Simple one-pass algorithm for penalized linear regression with cross-validation on MapReduce

Kun Yang

July 22, 2013

Abstract

In this paper, we propose a one-pass algorithm on MapReduce for penalized linear regression

$$f_\lambda(\alpha, \beta) = \|Y - \alpha\mathbf{1} - X\beta\|_2^2 + p_\lambda(\beta)$$

where α is the intercept which can be omitted depending on application; β is the coefficients and p_λ is the penalized function with penalizing parameter λ. $f_\lambda(\alpha, \beta)$ includes interesting classes such as Lasso, Ridge regression and Elastic-net. Compared to latest iterative distributed algorithms requiring multiple MapReduce jobs, our algorithm achieves huge performance improvement; moreover, our algorithm is exact compared to the approximate algorithms such as parallel stochastic gradient decent. Moreover, what our algorithm distinguishes with others is that it trains the model with cross validation to choose optimal λ instead of user specified one.

Key words: penalized linear regression, lasso, elastic-net, ridge, MapReduce

1 Introduction

Linear regression model has been a mainstay of statistics and machine learning in the past decades and remains one of the most important tools. Given the design matrix $X = (X_1, X_2, ..., X_p) = (x_{ij})_{n\times p} \in \mathbb{R}^{n\times p}$ and response Y. We fit a least square linear model by minimizing the residual sum of squares

$$\mathrm{RSS}(\alpha, \beta) = (Y - \alpha\mathbf{1} - X\beta)^T(Y - \alpha\mathbf{1} - X\beta) \tag{1}$$

There are two reasons why we are often not satisfied with 1: i) the least square estimates often have low bias but large variance, it is especially true when some of the predictors are redundant. Prediction accuracy can sometimes be improved by shrinking or setting some coefficients to zero. By doing so, we try to strike a balance between bias and variance of the model. A typical way to do shrinkage is to add some penalty terms in RSS; ii) with a large number of predictors, we often like to determine the smallest subset that exhibit the strongest effect

to enhance the interpretability of the model. Shrinkage is usually achieved by adding a penalty term in the RSS then mininizing the penalized loss function.

In this paper, we propose a one-pass algorithm on MapReduce for penalized linear regression. Compared to latest iterative distributed algorithms [1] requiring multiple MapReduce jobs, our algorithm achieves huge performance improvement; moreover, our algorithm is exact compared to the approximate algorithms such as parallel stochastic gradient decent [2]. Moreover, what our algorithm distinguishes with others is that it trains the model with cross validation to choose optimal penalty parameter instead of user specified one.

2 Simple One-Pass Algorithm

To fit the model, we need to solve the optimization

$$(\alpha, \beta) = \arg\min(Y - \alpha\mathbf{1} - X\beta)^T(Y - \alpha\mathbf{1} - X\beta) + p_\lambda(\beta) \tag{2}$$

where p_λ is some penalty function, popular choices are Lasso, Ridge and Elastic-net penalty. The columns of X are standardized to eliniminate the scaling issue, i.e., the columns are fisrt centralized then scaled to unit length

$$X = X_c D + C$$

where X_c is the standardized matrix; D is a diagonal matrix where diagonal elements are the standard deviation of each column; C is the center matrix with the form $\mathbf{1}(\bar{X}_1, \bar{X}_2, ..., \bar{X}_p)$, where $\bar{X}_i$ are the averages of X_i, $i = 1, 2, ..., p$.

We fisrt fit the model with standardized matrix X_c then transform the model back to the original scale, formally

$$\begin{aligned}
(\hat{\alpha}, \hat{\beta}) &= \arg\min(Y - \hat{\alpha}\mathbf{1} - X_c\hat{\beta})^T(Y - \hat{\alpha}\mathbf{1} - X_c\hat{\beta}) + p_\lambda(\hat{\beta}) && (3)\\
(\alpha, \beta) &= (\hat{\alpha} - CD^{-1}\hat{\beta}, D^{-1}\hat{\beta}) && (4)
\end{aligned}$$

Taking the first derivative of α and setting it to zero, we have $\hat{\alpha} = \mathbf{1}^T Y/n = \bar{Y}$ and

$$\begin{aligned}
& (Y - \hat{\alpha}\mathbf{1} - X_c\hat{\beta})^T(Y - \hat{\alpha}\mathbf{1} - X_c\hat{\beta}) && (5)\\
= \;& Y^TY - 2\hat{\alpha}Y^T\mathbf{1} + n\hat{\alpha}^2 - 2(Y - \hat{\alpha}\mathbf{1})^T X_c\hat{\beta} + \hat{\beta}^T X_c^T X_c\hat{\beta} && (6)\\
= \;& Y^TY - 2\hat{\alpha}Y^T\mathbf{1} + n\hat{\alpha}^2 - 2Y^T X_c\hat{\beta} + \hat{\beta}^T X_c^T X_c\hat{\beta} && (7)\\
= \;& Y^TY - n\bar{Y}^2 - 2(Y^TX - n\bar{Y}(\bar{X}_1, \bar{X}_2, ..., \bar{X}_p))D^{-1}\beta + && (8)\\
& \beta^T D^{-1}(X^TX - n(\bar{X}_1, \bar{X}_2, ..., \bar{X}_p)^T(\bar{X}_1, \bar{X}_2, ..., \bar{X}_p))D^{-1}\beta && (9)
\end{aligned}$$

Below (10) are the statistics we need to calculate in the algorithm and notice that they are all additive; moreover, unlike (X, Y) which usually has billions of columns and can only be stored in distributed system, these statistics can be easily loaded into memory.

$$n, Y^TY, X^TY, \bar{Y}, \{\bar{X}_i\}_{i=1}^p, X^TX \tag{10}$$

The full description of our algorithm is in Algorithm 1, where k is the number of cross validation and the rule of thumb is to set $k = 5, 10$; λs are the list of penalty parameters. In order to train the model with cross validation, we randomly distribute each sample to one of the data chunks; then calculate the statistics (10) for each chunk in the reduce phase.

Algorithm 1 Penalized Linear Regression MapReduce Algorithm

procedure PENALIZEDLR-MR(X, Y, k, λs)
MAP PHASE
  for each sample (x, y) **do**
    Generate key = random$\{0, 1, ..., k-1\}$
    Calculate statistics in (10) for (x, y): statistics = $[1, x, y, y^2, xy, x^T x]$
    Emit (key, statistics)
  end for
REDUCE PHASE
  for each (key, value list) **do**
    Aggregate the whole value list and **denote** it as chunk_statistics
    Emit (key, chunk_statistics)
  end for
CROSS VALIDATION PHASE
  $\{s_i = [n_i, X_i, Y_i, Y_i^T Y_i, X_i^T Y_i, X_i^T X_i]\}_{i=1}^k$ are the chunk_statistics from previous MapReduce job
  for each λ in λs **do**
    for $i \leftarrow 0, ..., k-1$ **do**
      train_data $= \sum_{k \neq i} s_k$
      test_data $= s_i$
      train the model (2) with **train_data** and calculate the mean squared prediction error p_i for **test_data**
    end for
    mean prediction error for λ is: pre(λ) = Average of $\{p_i\}_{i=1}^{k-1}$
  end for
  $\lambda_{\text{opt}} = \arg\min \text{pre}(\lambda)$
  data $= \sum_{i=1}^{k-1} s_i$
  train the model (2) with **data** and **transform** the model into original scale as in (3, 4)
  return $(\alpha, \beta, \lambda_{\text{opt}})$ or possible the prediction erros in cross validation for each λ
end procedure

3 Implementation

The commercial version of the implementation is available at Alpine Analytics Inc®: www.alpinedatalabs.com

4 Conclusion

In order to fully exploit the parallelism, the cross validation phase can be implemented in another MapReduce job. This feature is not in our current version because we notice that p is at the scale of $10,000$ covering most of the real word applications and it is also a physically and financially formidable task to collect billions of observations with millions of features. For the data we analyze at Alpine Analytics Inc®, they are all below the $10,000$ scale. Hence, we are confident that our version is sufficient for most applications. How to deal with more features is our future work.

References

[1] Stephen Boyd, Neal Parikh, Eric Chu, Borja Peleato, and Jonathan Eckstein. Distributed optimization and statistical learning via the alternating direction method of multipliers. *Foundations and Trends in Machine Learning*, 3(1):1–122, 2011.

[2] Martin Zinkevich, Markus Weimer, Alex Smola, and Lihong Li. Parallelized stochastic gradient descent. *Advances in Neural Information Processing Systems*, 23(23):1–9, 2010.