International Journal of Science and Research (IJSR)
ISSN (Online): 2319-7064
Index Copernicus Value (2015): 78.96 | Impact Factor (2015): 6.391

Machine Learning using MapReduce

Satwik Kumar Shiri[1], Satyam Thusu[2]

[1, 2]RV College of Engineering, Mysore Road, Bengaluru - 560059, India

Abstract: ***Machine learning methods often improve their accuracy by using models with more parameters trained on large numbers of data sets. Building such models on a single machine is often impractical because of expansive measure of calculation required. In this paper, we focus on developing a general technique for parallel programming of some of the machine learning algorithms. Our work is in distinct to the tradition in machine learning of designing ways to speed up a single algorithm at a time .We show that algorithms that fit the Statistical Query model can be composed in a certain "summation form," which allows them to be effectively parallelized. The central idea of this approach is to allow a future programmer or user to accelerate machine learning applications.***

Keywords: MapReduce, Machine Learning, Large Data Sets, Algorithms

1. Introduction

Machine learning (ML) is becoming increasingly popular due to a confluence of factors: an abundance of data produced and captured in digital form [1]; an abundance of compute power and convenient access to it from various devices; and advances in the ML field, making it applicable to an ever growing number of situations [2]. The acceptance and success of ML, from natural language processing to image recognition to others, comes from the increasing accuracy achieved by ML applications. This accuracy is achieved partly through advances in ML algorithms, but also through using known algorithms with larger models trained on larger datasets [2]. Building these models on a single machine is often impractical because of the large amount of computation required, or may even be impossible for very large models such as those in state-of-the-art image recognition.

2. A Taxonomy of the Machine Learning Algorithms

While ML algorithms can be classified on many dimensions, the one we take primary interest in here is that of procedural character: the data processing pattern of the algorithm. Here, we consider single-pass, iterative and query-based learning techniques, along with several example algorithms and applications.

2.1 Single-pass Learning

Many machine learning applications make only one pass through a data set and extracts relevant statistics for later use during inference. This is generally used in natural language processing, from machine translation to information extraction to spam filtering. These applications often fit perfectly into the MapReduce abstraction, encapsulating the extraction of local contributions to the map task, then combining those contributions to compute relevant statistics about the dataset as a whole. Consider the following examples, illustrating common decompositions of these statistics.

Estimating Language Model Multinomial: Extracting language models from a large corpus amounts to little more than counting n-grams, though some parameter smoothing over the statistics is also common. The map phase enumerates the n-grams in each training instance and the reduce function counts instances of n-grams.

Feature Extraction for Naive Bayes Classifiers: Estimating parameters for a naive Bayes classifier, or any fully observed Bayes net, again requires counting occurences in the training data. In this case, however, feature extraction is often computation-intensive, perhaps involving small search or optimization problems for each datum. The reduce task, is a summation of each (feature, label) environment pair.

Syntactic Translation Modeling: Generating a syntactic model for machine translation is an example of a research-level machine learning application that involves only a single pass through a preprocessed training set. Each training datum consists of a pair of sentences in two languages, an estimated alignment between the words in each, and an estimated syntactic parse tree for one sentence. The per-datum feature extraction encapsulated in the map phase for this task involves search over these coupled data structures.

2.2 Iterative Learning

The class of iterative machine learning algorithms can also be expressed within the framework of MapReduce by chaining together multiple MapReduce tasks [3]. While such algorithms vary widely in the type of operation they perform on each datum (or pair of data) in a training set, they share the common characteristic that a set of parameters is matched to the data set via iterative improvement. The update to these parameters across iterations must again decompose into per-datum contributions. In the examples below, the contribution to parameter updates from each datum (the map function) depends in a meaningful way on the output of the previous iteration. For example, the expectation computation of EM or the inference computation in an SVM or perceptron classifier can reference a large portion or all of the parameters generated by the algorithm. Hence, these parameters must remain available to the map tasks in a distributed environment. The information necessary to compute the map step of each algorithm is described below; the complications

that arise because this information is vital to the computation are investigated later in the paper.

Expectation Maximization (EM): The well-known EM algorithm maximizes the likelihood of a training set given a generative model with latent variables. The E-step of the algorithm computes posterior distributions over the latent variables given current model parameters and the observed data. The maximization step adjusts model parameters to maximize the likelihood of the data assuming that latent variables take on their expected values. Projecting onto the MapReduce framework, the map task computes posterior distributions over the latent variables of a datum using current model parameters; the maximization step is performed as a single reduction, which sums the sufficient statistics and normalizes to produce updated parameters.

We consider applications for machine translation and speech recognition. For multivariate Gaussian mixture models (e.g., for speaker identification), these parameters are simply the mean vector and a covariance matrix. For HMM-GMM models (e.g., speech recognition), parameters are also needed to specify the state transition probabilities; the models, efficiently stored in binary form, occupy tens of megabytes. For word alignment models (e.g., machine translation), these parameters include word-to-word translation probabilities; these can number in the millions, even after pruning heuristics remove the unnecessary parameters.

Discriminative Classification and Regression: When fitting model parameters via a perceptron, boosting, or support vector machine algorithm for classification or regression, the map stage of training will involve computing inference over the training example given the current model parameters. Similar to the EM case, a subset of the parameters from the previous iteration must be available for inference. However, the reduce stage typically involves summing over parameter changes. Thus, all relevant model parameters must be broadcast to each map task.

2.3 Query-based Learning with Distance Metrics

Consider distance-based machine learning applications that directly reference the training set during inference, such as the nearest-neighbor classifier. In this setting, the training data are the parameters, and a query instance must be compared to each training datum.

Multiple query instances can be processed simultaneously within a MapReduce implementation of these techniques, the query set must be sent to all map tasks. Again, we have a need for the distribution of state information. The query information that must be distributed to all map tasks need not be processed concurrently – a query set can be broken up and processed over multiple MapReduce operations. In the examples below, each query instance tends to be of a manageable size.

K-nearest Neighbors Classifier: The nearest-neighbor classifier compares each element of a query set to each element of a training set, and discovers examples with minimal distances from the queries. The map stage computes distance metrics, while the reduce stage tracks k examples for each label that have minimal distance to the query.

Similarity-based Search: Finding the most similar instances to a given query has a similar character, sifting through the training set to find examples that minimize a distance metric. Computing the distance is the map stage, while minimizing it is the reduce stage.

3. Distributed Machine Learning

Machine learning algorithms generalize from data. Machine learning algorithms train over data to create a model representation that can predict outcomes (regression or classification) for new unseen data. More formally, given a training set $\{(x_1, y_1), (x_2, y_2),\ldots..., (x_n, y_n)\}$, the goal of model training is to determine the distribution function f such that $y = f(x, w)$. The input x may consist of different features and the model consists of parameters w, representing the weights of individual features to compute y. The goal of model training is to estimate the values of model parameters w. During model testing, this model is tested using an unseen set of x_t to compare against ground truth (already known y_t), to determine the model accuracy. Thus, machine learning algorithms train to minimize the loss, which represents some function that evaluates the difference between estimated and true values for the test data.

Model training algorithms are iterative, and the algorithm starts with an initial guess of the model parameters and learns incrementally over data, and refines the model every iteration, to converge to a final acceptable value of the model parameters. Model training time can last from minutes to weeks and is often the most time consuming aspect of the learning process. Model training time also hurts model refinement process since longer training times limit the number of times the model configuration parameters can be tuned through re execution.

Machine learning algorithms can benefit from a scale-out computing platform support in multiple ways: First, these algorithms train on large amounts of data, which improves model accuracy. Second, they can train large models that have hundreds of billions of parameters or require large computation such as very large neural networks for large scale image classification or genomic applications [4]. Training with more data is done by data parallelism, which requires replicating the model over different machines with each model training over a portion of data. The replicas synchronize the model parameters after a fix number of iterations. Training large models requires the model to be split across multiple machines, and is referred to as model parallelism.

4. Tailoring Machine Learning Algorithms for MapReduce

In this section, we will briefly discuss the algorithms that can be expressed in summation form. In the following, x or x_i denotes a training vector and y or y_i denotes a training label.

Locally Weighted Linear Regression (LWLR)
LWLR [5] is solved by finding the solution of the normal equations $A\theta = b$ where $A = \sum_{i=1}^{m} w_i(x_i x_i^T)$ and $b = \sum_{i=1}^{m} w_i(x_i y_i)$. For the summation form, we divide the computation among different mappers. In this case, one set of mappers can be used to compute $\sum_{subgroup} w_i(x_i x_i^T)$ and another set to compute $\sum_{subgroup} w_i(x_i y_i)$. Two reducers respectively sum up the partial values for *A* and *b*, and the algorithm finally computes the solution $\theta = A^{-1}b$. Note if $w_i = 1$, the algorithm reduces to the case of ordinary least squares.

Naive Bayes (NB): In NB [6], we have to estimate $P(x_j = j \mid y = 1)$, $P(x_{jc} = k \mid y = 0)$, and P(y) from the training data. In order to estimate, we need to sum over $x_j = k$ for each *y* label in the training set to calculate $P(x \mid y)$. We can specify different set of mappers to calculate the following: $\sum_{subgroup} 1\{ x_j = k \mid y = 1\}$, $\sum_{subgroup} 1\{ x_j = k \mid y=0 \}$, $\sum_{subgroup} 1 \{ y = 1 \}$, $\sum_{subgroup} 1 \{ y = 0 \}$. The reducer then sums up intermediate results to get the final result for the parameters.

Gaussian Discriminative Analysis (GDA): The classic GDA algorithm [7] needs to learn the following four statistics $P(y)$; μ_0; μ_1 and $\sum$. For all the summation forms involved in these computations, we can leverage the map-reduce framework to parallelize the process. Each mapper will handle the summation (i.e. $\sum 1\{ y_i = 1 \}$, $\sum 1\{ y_i = 0\}$, $\sum 1\{ y_i = 0\} x_i$, etc) for a subgroup of the training samples. Finally, the reducer can aggregate the intermediate sums and calculate the final result for the parameters.

k-means: In k-means, the operation of computing the Euclidean distance between the sample vectors and the centroids can be parallelized by splitting the data into individual subgroups and clustering samples in each subgroup separately (by the mapper). To determine new centroid vectors, we can divide the sample vectors into subgroups, compute the sum of vectors in each subgroup in parallel, and finally the reducer can add up the partial sums and compute the new centroids.

Neural Network (NN): We can focus on back propagation [8] by defining a network structure (we use a three layer network with two output neurons classifying the data into two categories), each mapper propagates its set of data through the network. For each training data, the error is back propagated to calculate the partial gradient for each of the weights in the network. The reducer then sums the partial gradient from each mapper and does a batch gradient descent to update the weights of the network.

Principal Components Analysis (PCA): PCA [9] computes the principle eigenvectors of the covariance matrix $\sum = \frac{1}{m}\sum_{i=1}^{m} x_i x_i^T$ -- $\mu\mu^T$ over the data. In the definition for $\sum$, the term $\sum_{i=1}^{m} x_i x_i^T$ is already expressed in summation form. Further, we can also express the mean vector μ as a sum, $\mu = \frac{1}{m}\sum_{i=1}^{m} x_i$. The sums can be mapped to separate mappers, and then the reducer will sum up the partial results to produce the final empirical covariance matrix.

Independent Component Analysis (ICA): ICA [10] tries to identify the independent source vectors based on the assumption that the observed data are linearly transformed from the source data. In ICA, the objective is to compute the unmixing matrix W. We can use batch gradient ascent to optimize the *W*'s likelihood. In this, we can independently calculate the expression $[\ 1 - 2g\ (w_1^T x^{(i)}\ ]\ x^{(i)T}$ in the mappers and sum them up in the reducer.

5. Example

The Netflix Problem
Rating set: Over 100 million ratings by more than 480,000 users on over 17,000 movies.

Probe set: Around 1.4 million movie and user ids for which predictions must be made. Actual ratings are provided, so we can calculate the RMSE (root mean square error) rate.

Input Format: One text file containing data for a single movie. Each of these files contains the movie identification number of the movie as the first line. Every subsequent line contains comma separated values for a rater identification number, an integer rating greater than or equal to one and less than or equal to five given by that rater, and the date on which the movie was rated.

For example, 11674 (The Name of a Rose):
11674:
1331154, 4, 02-08-2004
551423, 5, 19-07-2004
716091, 4, 18-07-2005

Step 1: Data Preperation
Convert data to one line per movie:
movieID_D rater_i:rating_i,rater_j:rating_j,rater_k:rating_k,
movieID_E rater_u:rating_u, rater_v:rating_v,.....
For example the data for the movie "The Name of the Rose" will be transformed into the format:
11674 1331154:4,551423:5,716091:4,1174530:3,...

Step 2: Canopy Selection
Distance metric: if a set of z number of people rate movie A and the same set of z number of people rate movie B, then movies A and B belong to the same canopy.

Using this metric canopies may overlap, or in other words a movie may belong to multiple canopies. So far as each movie belongs to at least one canopy the necessary condition of canopy clustering will be met. Hence, in order for this to be true the value z must not be too large as the canopies may be

large and many data points may lie outside of canopies. If the value of z is too small then the number of canopies will be less and each canopy can have many data points. Hence, the eventual expensive data clustering may not be very good. Output is the canopy centers with their ratings data.

Map Step: Every mapper maintains a set containing the canopy center candidates it has determined so far. During every map the mapper checks if each successive movie is within the distance threshold of any already determined canopy center candidate. If the mapped movie is within the minimum then it is discarded, otherwise it is added to the set of canopy center candidates. The intermediate output sent to the reducer has the movieID as the key and the list of raterID-rating pairs as the value.

Reduce Step: The reducer repeats the same process. It takes the candidate canopy center movieIDs but removes those which are within the same threshold. In other words it removes duplicate candidates for the same canopy center. In order for this to work correctly the number of reducers is set to one.

Step 3: Mark by Canopy

Mark each movie from the full data set from Step 1 with the identification number of the canopies it belongs to. The two inputs used for this step are the output from Step 1 and the output from Step 2. The same distance metric from Step 2 is used to determine if the movie belongs to a particular canopy. The output will have the following format:

*movie*_A:

rater_i:rating_i,rater_j:rating_j,..;canopy_U,canopy_V,..

Map Step: Each mapper will load the canopy centers generated by Step 2. As each movie is received from the full data set the mapper determines the list of canopy centers that the movie is within, using the same distance metric from Step 2. The intermediate output is the movieID as the key and its raterID-rating pairs and list of canopies as the value.

Reduce Step: The reducers simply output the map output.

Step 4: Expensive Clustering: k-Means

The expensive clustering steps do not change the full movie data set. They merely move around the canopy centers so a more accurate clustering is determined.

The K-means clustering is performed repeatedly until convergence is achieved. In simple terms this means until the k-centers no longer change. However, in practice this can take an incredible amount of time or never be achieved at all. So for testing purposes the algorithm can be run iteratively up to five times and the final result considered converged. The expensive distance metric used in this step is cosine similarity. The two input data sets for this step are taken from data sets marked with canopies created by Step 3 and initially the canopy centers created by Step 2. The output will be a list with the new cluster centers (movieID) as the key and raterID-rating pairs list as its values in the same format as the output of the canopy selection MapReduce (Step 2).

Map Step: Each mapper takes the k-centers from the previous MapReduce into the memory. For the first iteration the canopy centers generated by Step 2 are used. Each movie that is mapped is also contains a list of the canopies it belongs to. Using the expensive distance metric the mapper determines which canopy the movie is closest to and outputs the chosen canopyID as the key and the mapped movie as the value.

Reduce Step: This step must determine the new center for every canopyID that it receives from the mapper. The process to do this involves determining the theoretical average movie in a canopy, then finding the actual movie that is most similar to this average value. When finding the average movie one determines the set of raters that belong to a canopy. For each of these raters it can be determined how they scored movies on average. With this information we can use cosine similarity to determine the movie most similar to this average movie.

Step 5: Inverse Indexer

This phase outputs results that can be used. The aim is to map each movie with a cluster center in an inverted index format. Hence, the cluster center movie identification numbers are used as the keys and the associated movie identification numbers are used as the values. The two inputs used for this step are the list of centroids and the full movie set. The output has the following format:

movieID_centroid

movieID_A:similarityA,movieID_B:similarityB,...

Map Step: The map loads the cluster centers determined by any one of the algorithms from Step 4. For each mapped movie that is within the cheap distance metric from Step 2 of any cluster center, the similarity is calculated for that movie to the cluster center using the appropriate distance metric. The intermediate output sent to the reducer will have the cluster center as the key and the mapped movieID and similarity as the value.

Reduce Step: The reduce simply concatenates the movieID similarity pairs for each cluster center.

Step 6: Data Prediction

Map Step: The database can be queried to find several similar movies using the following procedure:

a) If the movie is a cluster center then fetch several of the most similar to it.
b) If the movie is not a cluster center:
   - Find the cluster center it is closest to.
   - Fetch several movies most similar to this cluster center.

The intermediate output sent to the reduce is keyed by movieID and the value is a list of the similar movies.

Reduce Step: The reduce simply concatenates the similar movies for a particular probe movie.

6. Conclusion

Because of its simplicity and fault tolerance, MapReduce proves to be an admirable gateway to parallelizing machine

learning applications. The benefits of easy development and robust computation will come at a price in terms of performance, which is negligible when compared to advantages in terms of computation. Existing map-reduce frameworks can be optimized for batch processing systems. In this paper, by taking advantage of the summation form in a map-reduce framework, we can parallelize a wide range of machine learning algorithms MapReduce represents a promising direction for future machine learning implementations.

References

[1] M. Hilbert and P. L´opez, The worlds technological capacity to store, communicate, and compute information Science, 332(6025):60–65, 2011.
[2] A. Halevy, P. Norvig, and F. Pereira, "The unreasonable effectiveness of data", IEEE Intelligent Systems, 24(2), pp. 8–12, 2009.
[3] Cheng-Tao Chu, "Map-Reduce for machine learning on multicore", In NIPS, 2007.
[4] T. Chilimbi, Y. Suzue, J. Apacible, and K. Kalyanaraman. "Project Adam: Building an Efficient and Scalable Deep Learning Training System", In USENIX OSDI, 2014.
[5] R. E. Welsch and E. KUH, "Linear regression diagnostics", In Working Paper 173, Nat. Bur. Econ. Res.Inc, 1977.
[6] David Lewis, "Naive (bayes) at forty: The independence asssumption in information retrieval", In ECML98: Tenth European Conference On Machine Learning, 1998
[7] T. Hastie and R. Tibshirani, "Discriminant analysis by gaussian mixtures", Journal of the Royal Statistical Society, pages 155–176, 1996.
[8] R. J. Williams D. E. Rumelhart, G. E. Hinton, "Learning representation by back-propagating errors", In *Nature*, volume 323, pages 533–536, 1986.
[9] K. Esbensen Wold, S. and P. Geladi, "Principal component analysis", In Chemometrics and Intelligent Laboratory Systems, 1987.
[10] Sejnowski TJ. Bell AJ, "An information-maximization approach to blind separation and blind deconvolution", In Neural Computation, 1995.
[11] Jeffrey Dean and Sanjay Ghemawat, "MapReduce: Simplified data processing on large clusters", In ACM OSDI, 2004.
[12] Jim Gray "Scientific data managing in the coming decade", Technical Report MSR-TR- 2005-10, Microsoft Research, 2005.
[13] L. Bottou. Stochastic gradient descent tricks. In Neural Networks: Tricks of the Trade, pages 421–436. Springer, 2012.
[14] A. Cotter, O. Shamir, N. Srebro, and K. Sridharan, "Better mini-batch algorithms via accelerated gradient methods", In NIPS, 2011.
[15] Y. Koren, R. Bell, and C. Volinsky, "Matrix factorization techniques for recommender systems", IEEE Computer, 42(8):30– 37, 2009.
[16] R. Gemulla, E. Nijkamp, P. J. Haas, and Y. Sismanis, "Largescale matrix factorization with distributed stochastic gradient descent", In ACM KDD, 2011.
[17] M. Li, T. Zhang, Y. Chen, and A. J. Smola, "Efficient minibatch training for stochastic optimization", In ACM KDD, 2014.
[18] H. Cui, J. Cipar, Q. Ho, J. K. Kim, S. Lee, A. Kumar, J. Wei, W. Dai, G. R. Ganger, P. B. Gibbons, "Exploiting bounded staleness to speed up big data analytics", In USENIX ATC, 2014.
[19] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. Mc- Cauley, M. J. Franklin, S. Shenker, and I. Stoica, "Resilient distributed datasets: A fault-tolerant abstraction for in memory cluster computing", In USENIX NSDI, 2012.
[20] H. Yun, H.-F. Yu, C.-J. Hsieh, S. Vishwanathan, and I. Dhillon, "NOMAD: Non-locking, stOchastic Multi machine algorithm for Asynchronous and Decentralized matrix completion", In ACM VLDB, 2014.