Assignment 4: Optimization for Big Data Machine Learning Problems

Arash Vahdat

Fall 2015

Readings You are encouraged to take a look at the following readings before/while doing this assignment:

- Numerical Optimization by Jorge Genocidal and Stephen J. Wright, Sp ringer 2006, 2nd edition (available in the electronic format at the SFU library website):
  - Conjugate Gradient: Ch. 5.2
  - Limited-memory BEGS: Ch. 7.2

Provided Code and Data In this assignment, we will use the features computed in assignment 3. You can find these features and a starting sample code at the following location:

`/cs/vml2/avahdat/CMPT733_Data_Sets/Assignment4`

1 Introduction

Optimization algorithms play a key role in many machine learning frameworks. A fundamental understanding of these algorithms and their characteristics enables us to choose the appropriate optimization technique to train a learning model. This assignment implements several algorithms that are typically used for solving unconstrained optimization problems in big data. Spark provides some basic implementations of a subset of these algorithms but we will re-implement these algorithms to better understand how they work.

In the assignment 3, we trained several linear regression models for sentiment analysis. Our Spark implementation of these models had a relatively slow training time. In this assignment, we will compare the efficiency of different optimization algorithms for linear regression models.

Given a dataset of N samples, $\{(x_i, t_i)\}_{i=1}^{N}$, a linear regression model learns the parameter w such that $f(x_i) = w^T x_i$ can be as close as possible to t_i. The *ordinary linear regression* model finds such w by minimizing a squared error loss as follows:

$$E(w) = \frac{1}{2N} \sum_{i=1}^{N} \{w^T x_i - y_i\}^2 \tag{1}$$

To avoid overfitting to the training data, a special type of regression model, the *ridge regression* model, regularizes the parameter w using a L2 norm penalty. This model trains the parameter w by minimizing:

$$E(w) = \frac{1}{2N}\sum_{i=1}^{N}\{w^T x_i - y_i\}^2 + \frac{1}{2}\lambda||w||_2^2 \tag{2}$$

Goal: Given a training dataset represented by feature vectors and their corresponding target values, our goal is to minimize the objective function of the ridge regression model in Eq.2. We will use the training and test datasets provided in assignment 3. Each sample is represented using its 100 dimensional word2vec representation. You can find these datasets in the above-mentioned data directory provided for this assignment.

The sample code provided contains the following two files. The `linear_regression.py` file implements a basic ridge regression model and the `train_linear_regression.py` file implements a Spark script that loads training and test data to train a ridge regression model. As a solution to this assignment you should fill in the missing parts in these two files.

2 Computing the Objective Function and Its Gradient

Question 1: (10 marks) Implement the methods cost_grad_rdd() and cost_grad_sample() in the LinearRegressionModel() class. Note that the first method computes the value of the objective function in Eq.2 and its gradient on an RDD of labeled points using a map and reduce operation. The second method is used in the map phase to compute the cost (i.e. error) and its gradient for each data point (x_i).

Question 2: (5 marks) Extend the provided script in the `train_linear_regression.py` file to test your gradient computation. For doing so, use the finite difference technique which states that the partial derivative of a smooth function $f : \mathbb{R}^n \to \mathbb{R}$ with respect to the ith variable w_i can be approximated by

$$\frac{\partial E}{\partial w_i} \approx \frac{E(w+\varepsilon e_i) - E(w)}{\varepsilon} \tag{3}$$

for a very small positive ε and a sparse e_i that has all zeros except in the ith component.

Question 3: (5 marks) The optimization function in Eq.2 has a closed-form solution. This solution is intractable when the number of training samples and/or the length of the feature representation is very big. In our case, these quantities are small enough to solve the optimization problem exactly.

Extend the script in the `train_linear_regression.py` file to compute the optimum value for w using the closed-form solution $w^* = (\lambda NI + X^T X)^{-1} X^T t$. Here, X represents the training data matrix of size $N \times M$ where each row is a training sample (i.e. $M = 100$). I is the identity matrix of size $M \times M$, and t is a column vector of length N containing all target values for the training samples. After finding w^*, compute the globally minimum cost value for the ridge regression's optimization problem in Eq.2. Report the value you obtained for w^*.

Note that for computing the closed-form solution you are free to use numpy functions applied to the training data matrix instead of using map reduce operations.

Question 4: (10 marks) Implement the method train_gd() in the LinearRegressionModel() class. This method should train the parameters of a ridge regression model using the gradient descent algorithm shown in Algorithm 1. Run this algorithm for at most 100 iterations. Report the best step size resulting in the smallest objective value after 100 iterations and the smallest objective value obtained.

Algorithm 1 Gradient Descent Algorithm.

Input: starting point w_0, max_iteration, α a fixed step-size
Output: the position w with a minimal objective function
$k \longleftarrow 0$
while k < max_iteration **do**
  Calculate gradient $\nabla E(w_k)$ at position w_k
  $w_{k+1} = w_k - \alpha \nabla E(w_k)$
  $k = k + 1$
end while

Question 5: (10 marks) Implement the method train_cg() in the LinearRegressionModel() class. This method should train the parameters of a ridge regression model using the conjugate gradient descent algorithm shown in Algorithm 2. Run this algorithm for at most 100 iterations. Report the best step size resulting in the smallest objective value after 100 iterations and the smallest objective value obtained.

Algorithm 2 Conjugate Gradient Descent Algorithm.

Input: starting point w_0, max_iteration, α a fixed step-size
Output: the position w with a minimal objective function
$k \longleftarrow 0$
Calculate gradient $\nabla E_0 = \nabla E(w_0)$ at position w_0
$p_0 = -\nabla E_0$
while k < max_iteration **do**
  $w_{k+1} = w_k + \alpha p_k$
  Calculate gradient ∇E_{k+1} at position w_{k+1}
  $\beta \longleftarrow \frac{\nabla E_{k+1}^T \nabla E_{k+1}}{\nabla E_k^T \nabla E_k}$
  $p_{k+1} \longleftarrow -\nabla E_{k+1} + \beta p_k$
  $k = k + 1$
end while

Question 6: (10 marks) Implement the method train_lbfgs() in the LinearRegressionModel() class. This method should train the parameters of a ridge regression model using the Limited-memory BEGS (L-BEGS) algorithm shown in Algorithm 3. In this algorithm, the past memory states are represented as the last m updates from $s_k = w_{k+1} - w_k$ and $y_k = \nabla E(w_{k+1}) - \nabla E(w_k)$ where s_k represents the parameter update and y_k represents the gradient difference in each iteration. Run this algorithm for at most 100 iterations. Report the best step size resulting in the smallest objective value after 100 iterations and the smallest objective value obtained. You can use $m = 10$.

Algorithm 3 L-BEGS Algorithm.

Input: starting point w_0, max_iteration, α a fixed step-size
Output: the position w with a minimal objective function
$k \longleftarrow 0$.
while k < max_iteration **do**
  Calculate gradient $\nabla E(w_k)$ at position w_k.
  Compute direction p_k using Algorithm 4.
  $w_{k+1} = w_k + \alpha p_k$
  if k > m **then**
    Discard vector pair s_{k-m}, y_{k-m} from memory storage.
  end if
  $s_k = w_{k+1} - w_k$
  $y_k = \nabla E(w_{k+1}) - \nabla E(w_k)$
  $k = k + 1$
end while

Algorithm 4 L-BEGS two-loop recursion.

Input: $\nabla E(w_k), s_i, y_i$ where $i = k-m, \ldots, k-1$
Output: new direction p
$p = -\nabla E(w_k)$
for $i \longleftarrow k-1$ **to** $k-m$ **do**
  $a_i \longleftarrow \frac{s_i^T p}{s_i^T y_i}$
  $p = p - a_i y_i$
end for
$p = (\frac{s_{k-1}^T y_{k-1}}{y_{k-1}^T y_{k-1}}) p$
for $i \longleftarrow k-m$ **to** $k-1$ **do**
  $b = \frac{y_i^T p}{s_i^T y_i}$
  $p = p + (a_i - b) s_i$
end for

Question 7: (5 marks) Implement the method train_batch_lbfgs() in the LinearRegressionModel() class. This method should train the parameters of a ridge regression model using the minibatch version of the Limited-memory BEGS (L-BEGS) algorithm. The minibatch version is very similar to the algorithm that you have implemented in the previous question with one minor difference. Instead of working on the whole training data, iteratively sample a minibatch from the dataset (e.g. 10% of the dataset). Update the model parameters by applying the L-BEGS algorithm to the minibatch for a small number of iterations (e.g. 10 iterations). In the next iteration, sample another minibatch from the training dataset and reset the memory of the L-BEGS algorithm for the new minibatch. In this case, in order to compute a comparable value for the objective function, compute the cost (i.e. error) on the whole dataset after processing each minibatches.

Run this algorithm for at most 20 iterations. What is the best step size that results in the smallest objective value after 20 iterations? What is the smallest value obtained after 20 iterations? Note that you can gain a significant speedup by caching your minibatch. Similar to the previous question, you can use $m = 10$.

Question 8: (5 marks) Comparing the convergence rate for all the different algorithms is easier when we visualize their objective values across iterations in a single figure. Create a figure that shows the objective values in the y axis and the training iterations in the x axis. Use semilog for the y axis to compare all implemented algorithms. Include your figure in your report and describe which algorithm had the fastest convergence?

3 Submission

Your assignment should be submitted online at `https://courses.cs.sfu.ca/`. You should submit two files for this assignment:

1. **Report:** Create your report in PDF format including your answers to all above questions.
2. **Code:** Create an archive file containing all scripts used for this assignment. You should create your scripts such that your results can be reproduced again. Please include a small readme.txt file in your archive briefly explaining which script was used in which part.