4(b) Modify the program `ParallelRegression.py` so that the function `train` receives as input:

- an training RDD comprising (x, y) pairs,
- a $\beta_0 \in \mathbb{R}^d$, represented as a numpy array, indicating the starting value of gradient descent,
- a $\lambda \in \mathbb{R}$, indicating the regularization parameter of F
- the maximum number of iterations `max_iter`.
- the tolerance $\varepsilon > 0$.

and performs gradient descent with backtracking like search to minimize the regularized MSE F. The function should return three values:

- The computed β,
- the Euclidian norm $\|\nabla F(\beta)\|_2$ of the gradient of the regularized MSE F at the return value β,
- the number of iterations performed.

In doing so, the function should make use of `gradient`, `F`, as well as `lineSearch`.

At each iteration, print a single line containing some basic information about the progress of your execution, including:

- the present iteration number k (1,2,3, etc.),
- the time that has elapsed since the function started being executed (hint: use `time()` to get the present time)
- the present function value $F(\beta_k)$
- the present norm $\|\nabla F(\beta_k)\|_2$.

Include the code snippet you wrote in the report.

4(c) Use your code to train a model over `data/small.train` and test it over `data/small.test` with $\lambda = 0$. You can run the code by typing

```
spark-submit --master local[40] --driver-memory 100G ParallelRegression.py\
             --train data/small.train   --test data/small.test\
             --beta beta_small_0.0 --lam 0.0  --silent
```

What does `--silent` do? Which part of the code has this effect?

Add in your report the printout of your program (excluding default messages printed by spark).

Question 5. Use your code to regress values β for the two datasets in the assignment. You can extract the relevant information from the output printed from the program, but feel free to modify it so that the values you need for the assignment are stored in a file.

5(a) Train a model over `data/small.train` and test it over `data/small.test` for the following values of λ:

$$0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0, 11.0, 12.0, 13.0, 14.0, 15.0, 16.0, 17.0, 18.0, 19.0, 20.0$$

Produce either a table or a bar plot with the resulting test MSE values for each lambda in your report. Include the vector β that attains the smallest test MSE in your report.

5(b) Train a model over `data/big.train` and test it over `data/big.test` for the following values of λ:

$$0.0, 1.0, 2.0, 3.0, 4.0, 5.0, 6.0, 7.0, 8.0, 9.0, 10.0$$

Produce either a table or a bar plot with the resulting test MSE values for each lambda in your report.