Parallel learning of content recommendations using map-reduce

Michael Percy <mpercy@cs.stanford.edu>
Stanford University

Abstract

In this paper, machine learning within the map-reduce paradigm for ranking online advertisements is explored. Online Logistic Regression (OLR) is used to predict click-through-rates on advertising data. A parallelized stochastic gradient descent method is used in order to split the dataset across multiple machines, and the resulting predictions are evaluated.

Introduction

The content ranking and recommendation problem is a common one in industry, particularly in the realm of advertising. As the desired number of features to be used for ranking grows, and the number of items needing to be learned also increases, larger and larger numbers of training examples are relied upon in order to accurately predict appropriate contextual rankings of them. At the scale commonly seen in production systems today, parallelism is required in order to process such large amounts of data in a reasonable timeframe. Learning at such a large scale requires that the learning process be split across multiple machines and computation parallelized. A very common approach to parallelization is the map-reduce paradigm, which has an open source implementation as part of the Hadoop project. The goal of this project is to implement a parallel stochastic gradient descent (SGD) algorithm on top of map-reduce and apply that algorithm to learning a contextual content ranking.

Data & Features

The data set being used is the KDD Cup 2012 track 2 (*Predict the click-through rate of ads given the query and user information*) dataset [1]. This data set consists of online advertisement click-through data consisting of the ad id, user id, number of impressions, number of clicks, and various other features including contextual information such as query and keyword data. The data set is not huge by big data standards, but still sufficiently large as to benefit from parallelization, at 10GB uncompressed (CSV) at 150M records in the training set.

The following features from the data set were used for modeling the data: user gender, user age group (6 groups plus unknown), and ad position (cross product feature of depth x location). These features were chosen because they are very

common and widely available ad targeting features in industry, required less pre-processing to expose as feature vectors, and seemed like obvious choices for predicting who might click on a given advertisement.

Building Blocks

The software written as part of this project was built using the Apache Crunch [3] map-reduce framework. Crunch is an implementation of the FlumeJava [4] framework, which came out of Google. It provides a straightforward programming model that abstracts the MapReduce paradigm, providing a programming model that allows for defining pipelines that allow for shuffling, sorting, grouping, and parallel execution of mapper and reducer tasks, all within a hybrid functional / object-oriented paradigm in the Java language.

Another open source software project, which aims to provide building blocks for implementing scalable machine learning algorithms, is Apache Mahout [5]. Mahout provides several types of out-of-the-box machine learning algorithms, one of which is a general online logistic regression (OLR) implementation. Mahout's implementation of OLR is a single-machine in-memory implementation. As such, no features are provided to attempt to split its stochastic gradient descent method across multiple machines.

Parallel Stochastic Gradient Descent

In order for a machine learning algorithm to fully reap the benefits of running on a map-reduce cluster, a method for parallelizing the optimization step is desired. One method for parallelizing stochastic gradient descent is described in Zinkevich, et al paper from NIPS 2010 [2]. The method is, as the authors say, "strikingly simple."

First, a number of splits k is determined. The dataset is randomized and each machine is given an equal portion of the data to process. Each machine runs stochastic gradient descent (SGD) in parallel, using a fixed learning rate. Each machine returns the learned weight vector to the "master routine" which collects those weights and then simple averages them to get the resulting weight vector which is used for classification.

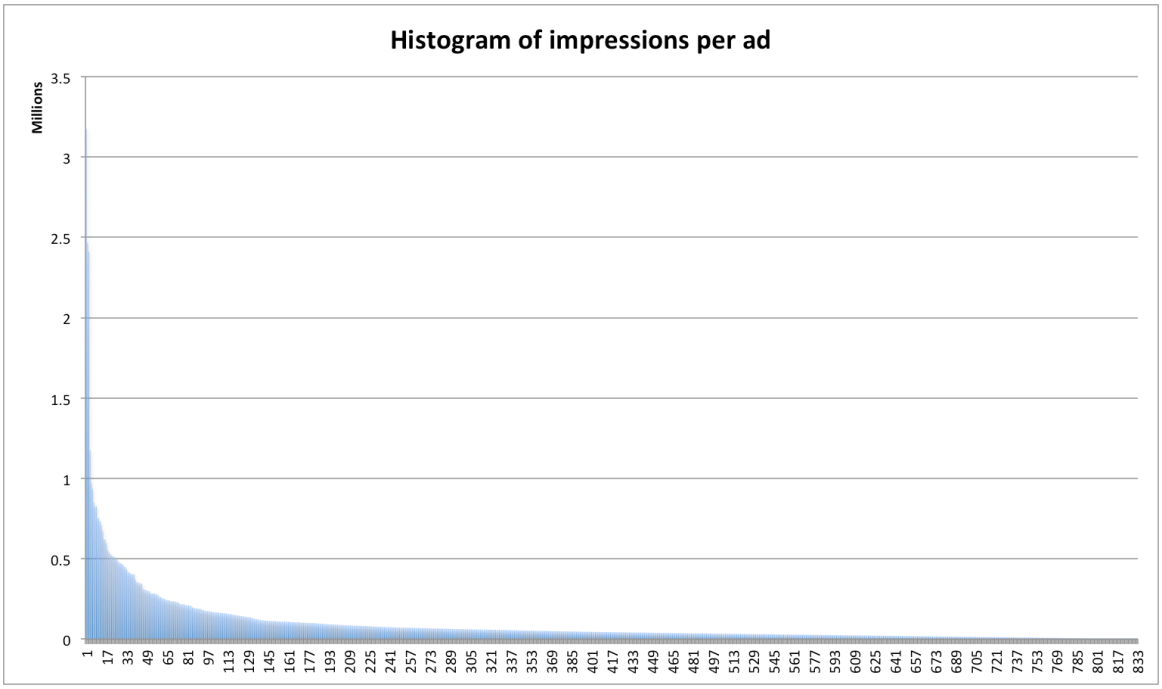
Ranking advertisements

One way of addressing the online advertising problem is to view it as a problem of predicting the expected click-through-rate (CTR) of each advertisement, based on what is known about the user viewing it, the context of the request, and the historical performance of the ad itself. While multiple ads may be shown on a single page at one time, because the inventory is so large – in the case of the KDD cup data there is an inventory of 641,707 ads – an assumption of independence seems reasonable. Therefore, one way to approach the problem is to model the problem as solving for $p(click|view, adid = i)$ where we are trying to find the probably of the user clicking on a given ad, assuming it was shown to them.

Implementation

In order to implement this CTR-prediction model, parallel stochastic gradient descent was used in tandem with online logistic regression to train models against advertisements of varying numbers of examples.

This histogram gives a sense of how widely the data set varied, in terms of impressions. There is a clear long-tail pattern, where a handful of ads get millions of impressions, a significant portion get hundreds of thousands, and many get only a handful of impressions.

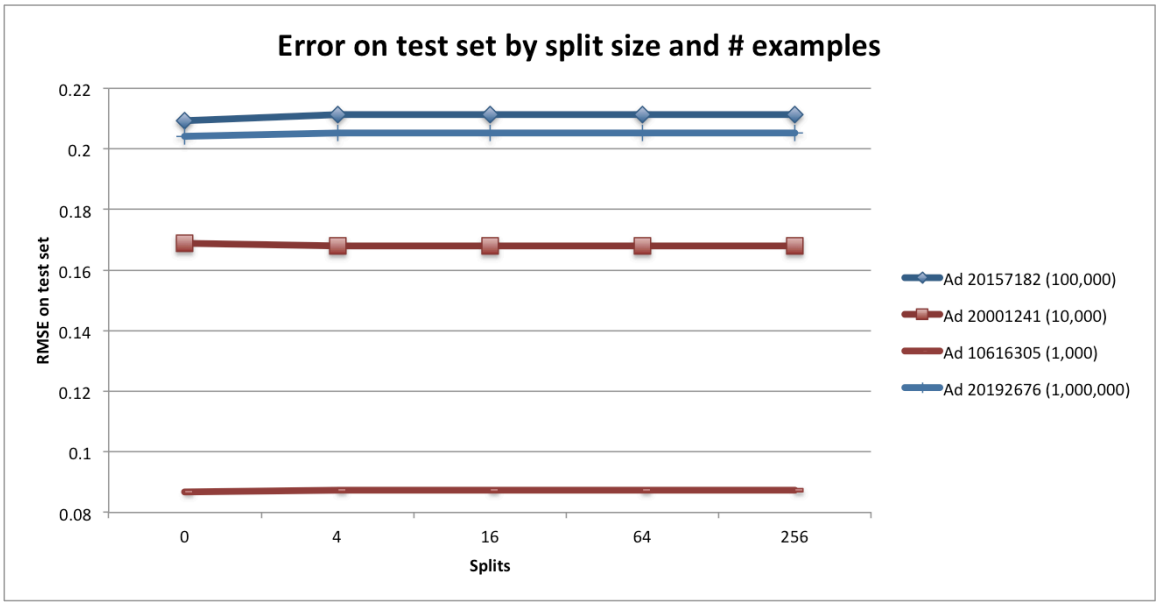


Several different data sizes were selected for modeling in order to get a sense of how the parallelization and splitting affected prediction accuracy. In each case, the total data set was split into a training set (90%) nd a test set (10%), and the root mean squared error (RMSE) was used to evaluate the effectiveness of the algorithm. In the case of the training data, the data was processed so that SGD would train against discrete click / no-click examples, however when evaluating on the test set, the logistic model prediction was compared to the session CTR of each user. In that way, we are able to treat the logistic regression output as predicting the CTR of each ad directly.

Below is a table showing how various split sizes affected the prediction error of the algorithm. Based on evaluating a wide range learning rates from 2 to 256, a learning rate of 16 was seen to be generally most effective across all data set sizes. Therefore, only the numbers corresponding to a learning rate of 16 are presented here. We show how the train and test error rate varies as the number of splits are varied.

AdID	Num. splits	Train RMSE	Test RMSE
10616305	0	0.0026785	0.0866869
10616305	4	0.015221	0.0872241
10616305	64	0.015221	0.0872241
20001241	0	0.2121002	0.2091639
20001241	4	0.2144927	0.2112489
20001241	64	0.2144927	0.2112489
20157182	4	0.1312184	0.1679808
20157182	64	0.1312184	0.1679808
20157182	0	0.1338959	0.168764
20192676	0	0.183745	0.2041563
20192676	64	0.1828694	0.205387
20192676	4	0.1828694	0.2053871

Looking at a graph of the test error data, we can see that as the data sets grow, it becomes more difficult to accurately predict the CTR. It seems likely that that has to do with ads getting more exposure having more general appeal, whereas ads that get less exposure may be more likely to have niche appeal. This seems indicative that the chosen features are not sufficient to model the data, as we can see from not only the test error going up, as larger numbers of impressions are used for training and testing, but also the training error rising as well. It seems likely that additional content- or context- based features could achieve better results on the more popular ads. Note below, in this graph, that the rough total number of impressions is listed next to each ad in parenthesis. That is the total number of impressions in the entire data set, before splitting into train & test sets.



Conclusions

As can be seen from looking at these graphs, the parallelization method proposed in [2] appears to be extremely effective. At least, it does not appear to hurt the RMSE on the test set very much. In the case of the 1,000,000 impression ad, taking the parallelization factor from 0 (regular OLR) to 64 only increased the RMSE by 0.6%. On the smaller (niche) ads, where our features modeled the data admirably, there was also a negligible difference, and in one case the error rate was even reduced by the parallelization. It seems likely that that can be attributed to some kind of regularization effect.

From a practical standpoint, it's also worth nothing that writing and debugging distributed programs is harder and takes longer than writing single-machine software in a language like Matlab. "Big data" also takes a lot longer to munge and pre-process, and requires clever tricks to handle effectively.

Future Work

While this parallelization technique appears to work well on this data set, additional investigation would be helpful in order to establish a baseline across a wider range of data sets and algorithms. In addition, modeling content and context features, in order to get more accuracy on more general-interest advertising, would help to validate that this technique scales to larger feature sets. In addition, providing this functionality out of the box in Crunch would be incredibly useful to the community at large.

References

1. 2012 KDD Cup track 2. *Predict the click-through rate of ads given the query and user information*. http://www.kddcup2012.org/c/kddcup2012-track2
2. Zinkevich, M., Weimer, M., Smola, A., & Li, L. (2010). Parallelized stochastic gradient descent. *Advances in Neural Information Processing Systems*, *23*(23), 1-9.
3. Apache Crunch. http://incubator.apache.org/crunch/
4. Chambers, C., Raniwala, A., Perry, F., Adams, S., Henry, R. R., Bradshaw, R., & Weizenbaum, N. (2010, June). FlumeJava: easy, efficient data-parallel pipelines. In *ACM Sigplan Notices* (Vol. 45, No. 6, pp. 363-375). ACM.
5. Apache Mahout. http://mahout.apache.org/