Parallel Data Mining

Team 2 – Flash Coders
Team Research Investigation
Presentation 2

Foundations of Parallel Computing
Oct 2014
Agenda

• Overview of topic
• Analysis of research papers
• Software design
Overview of topic
Computational Problem

Gradient descent can take a long time to converge with many training examples, and when it does it may not have found the global optimum.
Sequential Algorithm I

• Linear/logistic regression cost function:

$$\min_{\theta} J(\theta) = \frac{1}{2m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)^2$$

• Algorithm
  
  – Randomly initialize $\theta$
  
  – Repeat until convergence (for all $j=0,1,...,n$):

$$\theta_j = \theta_j - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_\theta(x^i) - y^i)x_j^i$$

What if $m$ equals 500 million training examples!

This term is $\frac{d}{d\theta_j} J(\theta)$
Sequential Algorithm II

• Predicted output is then $h_\theta(x^i)$ with trained parameters $\theta$
  
  – Linear Regression:
  
  $$ h_\theta(x^i) = \sum_{j=1}^{n} \theta^i x^i $$

  – Logistic Regression
  
  $$ z = \sum_{j=1}^{n} \theta^i x^i $$
  $$ h_\theta(x^i) = \frac{1}{1 + e^{-z}} $$
Paper analysis
Identifying Suspicious URLs: An Application of Large-Scale Online Learning
Data collection

• Created a labeled dataset of malicious Web sites with lexical and host-based features
  – Binary BOW of URL tokens, WHOIS info, and IP data
• Predict whether site is malicious from these features only, excluding Web content
  – Faster and safer not to touch malicious content
• Used benign URLs from Yahoo directory for the non-malicious class in the dataset
• This data is also used in two of the following papers
BOW grows fast

Instances: 2396130
Attributes: 3231961

Example Instance (anonymized data):
Results

• Used stochastic gradient descent to continually adapt to each new URL found
  – 98.4% classification rate

• Found size and velocity of dataset too large for computational complexity of batch processing
  – URL features changed too fast for the training time required
  – Model became stale if not retrained soon enough
  – A later paper will address this shortfall of batch training
Research Paper 2

Evaluating parallel logistic regression models
Problem

• Provide design guidelines to choose most effective combination of parameters for logistic regression on large data sets
• Compare and evaluate different distributed platforms, parallel algorithms, and sublinear approximation techniques.
Approaches

• Platforms:
  – Hadoop
  – Spark

• Algorithms:
  – Sequential optimization with stochastic gradient descent using LIBLINEAR (machine learning library)
  – Sublinear algorithms: use sampling of the dataset
Results I

• LIBLINEAR was the most precise and also the fastest
  – Ran on single machine
  – No inter-machine communication overhead and uses memory fully
  – Scalable, but limited by memory size
• LIBLINEAR was best if data set fits in memory, but couldn’t fit entire URL dataset
• Spark performs better than Hadoop
Hadoop vs SPARK

• Most machine learning algos iterate over the dataset
• Hadoop works off of disk while SPARK works off memory using RDDs
• Hadoop has non-negligible overhead for setting up tasks and passing parameters, especially for small datasets
• To get general sense of running time, we should consider both volume and the sparsity of data
• Observed drop in speedup with increased number of nodes in the cluster due to communication overhead (poor strong scaling)
Uses

• LIBLINEAR was the best logistic regression technique
• Our dataset, however, is too big to fit in memory. Has about 3 million features, 2 million instances, and is very sparse
• Authors recommend parallel gradient descent for this situation, which is exactly what we will do using the tardis cluster
• Measure speedups with different number of workers
Research Paper 3

Efficient Mini-batch Training for Stochastic Optimization
Problem

• Stochastic gradient descent for large-scale optimization problems has significant communication overhead
• To reduce the communication cost in parallelized SGD, mini-batch training needs to be employed
• But, when mini-batch size increases then rate of convergence decreases (the problem in Paper 1)
  – Uses the same URL data here from Paper 1
• This paper discusses a technique that regularizes the objective function within each mini-batch iteration to prevent the convergence rate from decreasing with increased mini-batch sizes
Batching reduces synchronization cost

- The fraction of synchronization cost as a function of minibatch size when communicating between 12 nodes.
- Note cost for every algorithm decreases with batch size.
Modified minimization equation

\[ w_t = \arg\min_{w \in \Omega} \left[ \phi I_t (w) + \frac{\gamma_t}{2} \| w - w_{t-1} \|_2^2 \right] \]

Note: \( w = \theta \)

- Prevents divergence from previous consensus found
- Limits dramatic changes of parameter \( w \) over batch iterations

General logistic regression

Batch Penalty – \( w \) has larger changes in beginning and smaller at the end
Results

- Objective value versus mini-batch size after $10^7$ examples are processed
- Note SGD diverges but efficient mini-batch training does not
Use

- Use the technique described for efficient mini-batch training without subsequent decrease in the convergence rate
Research Paper 4

Parallel Large Scale Feature Selection for Logistic Regression
Problem

• Need to try all $2^N$ feature combinations to find the best model
  – In sparse, high-dimensional data this is infeasible

• Logistic regression works well in high-dimensional data since it is likely linearly separable (this is the premise of kernels)
Theory

• Approximate the best feature to add to an existing model by building a new model in parallel for each feature NOT already in the model
  – Dth iteration, need to train N – D models

• Add the best feature found to the model and re-run full logistic regression, to make sure approximation error does not propagate
  – Have to run full regression N times, or stop short
Parallel Algorithm

**MapFunction**($\{X, \bar{y}\}, \bar{\beta}$)

- **Input**: A data block $\{X, \bar{y}\}$ and model $\bar{\beta}$.
- **Output**: Intermediate data sets $T_d \forall d$.

1. FOR each $\{\bar{x}_i, y_i, \bar{x'}_i\}$ in $\{X, \bar{y}\}$:
   2. $p_i = f(\bar{x}_i, \bar{\beta})$
   3. FOR each $x'_{id} \in \bar{x'}_i$:
      4. Store $(y_i, p_i)$ in the intermediate data $T_d$
      5. $T_d = T_d \cup (x'_{id}, y_i, p_i)$

**ReduceFunction**($T_d$)

- **Input**: An intermediate data set $T_d$.
- **Output**: Estimated coefficient $\beta'_{id}$.

1. $\beta'_{id} = 0$
2. Until convergence of $\beta'_{id}$:
3. $\frac{\partial L}{\partial \beta'_{id}} = \frac{\partial^2 L}{\partial \beta^2_{id}} = 0$
4. FOR each $(x'_{id}, y_i, p_i) \in T_d$:
   5. $a_i = \log \left( \frac{-p_i}{1-p_i} \right)$
   6. $p'_i = \frac{e^{a_i + \beta'_{id}}}{1 + e^{a_i + \beta'_{id}}}$
   7. $\frac{\partial L}{\partial \beta'_{id}} = \frac{\partial^2 L}{\partial \beta^2_{id}} + (y_i - p'_i)x'_i$
   8. $\frac{\partial^2 L}{\partial \beta^2_{id}} = \frac{\partial^2 L}{\partial \beta^2_{id}} - p'_i(1 - p'_i)x'^2_{id}$
9. $\beta'_{id} = \beta'_{id} - \frac{\partial L}{\partial \beta'_{id}} / \frac{\partial^2 L}{\partial \beta^2_{id}}$

Note:
- Authors used maximization of loglikelihood but this is easily mapped in minimization of squared error.
- Newton’s method of optimization used instead of stochastic gradient descent.

**Diagram**

- Training Data: $(\bar{x}, y, \bar{x'})$
- Intermediate Data: $(x'_{id}, y, \hat{p}_i)$
- Output: $\beta'_{id}$
Results

• Accurately found the true optimal feature to add on many iterations of feature selection

• Accuracy decreased as the number of irrelevant features in the dataset increased
  – Larger search space to find the true optimal

• Accuracy decreased as the number of base features increased
  – Each added feature has smaller marginal effect, making optimal one harder to find

• Feature rankings stable after looking at only 28% of the data
Use

• To improve our model, can use feature selection to grab only those relevant features of the 3 million inside the URL data
  – Good performance and comparable to results found in re-learning the whole model
Software design
**JOB**

**Training Phase Task**
- Calculate $\theta$ (parallelFor) on one node for a random starting initialization across nodes

**Testing Phase Task**
- Calculate test set error for each $\theta$ in Tuple Space

**Reduction Task**
- Find $\theta$ with smallest test error

References


Further References


Questions