Large Scale Multi-view Learning on MapReduce

Cibe Hariharan  
Ericsson Research India  
Email: cibehariharan@gmail.com

Shivashankar Subramanian  
Ericsson Research India  
Email: s.shivashankar@ericsson.com

Abstract—Across domains, many machine learning problems involve data which naturally comprises multiple views. Multi-view Learning is a machine learning technique that can utilize multiple views in a supervised, semi-supervised or unsupervised setup. Here we focus on co-training style multi-view learning algorithm under semi-supervised conditions which leverages both labeled and unlabeled data. In many domains, amount of (unlabeled) data available is very huge in size, which makes it impossible to learn serially in a single machine. Seminal work in this field was done by Blum and Mitchell, where they proposed a co-training based multi-view learning algorithm which bootstraps a set of classifiers from high confidence labels. In this work, we study various distributed multi-view learning using both consensus and complementary principles. We also propose an efficient computational design on Hadoop for learning multiple classifiers.

I. INTRODUCTION

Typically all the machine learning techniques assume the datapoints to have only one representation, which is also called as single view. But many real-world datasets possess additional information that can be utilized to improve the performance. For example, in text classification, sentences in a document can be represented using ‘bag of words’ representation which captures the lexical information. Alternatively, the sentences can also be represented using its ‘parse tree’ which captures the syntactic information. Rather than using either of the views (representations) separately or using a single representation by combining both views, it might be effective to use both views together where a model built on one of the views can aid learning the model built on other view. A relatively new machine learning technique to handle multiple views is called as Multi-view Learning [1], [2]. It has been successfully used in conjunction with traditional paradigms such as supervised, unsupervised and semi-supervised approaches [1], [2], [4], [6], [7]. Researchers have shown that utilizing multiple views together is more efficient than using a single view alone or using a combined representation of multiple views.

Seminal work in this field was done by Blum and Mitchell [1], where they proposed a “co-training” based multi-view learning algorithm which bootstraps a set of classifiers from high confidence labels. In that work, they assumed the views to be self-sufficient and conditionally independent given the class label. In other words, they assume a natural split of features into multiple views. For example, in web page classification, an input web page can be classified using its content or the text on hyperlinks pointing to this page. In object detection, an object can be classified using its color or shape. In a multi-modal setting, multiple views can be defined on signals from separate input sensors. For example, in biometrics, a person’s identity can be found using his fingerprint or iris scan inputs. It is impractical to expect such natural splits in all the domains. So, researchers started working towards finding suitable artificial feature splits and necessary conditions for multi-view learning to be effective. Nigam and Ghani [2] presented other SSL algorithms for multi-view learning, which includes co-EM and self-training. They also analysed the conditions necessary for “co-training” to be successful. Following Nigam and Ghani, many approaches for multi-view learning [4], [5], [6], [7], [16] exploit multiple redundant views to effectively learn a set of classifiers defined using each of the views. Multi-view learning is found to be advantageous compared to single-view learning especially when the weaknesses of one view complement the strengths of the other [1], [3], [6].

This field has recently been receiving a lot of attention in various communities under different themes depending on the problem addressed and the nature of solution proposed. Despite that, most of the works handle sequential/single machine learning only. Since many domains have datasets which are huge in size, it naturally raises the need for distributed multi-view learning. One of the attempts by researchers to handle this challenge was Tri-training on MapReduce [9]. Tri-training [13] is another co-training style semi-supervised algorithm, which does not require that the instance space be described with sufficient and redundant views, nor does it put any constraints on the supervised algorithm. Tri-training generates three classifiers from the original labeled example set which are then refined using unlabeled examples in the iterations. For each iteration, an unlabeled example is labeled for a classifier if the other two classifiers agree on the labeling. In this work [9], researchers propose a Hadoop implementation of tri-training algorithm. They also combine the method with data editing techniques to reduce the I/O involved by labeling points that would reduce the training error only. Since it handles Tri-training only and is also dependent on data editing techniques to reduce I/O, the proposed method cannot handle all variants of large scale multi-view learning algorithms efficiently. In order to overcome these drawbacks, we propose an efficient implementation of multi-view learning on MapReduce, for algorithms based on both consensus and complementary principles [11]. Consensus principle aims to maximize the agreement on multiple distinct views. The complementary principle states that in a multi-view setting, each view of the data may contain some knowledge that other views do not have; therefore, multiple views can be employed to comprehensively and accurately describe the data. We also compare various algorithms on standard benchmark datasets, to choose one for evaluating our proposed large scale implementation.
A. MapReduce

MapReduce is a distributed/parallel programming model for handling large data. Large data sets which cannot fit into memory are divided into many moderate chunks and then processed parallely. Map function takes a key/value pair as input, and produces one or more intermediate key/value pairs as output. Reduce function takes a set of intermediate values, with the same intermediate key, as input, and reduces them into one or more values. MapReduce libraries have been written in many programming languages, with different levels of optimization. A popular open-source implementation of MapReduce is Apache Hadoop [10]. In this work we use Apache Hadoop for evaluating distributed multi-view learning on MapReduce.

Organization of this paper is as follows: In Section II, we discuss about various multi-view learning algorithms proposed by researchers in the literature and provide comparative results using standard benchmark datasets. In Section III, we provide an efficient computational design for both complementary and consensus principle based multi-view learning method variants. In Section IV, we provide experimental results comparing the proposed computational design with default implementation on Hadoop. And finally in Section V, we discuss our contribution and also future work that could follow on.

II. COMPARITIVE STUDY OF MULTI-VIEW LEARNING ALGORITHMS

We compare Co-training [1], Co-EM [2], Tri-training [12], Committee Machines based multi-view learning [13] and One Teaching All multi-view learning approaches.

- **Co-training** was originally proposed for the problem of semi-supervised learning, in which there is access to labeled as well as unlabeled data. It considers a setting in which each example can be partitioned into two distinct views, and makes three main assumptions for its success: sufficiency, compatibility, and conditional independence. In the original co-training setup [1], two individual classifiers, \( h_1 \) and \( h_2 \), are trained on two views \( V_1 \) and \( V_2 \) respectively. And then each individual classifier labels some unlabeled datapoints \( U' \) to augment the labeled training examples \( L \) for re-training the classifiers. The intuition behind the co-training algorithm is that classifier \( h_1 \) adds examples to the labeled set that classifier \( h_2 \) will then be able to use for learning.

- Instead of committing labels for the unlabeled examples, in Co-EM we choose to run EM in each view and give unlabeled examples probabilistic labels that may change from one iteration to another. Co-EM outperforms co-training for many problems, but it requires the algorithm to process probabilistically labeled training data and the classifier to output class probabilities.

- **Tri-training** extends co-training algorithm for three learners. If two of them agree on the classification of an unlabeled point, the classification is used to teach the third classifier. This approach thus avoids the need of explicitly measuring label confidence of any learner.

- **Committee Machines** based multi-view learning [13] deals with multiple learners. The performance of traditional SVM-based relevance feedback approaches is often poor when the number of labeled feedback samples is small, thus researchers [13] developed multi-training SVM (MTSVM) to mitigate this problem. MTSVM combines the merits of the co-training technique and a random sampling method in the feature space. However, simply using the co-training algorithm with SVM is not realistic, because the co-training algorithm requires that the initial sub-classifiers have good generalization ability before the co-training procedure commences. Thus the authors employed classifier committee learning to enhance the generalization ability of each sub-classifier.

- **One Teaching All** multi-view learning can be seen as a multiple learner algorithm based on complementary principle. Here, the most confident learner teaches all other learners.

We compare the performance of these multi-view learning algorithms on standard benchmark datasets: WebKB\(^2\), Twitter [14] and synthetic dataset\(^3\). We use the content views of Twitter dataset. For datasets in WebKB and synthetic dataset that have two views only, in the case of tri-training, we use a third learner with different parameter choice. We use SVM as the learning algorithm on all the views. From the results in the following figures, we can see that both consensus and complementary principle based multiple learner algorithms — Committee Machines based multi-view learning and One Teaching All multi-view learning perform consistently well. In the following section, we propose an efficient computational design on Hadoop which can handle both types of learning efficiently. It can also handle other variants which does not use confidence. Since the computational study will not vary much for Committee Machines based multi-view learning and One Teaching All multi-view learning, we use Committee Machines based multi-view learning in the experimental results section.

\(^2\)http://www.cs.cmu.edu/webkb/
\(^3\)http://mlg.ucd.ie/datasets/segment.html
III. PROPOSED COMPUTATIONAL DESIGN FOR MULTI-VIEW LEARNING ON HADOOP

In this section, we elaborate our proposed computational design for multi-view learning on Hadoop. Default computational design using Hadoop is shown in Figure 1. In the default setup, classifiers on multiple views are learnt using the Mappers on each node individually, and the results are combined using Reducers. In such typical implementations [9], we would need another MR to compute labels and update all the nodes (or distributed cache in Hadoop). The default implementation is in-efficient for a couple of reasons (i) additional computational overhead to launch MR job for label updations (ii) necessity to broadcast new labels to all the nodes/distributed cache even though the data might be residing in a subset of the cluster nodes. In-order to overcome these drawbacks, we propose a computational design with the following components

- **Mapping Table**: This lookup table acts as a meta-data for the multiple view data partitioned across various nodes in the Hadoop cluster. It contains Node Id, Partition Id and View Id. Node Id refers to the node in the Hadoop cluster which has the data. Partition Id refers to the partition of view data which resides in the node and View Id refers the view index to which the data partition belongs to. **Mapping Table** is populated once before the process starts. This meta information is critical to avoid unnecessary I/O transfer to nodes which do not need information about label updates w.r.t datapoints belonging to other partitions.

- **Label File**: Label file is populated in each node for the partitions residing in it. It contains the labels of datapoints in each partition. This file can be cached in local filesystem, or in memory or can be a HDFS file. In the experiments we use a map file in HDFS to allow random access to its contents.

- **Bi-directional Reducers**: In typical distributed machine learning, Reducers are used to aggregate results from Mappers and update them on HDFS or distributed cache. In the proposed setup, we allow Reducers to take inputs from Mappers and also update labels in the Label File during the same iteration. Thus we refer to it as Bi-directional Reducers. It allows us to reduce the additional computational overhead required in the typical implementation [9]. They use the lookup **Mapping Table** to find where the **Label File** is residing, thus avoiding unnecessary broadcast.

In Figure 1 and 2, baseline and the proposed design are shown for a sample case with 2 views and 3 partitions, but it is generic for any number of given views and partitions. Basic pseudocode for the implementation of proposed design is given in Algorithm 1. We do not get into the details of learning local machine learning models on each partitions and computing a global model using bi-directional reducers. We assume that it is possible as in the case of Naive Bayes classifier or in other methods where parameters can be computed locally and aggregated globally [15]. Also, the mapping table created once, can be persisted after creation, and reducers can be scheduled

on the same machine by using any of the existing solutions [16]. Similarly, the label file updations for replications of partitions can be avoided using Haloop, where we can schedule Mappers on the cached node.

**Algorithm 1 Multi-view Learning on MapReduce**

Get views and partition details in each node by querying the Namenode

Initialize \( MappingTable(N, V, P) \)

for View \( V_i \) in \( V \) do
  for Partition \( P_j \) in \( P \) do
    Create a \( LabelFile(P_j, V_i, L_{ij}) \) with labels from initial Labeled data, and \textit{unk} for other datapoints
  end for
end for

repeat
  for View \( V_i \) in \( V \) do
    for Partition \( P_j \) in \( P \) do
      Learn \textit{model} using Mapper
    end for
    Compute global \textit{model} and labels \( L_{ij} \) using Bi-directional Reducer
    Update \( LabelFile(P_j, V_i, L_{ij}) \) by looking up \( MappingTable(N, V, P) \)
  end for
until Convergence

IV. EXPERIMENTAL RESULTS

In this section, we compare our proposed computational design on Hadoop (Fig 2) with default implementation on Hadoop (Fig 1). We use US Census KDD cup 1999 dataset and Poker Hand dataset from UCI repository\(^5\). US Census dataset has 4 million records and Poker Hand has 1 million records. We are using four commodity machine Hadoop cluster for our experiments. The datasets chosen do not have multiple views naturally, so we used random splits to convert them into two view datasets.

From the results it can be seen that except for the first iteration, which involves additional overhead of creating Mapping Table and Label File, across all other iterations the proposed method is much faster than the default Hadoop implementation. We used very less training ratio (1%) to study the time efficiency.

V. CONCLUSION AND FUTURE WORK

Though multi-view learning is drawing huge attention among the researchers working on different applications across communities, there has not been much attention paid towards large scale multi-view learning on a popular programming model such as MapReduce. In this work, we proposed a computational design which allows the reducers to not just update the ML model but also to push back the updated labels to the mappers. This reduces the overall I/O requirement by not broadcasting all the labels and additional computational requirements (extra MR iteration) as in the prior art [9] to compute and broadcast labels. The proposed method is generic to all variants of multi-view learning, which do not use confidence. As a future work, we want to incorporate ranking based on confidence value into the computational design. Secondly, we can utilize Haloop [16] like systems for caching Mapping Table and Label Files on the nodes where Mapper and Reducer jobs will be scheduled.

REFERENCES


\(^5\) http://archive.ics.uci.edu/ml/index.html
Fig. 1. Multi-view Learning on Hadoop

Fig. 2. Proposed Computational Design on Hadoop