CS 395T	Scalable Machine Learning	Fall 2014

Homework 7

Lecturer: Inderjit Dhillon Date Due: Nov 24, 2014

Keywords: *PageRank, Logistic Regression, Spark, GraphLab, Hadoop*

Note:

1. You will work on Rustler and Stampede clusters for this homework.
2. Submit your code through Canvas. Submit a hard copy of your answers in class.

Problem 1: Parallel PageRank

In this problem, you will evaluate and compare the performances of the PageRank algorithm on multi-core and distributed programming platforms.

(a) (4 pt) Implement the PageRank algorithm and parallelize using it OpenMP (adapt your implementation of Eigenvector centrality in Homework 4).

(b) (2 pt) Download GraphLab from `https://github.com/graphlab-code/graphlab` and build the graph analytics toolkit that contains PageRank implementation (`toolkits/graph_analytics`).

(c) (2 pt) Download Apache Spark from `http://spark.apache.org/downloads.html` (You can download a suitable binary to match the Hadoop on Rustler, or build it yourself on Rustler from source).
Check `http://spark.apache.org/docs/latest/running-on-yarn.html` to see how you can run Spark on the Hadoop cluster set up in Rustler. Apache Spark comes with the graph analytics library called `graphx` and with an implementation of the PageRank algorithm in Scala using the `graphx` library.

(d) (2 pt) Download Apache Mahout (version 0.6) from `http://archive.apache.org/dist/mahout/` (Recent Mahout releases do not have PageRank implementation, so you have to download version 0.6), and build the package on Rustler.

(e) (10 pt) In Homework 5, you have already parallelized the PageRank algorithm on Galois. Run the three multi-core implementations (OpenMP, GraphLab and Galois) and the two distributed implementations (Apache Mahout and Spark graphx) using 1, 8 and 16 threads on the LiveJournal network `livejournal.dat` and the Friendster social network `friendster.dat`. Use $\alpha = 0.15$ (note that α is the restart probability) and number of iterations $T = 100$. Report the runtime for each case ($3 \times 5 \times 2$ in total) and comment on the performances of different platforms.

Problem 2: Distributed ALS

In this problem, you will evaluate and compare two map-reduce based implementations of the ALS method for the Netflix problem, namely, Apache Mahout (Hadoop) and MLlib (Spark). There are two ratings datasets `movielens`, with $\sim$ 10M ratings, and `netflix`, with $\sim$ 1B ratings available in the HDFS.
(Check `hadoop fs -ls /user/naga86/hw7`). Each dataset has train and test ratings.

(a) (1 pt) Download the latest Apache Mahout (version 1.0, unreleased) from `https://github.com/apache/mahout` and build the package. MLlib comes as part of Apache Spark that you used in Problem 1 (or you can build MLlib yourself on Rustler, if you downloaded the source for Apache Spark).

(b) (3 pt) Cross-validate (use 10-fold) the regularization parameter λ for ranks $k = 10, 100$ on Mahout (Hadoop) and MLlib (Spark) implementations of ALS algorithms on Rustler, for both the datasets. For the two datasets and the two implementations, report the best λ for each k.

(c) (4 pt) Train the low-rank model using the Mahout and MLlib implementations on the two datasets with $k = 10, 100$, using the respective best λ value from part (b). Report the runtime for each case ($2 \times 2 \times 2$ in total).

(d) (2 pt) Evaluate each of the trained models on the respective test ratings. Report the test RMSEs for each case.

Problem 3: Newton Method for Logistic Regression

Given a set of instance-label pairs $(\boldsymbol{x}_i, y_i)$, $i = 1, \ldots, n$, $\boldsymbol{x}_i \in \mathbf{R}^d$, $y_i \in \{+1, -1\}$, L2-regularized logistic regression estimates the model $\boldsymbol{w}$ by solving the following optimization problem:

$$\min_{\boldsymbol{w}} \quad \frac{1}{2}\boldsymbol{w}^T\boldsymbol{w} + C\sum_{i=1}^{n} \log\left(1 + \exp(-y_i \boldsymbol{w}^T \boldsymbol{x}_i)\right) := f(\boldsymbol{w})$$

Note that you have derived the gradient and Hessian of $f(\boldsymbol{w})$ in homework 2. In this problem, you will implement a logistic regression solver using C++ and OpenMP. We focus on sparse document datasets, so each $\boldsymbol{x}_i \in \mathbb{R}^d$ is a sparse vector, and d is very large.

(a) (3 pt) Write a C/C++ function to compute $f(\boldsymbol{w})$ and $\nabla f(\boldsymbol{w})$.

(b) (3 pt) An important operation in the Newton method is to compute the Hessian-vector product, i.e., $\nabla^2 f(\boldsymbol{w})\boldsymbol{v}$ where $\boldsymbol{v} \in \mathbb{R}^d$ is an arbitrary vector. Figure out an efficient way to compute the Hessian-vector product for L2-regularized logistic regression.

(Hint: the Hessian-vector product can be computed in $O(\text{nnz})$ time where nnz is number of nonzeros in the training data.)

(c) (4 pt) In the Newton method, at each iteration the Newton direction $\boldsymbol{d}$ is computed by solving the following linear system:

$$\nabla^2 f(\boldsymbol{w})\boldsymbol{d} = -\nabla f(\boldsymbol{w}).$$

Implement the Conjugate Gradient method (Algorithm 1) to compute $\boldsymbol{d}$. Use the fast Hessian-vector product you derived in prob. (b) for step 3.

(d) (5 pt) Implement the Newton method to solve the L2-regularized logistic regression problem. The detailed algorithm is described in Algorithm 2. Run the algorithm on the RCV1 dataset (available from Homework 3, `hw3-data.zip` in Canvas) for 10 outer iterations with $\lambda = 1$. Output the objective function value $f(\boldsymbol{w})$, wall time (only for the training procedure), and prediction accuracy for each iteration.

(e) (5 pt) The bottleneck of this Newton method is the Hessian-vector product used in step 3 of Algorithm 1. Use OpenMP to parallelize this part. Report the speedup using 8, 16 cores on the RCV1 dataset. (Hint: use your OpenMP implementation of sparse matrix-vector multiplication in Homework 4.)

Algorithm 1 Conjugate Gradient to Solve $A\boldsymbol{x} = \boldsymbol{b}$

- Input: A and $\boldsymbol{b}$
- $\boldsymbol{x}_0 = \underline{0}$
- $\boldsymbol{r}_0 = \boldsymbol{b} - A\boldsymbol{x}_0$
- $\boldsymbol{p}_0 = \boldsymbol{r}_0$
- $k = 0$
- For $k = 0, 1, \ldots$
  1. $\alpha_k = \frac{\boldsymbol{r}_k^T \boldsymbol{r}_k}{\boldsymbol{p}_k^T A \boldsymbol{p}_k}$
  2. $\boldsymbol{x}_{k+1} = \boldsymbol{x}_k + \alpha_k \boldsymbol{p}_k$
  3. $\boldsymbol{r}_{k+1} = \boldsymbol{r}_k - \alpha_k A \boldsymbol{p}_k$
  4. If $\frac{\|\boldsymbol{r}_{k+1}\|}{\|\boldsymbol{r}_0\|} \leq 0.01$, exit the for loop
  5. $\beta_k = \frac{\boldsymbol{r}_{k+1}^T \boldsymbol{r}_{k+1}}{\boldsymbol{r}_k^T \boldsymbol{r}_k}$
  6. $\boldsymbol{p}_{k+1} = \boldsymbol{r}_{k+1} + \beta_k \boldsymbol{p}_k$

Algorithm 2 Newton method for L2-regularized logistic regression

- Input: $\{\boldsymbol{x}_i, y_i\}_{i=1}^n$, regularization parameter λ.
- For iter $= 1, 2, \ldots, 10$
  1. Solving $\nabla^2 f(\boldsymbol{w})\boldsymbol{d} = -\nabla f(\boldsymbol{w})$ by Algorithm 1 to get $\boldsymbol{d}$.
  2. Find the maximum step size $\alpha = \max\{1, 2^{-1}, 2^{-2}, \ldots\}$ such that $\alpha\boldsymbol{d}$ satisfies the following line search condition:
  $$f(\boldsymbol{w} + \alpha\boldsymbol{d}) < f(\boldsymbol{w}) + 0.01\alpha\boldsymbol{d}^T \nabla f(\boldsymbol{w}).$$
  3. $\boldsymbol{w} \leftarrow \boldsymbol{w} + \alpha\boldsymbol{d}$.