Data-Intensive Multimedia Semantic Concept Modeling using Robust Subspace Bagging and MapReduce

ABSTRACT

With the rapid growth of multimedia data, it becomes increasingly important to develop semantic concept modeling approaches that are consistently effective, highly efficient, and easily scalable. To this end, we first propose the robust subspace bagging (RB-SBag) algorithm by augmenting random subspace bagging with forward model selection. Compared with traditional modeling approaches, RB-SBag offers a considerably faster learning process while minimizing the risk of overfitting. Its ensemble structure also enables a convenient transformation into a simple parallel framework called MapReduce. To further improve scalability, we also develop a task scheduling algorithm to optimize task placement for heterogenous tasks. On a collection consisting of more than 250,000 images and several standard TRECVID benchmark datasets, RB-SBag achieved more than a 10-fold speedup with comparable or even better classification performance than baseline SVMs. We also deployed the MapReduce implementation on a 16-node Hadoop cluster, where the proposed task scheduler demonstrates a significantly better scalability than the baseline scheduler in the presence of task heterogeneity.

1. INTRODUCTION

Recent years have witnessed an exponential growth of multimedia data fostered by cheaper data storage, explosion of digital content capture devices, and wider availability of distributed computing platforms on commodity hardware. Already, over 850 million photos are uploaded to Facebook each month[1], and more than 13 hours of video are uploaded to YouTube every minute[2]. Automatic extraction of media semantics [18, 20, 21] therefore becomes crucial for effective management of such massive amount of multimedia data and enabling semantic-based applications such as image and video retrieval, objectionable content filtering, and content categorization for advertisement and other monetization purposes. A promising approach for semantic content management of multimedia data is to model and detect a large number of semantic concepts in images and video, covering a wide range of categories, including people-related, objects, scenes, events, activities, and so forth.

Learning semantic concepts from multimedia content is typically posed as a set of binary supervised learning problems, which aim to categorize unannotated examples through low-level features. Among these modeling algorithms, kernel machines, such as support vector machines (SVMs), are regarded as the state-of-art methods [20]. However, given that many such algorithms bear a complexity at least quadratic to the number of labeled data, it is difficult for the plain-vanilla kernel-machine approaches to keep up with the data explosion in the era of terabytes or even petabytes, driven by an ever-increasing amount of data, a richer set of multimodal descriptors, and a more complex space of semantic categories. The computational demand of learning kernel machines can quickly become unaffordable, and thus hinder their practicality even if trying to be performance-savvy.

The emergence of a new computing paradigm, dubbed Data-Intensive Scalable Computing (DISC) [5], may offer a better solution for high-volume multimedia modeling and analysis. Different from conventional super-computing models, DISC mainly focuses on data rather than computation. The emphasis is to acquire and maintain continually changing and increasing data collections, in addition to performing large-scale computations over the data using high-level programming models. The leading example is Google, which uses its MapReduce framework [11] to process 20 peta-bytes of data per day. Inspired by Google's success, a number of scalable machine learning algorithms have been developed using MapReduce [8]. However, even with perfect scalability, simple learning parallelization may not be sufficient for multimedia processing, e.g., more than 270,000 CPUs are needed to process 1000 SVM models for YouTube videos in real time. It is therefore highly desirable to develop concept modeling methods that are not only scalable to distributed platforms, but also robust and much more efficient.

In this paper, we propose a new algorithm for data-intensive semantic concept modeling called Robust Subspace Bagging (RB-SBag), and its MapReduce implementation. To improve modeling robustness and efficiency, RB-SBag combines both random subspace bagging and forward model selection into a unified approach. It can automatically select the most effective base models learned from bootstrapped data examples and sampled feature spaces, before merging them into a composite classifier. The ensemble structure of RB-SBag also allows us to straightforwardly transform it into a two-stage MapReduce process. To improve scalability, we also propose a runtime model and a task scheduling algorithm to balance the durations of heterogenous tasks. Com-

[1] http://www.facebook.com/press/info.php?statistics

[2] `http://digital-stats.blogspot.com/2008/11/13-hours-of-video-are-uploaded-to_29.html`

pared with the state-of-art SVM-based methods, our experiments on a large collection containing more than 250,000 images and several standardized benchmarks, show that RB-SBag enjoys a learning speedup of an order of magnitude. Experiments using the MapReduce implementation on a 16-node Hadoop cluster further show that the proposed task scheduler can provide a significantly better scalability than the Hadoop scheduler in the presence of task heterogeneity. The proposed RB-SBag algorithm, coupled with its MapReduce implementation and improved scheduler, leads to a combined speedup of over two orders of magnitude.

2. RELATED WORK

The task of automatic semantic concept detection has been investigated extensively in recent years. It has been shown that, with enough training data, these classifiers can reach the level of maturity needed for semantic applications such as multimedia retrieval [18]. A large variety of learning approaches have been investigated, including SVMs, HMMs, kNN, logistic regression, AdaBoost, and so on. Among them, SVMs are considered as the state-of-art modeling approaches, with sound theoretical justifications [20]. However, few of these approaches have been shown to be scalable to datasets with hundreds of thousands, or even millions, of training instances. This issue becomes more critical given the recent emergence of numerous large-scale multimedia collections, e.g., the 80 million "tiny images" collected by Torralba et al. [25], 3.2 million images with 5247 synset annotation provided by ImageNet [12], and more than 100,000 YouTube video clips from LIBSCOM [14].

Advanced optimization methods have been proposed to speed up learning algorithms such as SVMs. Examples include down-sampling the data/feature space [24,27], and removing instances likely to be non-support vectors [22]. Another scalable option for concept detection is called search-based annotation [26]. It retrieves the top matched examples, and extracts the co-occurrence patterns from their associated tags. This is inspired by the fact that content-based retrieval can scale to millions of images with help from advanced feature indexing techniques, such as kd-trees and locality sensitive hashing (LSH). However, since search-based annotation assumes all concepts share the same distance metric and needs to keep every training data instance for the prediction phase, it is usually less portable and accurate than learning-based methods.

Our work is also related to the parallelization of kernel machine learning on distributed computing platforms [11] or multi-core processors [8]. These approaches can be categorized into two families. The first family of methods focus on rewriting and distributing the core optimization process, such as quadratic programming for SVMs. For instance, Zanni et al. [29] used parallel gradient prediction to decompose QP sub-problems using the gradient projection. PSVM [7] iteratively applies a row-based approximate matrix decomposition and distributes a fraction of the factorized matrix to each node. The second family of methods focuses on partitioning the training data to multiple subsets, and independently optimizing each subset. Collobert et al. [10] developed a parallel implementation for mixture of SVMs, which learns multiple SVMs on random subsets and combines them via a neural network gating function. Graf et al. [13] moved one step forward by iteratively combining and filtering a "cascade" of SVMs until the global optimum is reached. Both types of approaches parallelize the computation by approximation, but they bear different strengthes. The optimization-centric parallelism leads to a much finer parallelization granularity, and thus has the potential for better scalability on larger compute clusters. In contrast, data-centric parallelism is simpler to develop since it treats the learning algorithm as a "black box". It also requires less communication between worker nodes, which is more suitable for the MapReduce-type programming model. Finally, these two types of approaches are not mutually exclusive, and they can be applied together to improve machine learning scalability.

3. ROBUST SUBSPACE BAGGING

To improve the efficiency, robustness and scalability of semantic concept detection, we propose a modeling approach called *robust subspace bagging* (RB-SBag). It combines data sampling, feature sampling, and model selection into a unified learning framework. We designed this algorithm following the data-centric parallelization principle, because for data-intensive semantic modeling, it is not uncommon for users to work with hundreds or thousands of concepts simultaneously, and thus a coarse-grain parallelism will suffice.

3.1 Random Subspace Bagging

The proposed algorithm improves upon *random subspace bagging* (RSBag) [27], which we review in this section. Let us first present the notations and terminologies in this work. Let $\mathcal{X}$ be the training collection, where each example $\mathbf{x} \in \mathcal{X}$ is represented by multiple types of features, e.g., color, texture, motion, audio, and so on. The example x is also annotated with a number of concept labels $y_c \in \{-1, +1\}$, which indicate if x has the concept c or not. The goal of the concept modeling algorithms is to produce a *concept classifier* of the form $F_c : \mathcal{X} \rightarrow \mathbb{R}$. Ensemble learning approaches, e.g., bagging, combine the outputs from a family of *base models* $\mathcal{H}$. Each base model $h \in \mathcal{H}$ is a binary classifier that produces real-valued predictions $h : \mathcal{X} \rightarrow \mathbb{R}$. The base models can be generated from different learning algorithms, e.g., decision trees, support vector machines (SVMs), etc., and typically capture different aspects of the target concept. The learning algorithm then combines the multiple base models into a composite classifier as the final output.

The advent of RSBag is inspired by the success of bagging [3] and random subspace methods [15]. Its basic idea is to average a collection of base models learned from bootstrapped data samples in selected feature space. The most prominent example for RSBag is random forest [4], which aggregates an ensemble of unpruned classification/regression trees using both bootstrapped examples and selected features in the tree induction process. Random forest has been empirically demonstrated to outperform a single tree classifier. More recently, RSBag with SVM base models have been demonstrated to be effective in the tasks of image retrieval [24] and multimedia concept detection [27].

The details of RSBag are shown in Algorithm 1. From the training data, the algorithm learns N base models, each of which is constructed from a balanced set of bootstrapped samples from the positive data and the negative data with sample ratio r_d, unless the sample size is larger than data size. On the feature side, if the training data contains multiple feature descriptors, such as color correlogram, edge histogram, etc., each descriptor is iteratively selected. Then

Algorithm 1 Random subspace bagging (RSBag)

Input: concept c, number of models N per feature descriptor, training examples X_c with positive data X_{c+} and negative data X_{c-}, data sampling ratio $r_d(\leq 1)$.

1. For each descriptor D, i = 1 to N,
   (a) Select $\min(|X_{c+}|, |X_c|r_d)$ samples X_+^i from X_{c+}, and $\min(|X_{c-}|, |X_c|r_d)$ samples X_-^i from X_{c-};
   (b) Keep the entire descriptor, or select a random feature set D^i with sampling rate $r_f(\leq 1)$;
   (c) Learn a base model $h_c^i(\mathbf{x})$ with X_+^i, X_-^i and D^i.
   (d) Update $F_c^i(\mathbf{x}) \leftarrow F_c^{i-1}(\mathbf{x}) + h_c^i(\mathbf{x})$;
2. Output the final ensemble concept classifier F_c^N.

Algorithm 2 Robust subspace bagging (RB-SBag)

Input: Same as RSBag, plus a validation set $\mathcal{V}_c$.

1. For each descriptor D, for i = 1 to N, follow step (a) - (c) in Algorithm 1 to create a model pool $\mathcal{H}$
2. For each performance measure j, initialize $\mathcal{H}_j = \mathcal{H}$
   (a) While $\mathcal{H}_j$ is not empty, $i \leftarrow i + 1$
      i. Select base model $h_{cj}^i(\mathbf{x})$ that maximizes the measure j on $\mathcal{V}_c$, after being added to F_{cj}^{i-1};
      ii. $F_{cj}^i(\mathbf{x}) \leftarrow F_{cj}^{i-1}(\mathbf{x}) + h_{cj}^i(\mathbf{x})$;
      iii. Remove $h_{cj}^i(\mathbf{x})$ from the model pool $\mathcal{H}_j$.
3. Output the ensemble classifier F_c as the one with the best average precision on $\mathcal{V}_c$ among all $\{F_{cj}^i\}$, $\forall i, j$.

the algorithm can either use the entire descriptor space, or further sample a subset of features with a rate of r_f. Finally, all the base models are merged into an ensemble classifier.

By exploiting data and feature redundancy, RSBag can efficiently learn concept classifiers of smaller size. For example, if we learn 5 base models using SVMs with a 20% data sampling ratio and a 50% feature sampling rate, we can achieve a 10-fold speedup in the learning process [27]. Moreover, by adjusting the number of base models, RSBag also provides the flexibility to trade off learning performance and efficiency on the fly. These advantages make it a very good fit for data-intensive concept modeling.

To explain why the random subspace bagging can perform similarly as the baseline classifiers, we present an upper bound for its generalization error, which is controlled by two complementary ingredients, the model strength s and the model correlation $\bar{\rho}$, shown in the following theorem [4]:

THEOREM 1. *For each concept c in Algorithm 1, an upper bound for the generalization error is given by,*

$$E^*(F_c) \leq \bar{\rho}(1 - s_c^2)/s_c^2,$$

where s_c is the strength of base models for the concept c,

$$s_c = 2E_{\mathbf{x},y_c} P_\Theta(h(\mathbf{x}, \Theta) = y_c) - 1,$$

and ρ is the correlation between any two base models, i.e.,

$$\bar{\rho} = E_{\Theta,\Theta'}\left[\rho_{\mathbf{x}}(h(\mathbf{x}, \Theta), h(\mathbf{x}, \Theta'))\right].$$

Therefore, if the base models are sufficiently strong and with small correlation, the above theorem provides an important performance guarantee for random subspace bagging. However, without any controls on the base models, RSBag can potentially suffer from the risk of overfitting because the effectiveness of each data/feature subset may vary significantly. The correlation between classifiers is not always small either. As a result, RSBag is prone to suffer from the potential to overfit when a large number of low-quality base models are generated.

3.2 Robust Subspace Bagging

To reduce the risk of overfitting, we propose a concept modeling approach called Robust Subspace Bagging (RB-SBag) by controlling the strength and correlation of the selected base models. This can be achieved by incorporating a forward model selection step to iteratively find the best models based on a validation collection. In fact, numerous studies have shown that bagging can be benefit from such forward ensemble selection in terms of both effectiveness and efficiency [6, 19]. For instance, Carunna et al. [6] proposed several heuristics for selecting essential classifiers in a bagging ensemble. However, to the best of our knowledge, our proposed algorithm makes the first attempt to combine forward model selection with data/feature sampling in order to provide robust performance on massive datasets. This is especially critical when merging random-sampled base models, because their strengths tend to vary considerably and are in general much "weaker" than the full models.

The details of RB-SBag are shown in Algorithm 2. We reserve a portion of the labeled training data to serve as a validation set $\mathcal{V}_c$ for forward model selection. The base models are learned on the remaining training data $\mathcal{X}_c$ with the same steps as RSBag. After all the base models are generated, we re-compute forward model selection on 5 performance measures, i.e., average precision, accuracy, precision, recall and F1, in order to mitigate the risk being trapped in a local optimum. For each measure, the algorithm iteratively selects the most effective base model from the model pool, adds it to the composite classifier without replacement, and evaluate its average precision[3] on $\mathcal{V}_c$. Finally, it outputs the ensemble classifier F_{cj}^i with the highest average precision, where the number of selected base models i are usually much smaller than N. This selection step runs very fast, and typically prunes more than 70-80% base models in practice.

RB-SBag offers a number of advantages when we have massive labeled data. For example, sampling data and features allows us to efficiently learn base models with much less computational resources as well as less memory consumption. In the case of SVMs, it reduces the time complexity to $Nr_d^2 r_f$ times the full SVM baseline, and the memory consumption to Nr_d^2 times, where r_d and r_f sampling factors can be as low as 10-20%. This also allows us to perform more extensive parameter selection on each sample dataset. Moreover, model selection can not only improve the learning robustness, but also prune most useless base models, which brings a significant speedup in the detection phase. Note that while having better model robustness, RB-SBag may be affected by the risk of losing a small amount of labeled data

[3]Average precision is the primary concept modeling measure adopted by NIST in the TRECVID evaluation [21].

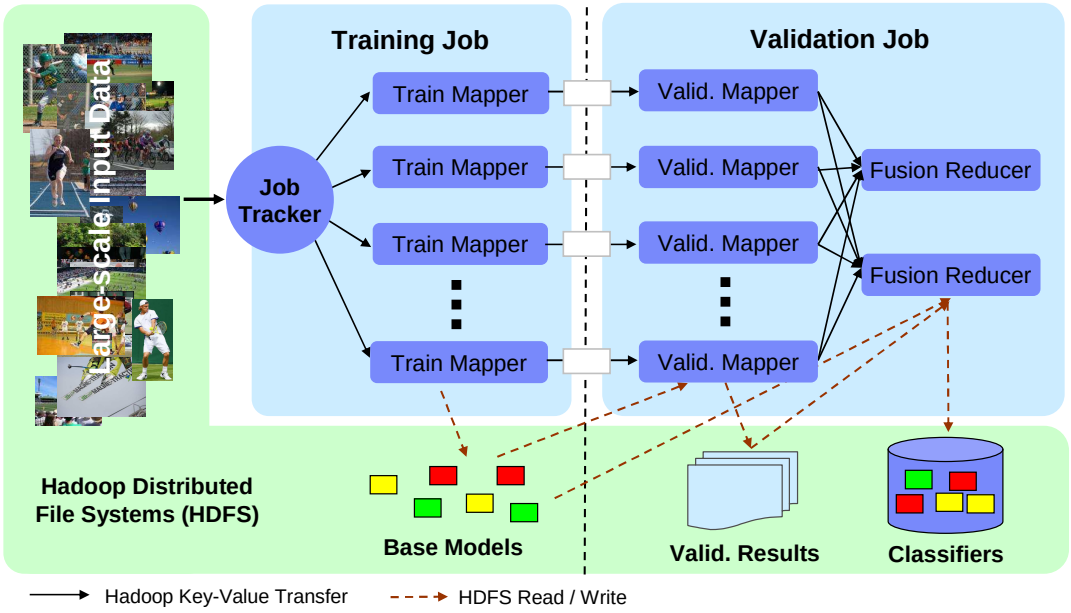


Figure 1: Illustration of the Map-Reduce implementation for Robust Subspace Bagging.

due to reserving it for model selection. This issue, however, is mitigated in the data-intensive learning scenario because of sufficient training data, as confirmed by our experiments.

4. MAP-REDUCE IMPLEMENTATION OF ROBUST SUBSPACE BAGGING

The need for distributed computing is apparent for modeling semantic concepts on massive multimedia data, which can range anywhere from tens of gigabytes, to terabytes or even perabytes. Inspired by the map and reduce functions commonly used in functional programming, Dean and Ghemawat [11] introduced a parallel computation paradigm called MapReduce. Its popular open-source implementation, Hadoop [1], has been successfully deployed to process hundreds of terabytes of data on at least 10,000 processors. Compared with other parallel programming frameworks, MapReduce provides the necessary simplicity by making the details of parallelization, fault-tolerance, data distribution and load balancing transparent to users. Also, this model is easily applicable to a wide range of data-intensive problems, such as machine learning, information extraction, indexing, graph construction and so on [11].

The programming model of MapReduce is as follows. Its basic data structures are a set of $\langle key, value \rangle$ pairs with user-specific interpretation. Two individual functions are needed for any computation, called *Map* and *Reduce*. The *Map* function first reads a list of input key and associated values, and produces a list of intermediate $\langle key, value \rangle$ pairs. After grouping and shuffling intermediate pairs with the same keys, the *Reduce* function is applied to perform merge operations on all intermediate pairs for each key, and emit output pairs of $\langle key, value \rangle$[4]. This model provides sufficient high-level information for parallelization, where the Map function can be executed in parallel on non-overlapping data partitions, and the Reduce function can be executed in parallel on intermediate pairs with the same keys. Its abstraction can be summarized by the following pseudo-code,

$$\begin{aligned} map &: (k_1, v_1) \rightarrow list(k_2, v_2), \\ reduce &: (k_2, list(v_2)) \rightarrow list(k_3, v_3). \end{aligned}$$

Because of its ensemble structure, RB-SBag can be straightforwardly transformed into a two-stage MapReduce process, corresponding to the first two steps described in Algorithm 2. Figure 1 illustrates the main idea of the MapReduce implementation for RB-SBag based on Hadoop. The first MapReduce job only contains a *training map function*, designed to generate and store the pool of base models, without using any reduce functions. The abstraction for its input and output key-values can be written as,

$$map_t : ([i,t], L_{train}) \rightarrow ([i,t], L_h),$$

where i is concept index, t is the bag index and their joint vector $[i,t]$ forms the mapping keys. For values, L_{train} is the location of training data, and L_h is the location of the output base model h. After all the base models are produced, the next MapReduce job computes the prediction results on the validation set $\mathcal{V}$ using a *validation map function*, conducts forward model selection and combines multiple models into composite classifiers using a *fusion reduce function*. Its abstraction can be written similarly,

$$\begin{aligned} map_v &: ([i,t], L_h) \rightarrow list(i, [L_h, L^h_{pred}]), \\ reduce_f &: (i, list[L_h, L^h_{pred}]) \rightarrow (i, L_{Ci}), \end{aligned}$$

where L^h_{pred} refers to the location of prediction results on $\mathcal{V}$, and L_{Ci} refers to the final composite classifiers. All the input, output and intermediate results are stored in the Hadoop distributed file systems (HDFS) which provides a large-scale data storage infrastructure based on clusters of commodity computers.

[4]Note that the input and output $\langle key, value \rangle$ pairs can have different formats

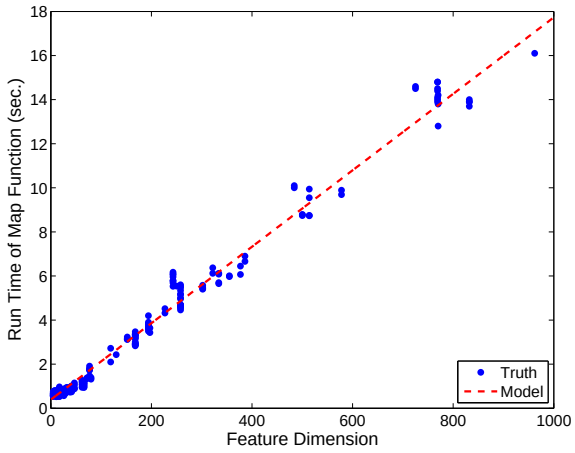


Figure 2: Runtime of map functions against feature dimension of its base model for concept "Vehicle". The dash line are fitted based on the time model.

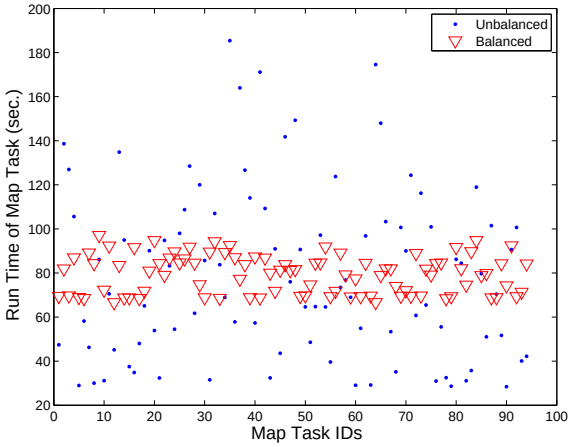


Figure 3: Run time distribution of training map tasks for "Vehicle" before and after balancing.

The above MapReduce process provides a coarse-grained parallelism at the level of base models. Therefore, existing learning tools such as libSVM or Weka can be directly re-used without any major re-development efforts. Under this design, the maximal number of processors that can be scaled to is limited by the total number of base models over all concepts. However, given the large variety of multimedia features, diversity of concepts, and scale of training data, this limit can easily grow beyond the common size of commodity clusters nowadays. For instance, following our default setting in this work that learns 200 base models per concept, the proposed algorithm can scale to 10,000 processors on a set of merely 50 concepts, whereas most typical Hadoop-based clusters contain less than 500 nodes [2].

5. HETEROGENEOUS TASK SCHEDULING

As a crucial component for distributed computing, scheduling aims to optimize the overall running time of MapReduce jobs. *Task* is the basic unit for scheduling in Hadoop, where each task can encapsulate multiple map / reduce functions. The Hadoop scheduling mechanism is implemented in a master node which runs a job tracker. The job tracker manages a number of worker nodes which can run one or more task trackers. When a task tracker notifies the master it has empty slots, the scheduler assigns it the next available task.

Such a passive scheduling algorithm works well when the following assumption holds, i.e., tasks in the same category (map or reduce) require roughly the same amount of time to execute [28], so that no obvious stragglers will slow down the job. However, this assumption is not valid in the MapReduce process for RB-SBag, because it can learn from heterogenous types of features with various dimensions, and the number of data samples are not guaranteed to be identical either. If task durations are not balanced, a long task with large number of features and samples can considerably drag down the scalability on the entire cluster.

In this section, we propose a task scheduling algorithm by estimating the running time of each map function and organizing complementary functions into tasks of similar estimated durations. We mainly investigate the scheduling method for the training map functions, because the reduce functions run much faster, and the validation map functions will also be balanced if the training functions are balanced.

5.1 Runtime modeling

We developed a runtime model to predict the running time of map functions based on historical data. For machine learning algorithms, a general runtime model for each map function can be described as follows,

$$T(|X|, |F|) = t_0 + |X|^{\alpha}|F|^{\beta}t_1 + \epsilon,$$

where t_0 is the initialization overhead, t_1 is the time scaling factor, $|X|$ and $|F|$ is the number of data samples and feature dimensions, α, β are their corresponding exponents, and ϵ is a noise term following a normal distribution $N(0, \sigma^2)$. The noise term captures the variance of system overhead, which is inherent to the run-time environment and thus cannot be eliminated by using better time models.

The runtime model is specific to the underlying learning algorithms. As an example, we use the state-of-the-art modeling algorithm, SVMs, to describe our parameter estimation method, but this method is generally applicable to other learning algorithms. To estimate the model parameters, we first collected the time statistics of map functions running on a single concept with fixed data samples but different types of features. Figure 2 plots the time distribution of 200 map functions against the number of feature dimension for learning concept "vehicle". Based on these results, we use regression techniques to estimate the parameters as $t_0 = 0.39\,\text{sec}, |X|^{\alpha}t_1 = 0.017\,\text{sec}, \sigma = 0.43$ and $\beta = 1$. Similarly, we can then estimate the parameter α to be 2, and $t_1 = 1.7 \times 10^{-8}\,\text{sec}$ based on the statistics across multiple concepts. As shown in Figure 2, our models provides a close approximation for the true run time of map functions. It is also in line with previous studies [17] which show that learning time for SVMs is linear with respect to feature dimensionality, and quadratic to data size.

5.2 Task scheduling using time models

With the proposed time models, we can naturally formulate task scheduling as a multi-processor scheduling problem, i.e., finding the minimum time to complete all map functions on M identical processors given the map function

Algorithm 3 The Multi-Fit algorithm

Input: M tasks, N map functions with duration T_i, iteration of binary search K.

1. (Optional) sort the map duration $\{T_i\}$;
2. Lower bound $B_l = \sum_i T_i/N$, Upper bound $B_u = 2 \cdot B_l$;
3. For k = 1 to K,
   (a) $B \leftarrow (B_l + B_u)/2$, $P_m \leftarrow 0$, placed(i) $\leftarrow$ false;
   (b) for i = 1 to N, for m = 1 to M,
      i. if T_i can be packed in m, i.e., $T_i + P_m \leq B$, then $P_m \leftarrow P_m + T_i$, placed(i) $\leftarrow$ true;
   (c) if $\forall$ placed(i), then $B_u \leftarrow B$; else $B_l \leftarrow B$;
4. Output the bound B and the map function placement.

i of length T_i. Unfortunately, this problem has been proven to be NP-hard. Therefore, several heuristic algorithms have been suggested in the hope of providing near-optimal solutions. A well-known approach is the MultiFit algorithm developed by Coffman et al. [9]. Its basic idea is to convert multi-processor scheduling into a series of bin packing problems, together with a binary search over bin capacity, to find the minimum capacity such that all jobs (i.e., map functions) can fit into M bins (i.e., tasks). Therefore, we adopt MultiFit for task scheduling. Algorithm 3 describes the algorithm details in our context. To illustrate, Figure 3 compares the runtime distribution of 96 training map tasks for "Vehicle" before and after scheduling. It clearly demonstrates our task scheduling algorithm can provide more balanced task distributions than the baseline scheduler.

Note that the list of map duration $\{T_i\}$ can be either sorted in a descending order, or left unsorted. These two choices correspond to two kinds of bin-packing algorithms, i.e. *First-Fit-Decreasing (FFD)* and *First-Fit (FF)*. Among these two choices, MultiFit-FFD has a slightly better worse-case performance, i.e. $1.2 + 2^{-K}$ times the optimal solution [9]. However, we found that the MultiFit-FFD tends to aggregate a large number of small map functions, as well as a small number of large functions. While their theoretical durations are roughly equivalent, their real performance can end up to be very different because of the unpredictable noise terms and system overhead. Alternatively, MultiFit-FF generates more balanced configurations of tasks, and leads to more robust runtime scalability in practice. More discussions on these two schemes are presented in Section 6.

6. EXPERIMENTS

This section presents the results of semantic concept modeling on a large-scale image collection, as well as several benchmark datasets, in order to demonstrate the effectiveness and scalability of the proposed algorithms.

6.1 Experimental Setting

Our evaluation is mainly carried out on a large-scale collection consisting of 261,480 images. These images are collected from diverse domains such as WWW crawls, social media sites, news videos and so on. All the images are manually annotated with 531 concepts organized in a customized

Figure 4: Image examples for the large-scale image collection. One image is shown for each concept.

visual taxonomy. These annotations are iteratively verified by professional users to guarantee their correctness. To obtain sufficient results in a reasonable time, we chose 63 representative concepts in our experiments. Figure 4 shows one image example for each of the 63 concepts, which covers a wide range of categories including activities, people, general objects, landmarks, sports and scenes. The number of positive examples ranged from 140 to more than 90,000 with an average around 4,300 for each concept. The negative images are automatically derived from the taxonomy. We randomly split the collection into the following: 80% as development data for learning semantic concepts, and the remaining 20% as testing data for evaluating modeling performance.

Because empirical results shows that concept detection can benefit from a rich set of features, we adopted a total of 98 types of visual features by generating 13 different descriptors (e.g., color correlogram, wavelet texture) on 8 granularities (e.g., global, grid) [23]. The feature dimensions ranged widely from 6 to more than 1000 [23]. The disk space of all these features sums up to be more than 150GB. The effectiveness of concept modeling is evaluated using average precision (AP) [21]. Mean average precision (MAP) is the mean of average precision over all the concepts.

We also examine RB-SBag on three TRECVID benchmark collections, i.e., TREC'05, '07 and '08. Their statistics are shown in Table 1(a). All these collections and concepts are officially provided, except TREC'05 is a subset sampled from its official version. We vary the features in these collections to examine the generalizability of RB-SBag. Three features are used in TREC'05 including color histogram, color correlogram and edge histogram. Three more global-level features are added to TREC'07 [23]. For TREC'08, we instead used a 1000-dimensional image codebook features based on local-feature-point SIFT descriptors [16].

The platform dedicated for our experiments is a 16-node Hadoop cluster. It consists primarily of commodity workstations running with 2.4GHz x86 processors and memory capacity of 2GB. We configure each worker node to run up to 2 map tasks and 2 reduce tasks simultaneously.

(a) Data and algorithm statistics. N_{train}, N_{test}, N_c, N_f are the number of training data, testing data, concepts and features. r_f, r_d are feature and data sampling ratios. N is the number of base models.

	N_{train}	N_{test}	N_c	N_f	r_f	r_d	N
TREC05	4894	1631	39	343	0.1	0.2	5
TREC07	21532	22085	20	825	0.5	0.2	5
TREC08	43617	112388	20	1000	0.5	0.2	5

(b) Performance comparison w.r.t. mean average precision. Speedup is the runtime ratio between Baseline and RB-SBag.

	Baseline	RB-SBag	Speedup
TREC05	0.4480	0.4520	50
TREC07	0.0638	0.0670	10
TREC08	0.1325	0.1322	11

Table 1: Data statistics and performance comparison on three TRECVID benchmark collections.

6.2 Results of Robust Subspace Bagging

We first report the experimental results on the TRECVID benchmarks. The baseline classifiers are learned using SVMs with all development data and features (**Baseline**). The kernel chosen for TREC'05 and TREC'07 is a RBF kernel $K(x,y) = e^{-\rho d(x,y)}$, $d(\cdot) = \|x-y\|^2$, and that for TREC'08 is a χ^2 kernel $d(\cdot) = \sum_i \frac{(x_i-y_i)^2}{x_i+y_i}$, based on the characteristics of their features. ρ is determined based on two-fold cross validation. This baseline leads to a very close performance to the state-of-the-art systems when similar features are used.

For RS-SBag, we split the development data into 80% as training set for learning base models, and 20% as validation set for model selection. A total of 5 SVM base models are learned for each concept. Because their entire collections can be loaded to memory, we simply concatenate all features into a single set of features. The selected data sampling ratio r_d and the feature sampling ratio r_f are listed in Table 1(a). They are determined using the same two-fold cross validation as learning ρ, where a list of pre-defined ratios are searcher for the minimal ones that can achieve at least 90% of the baseline performance. Table 1(b) shows RB-SBag can consistently achieve competitive or even better performance than the baseline SVMs, with a 10x-50x speedup on the modeling process. This demonstrates that RB-SBag can produce robust learning performance with significantly less training time.

Next, we present and discuss how RS-SBag performs on the large-scale image collection. Similar to above experiments, 80% of the development data is utilized for training, and the rest for validation. Base models are created by SVMs with a RBF kernel, with ρ estimated by two-fold cross validation. The data sampling ratio is set to 0.2, and each of 98 feature types are iteratively selected. We generate 2 base models per feature type, which results in 196 base models per concept. These are the default settings for our experiments unless stated otherwise.

Figure 5 compares MAP and learning time for robust subspace bagging with several other methods[5]. The baseline method learns and averages a single SVM base models for each feature type on the entire development data. We also compared RS-SBag with two other baselines as the intermediate steps, i.e., learning base models with model selection but no sampling (**Base-Valid**), and random subspace bagging (**RSBag**) with sampling but no model selection. It can be observed that on average, Base-Valid provides a similar performance as Baseline using less than 70% of the learning time. This efficiency improvement comes from the smaller training dataset for learning base models since a portion of the training data is reserved for model validation and selection purposes. In comparison, RSBag offered an more impressive 10-fold speedup over Baseline but its MAP is degraded due to overfitting. By merging the strengthes of both random sampling and model selection, RB-SBag achieves a more robust modeling performance with a negligible time increase over RSBag. This is consistent with our observations on the TRECVID benchmarks as well.

The overfitting effect of RSBag can be further illustrated by Figure 6, which shows the learning curves on the validation set for 3 frequent concepts and 3 infrequent concepts. With a growing number of base models sorted in descend-

[5]The relatively higher baseline performance for this collection is mainly because of the complete randomness of train-test partition and a less skewed data distribution.

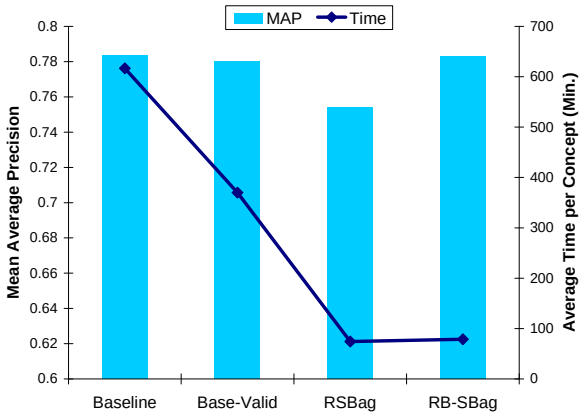


Figure 5: Comparison of testing MAP and average learning time between RB-SBag and three baselines on the large-scale image collection.

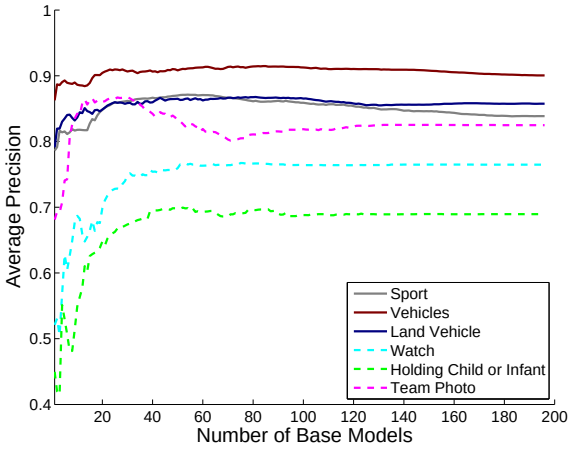


Figure 6: Validation performance with a growing number of base models for 6 selected concepts. 3 are frequent (solid), and 3 are infrequent (dashed).

(a) 10 Most Frequent Concepts

	Base	Base-V	RSBag	RB-SBag	S-Up
Human	0.906	0.915	0.901	0.913	16.48
Individual	0.728	0.742	0.709	0.756	12.26
Outdoors	0.945	0.957	0.939	0.960	7.44
Vehicles	0.914	0.937	0.895	0.937	11.90
Urban Scene	0.903	0.915	0.883	0.898	8.22
Group	0.779	0.789	0.757	0.791	9.30
Sport	0.867	0.878	0.852	0.860	11.45
Landmark	0.989	0.993	0.987	0.990	10.12
Land Vehicle	0.898	0.912	0.867	0.913	9.60
Nature	0.889	0.908	0.868	0.895	12.17

(b) 10 Least Frequent Concepts

	Base	Base-V	RSBag	RB-SBag	S-Up
White House	0.946	0.935	0.931	0.935	4.95
Stat. Liberty	0.879	0.874	0.796	0.853	6.86
Ice Scene	0.767	0.736	0.683	0.739	6.55
Parade	0.422	0.443	0.396	0.419	8.26
Football	0.929	0.917	0.878	0.902	5.96
Pyramids	0.933	0.932	0.891	0.926	5.33
Watch	0.917	0.928	0.876	0.915	9.33
Holding Child	0.606	0.591	0.585	0.577	3.70
Raft	0.850	0.801	0.790	0.799	8.31
Team Photo	0.672	0.618	0.526	0.565	5.26

Table 2: Testing MAP and learning speedup of RS-SBag and three other methods for 10 most and 10 least frequent concepts. The speedup S-Up is the runtime ratio between Baseline and RB-SBag.

ing order of their performance, we found that the modeling performance of these concepts saturates within a wide range of 20 $\sim$ 80 base models, after which some of the concepts start to overfit, e.g., "Sport" and "Team Photo". More importantly, the saturation points are inconsistent across concepts, and thus we cannot rely on a fixed cutoff threshold. Without a proper model selection and pruning step, RSBag can easily suffer from incorporating ineffective base models that are generated from small random samples.

Table 2 lists a more detailed report on the 10 most frequent concepts and the 10 least frequent concepts. By comparing these two groups side by side, we can obtain a deeper understanding on when and why the proposed algorithm succeeds or fails. For instance, RB-SBag can bring a larger learning speedup to the frequent group than the infrequent group, because for the frequent group, RB-SBag can sample more aggressively without being limited by the number of available positive examples. With respect to the learning performance, RB-SBag and Base-Valid appear to consistently outperform Baseline and RSBag in the frequent group, which again justifies the advantage of forward model selection. However, if we switch to the infrequent group, we can observe that both Base-Valid and RB-SBag have a worse performance than Baseline. This can be attributed to the redistribution of labeled data from model learning to forward selection, where 20% of the training data is reserved for validation purposes. Such a loss is not critical when training data is abundant but it is more likely to hurt the performance with less data available. Therefore, it is an interesting topic to study the trade-off between data used for model learning and selection in the future work.

To further investigate the sensitivity of data sampling ratio for RB-SBag, we plot Figure 7 to show the mean average precision against an increasing data sampling ratios from

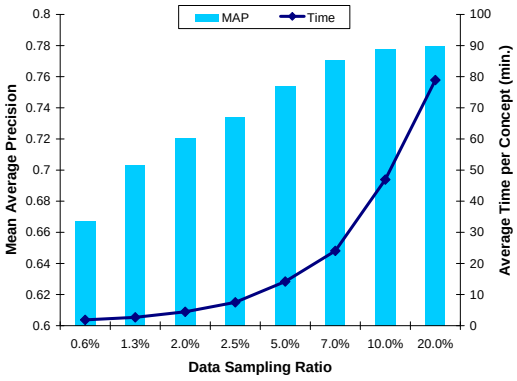


Figure 7: Testing MAP and average learning time against data sampling ratios for RB-SBag.

Configuration	2 nodes	4 nodes	8 nodes	16 nodes
Map Time (sec.)	71.0s	71.5s	71.8s	70.3s

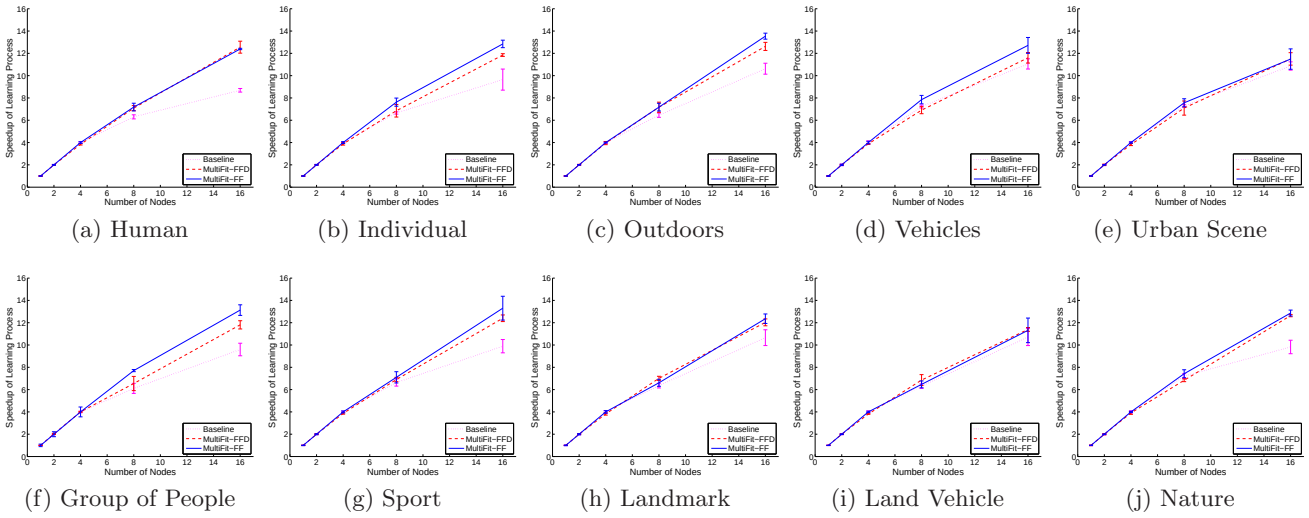
Table 3: Average runtime of training map tasks against the number of nodes.

0.6% to 20%. As can be seen, the learning performance increases dramatically at the beginning before reaching the plateau on 10% - 20%. However, this performance improvement comes at the price of a slower learning process, where the run time with ratio 20% is 10 times of the run time with ratio 2.5%. Therefore, the best sampling ratio should be dynamically selected based on users' requirement on performance and time consumption.

6.3 Results of MapReduce Implementation

In this section, we evaluate the scalability of the proposed task scheduling algorithm for the MapReduce implementation of RB-SBag. Because of resource constraints, the following experiments only evaluated the 10 most frequent concepts. For each concept, we spawn 1 map function to learn each of its base models, which sums up to $N = 196$ in total. These training map functions are then grouped into $M = 32$ map tasks for scheduling purpose. The Hadoop scheduler simply groups these functions in a sequential order. For the MultiFit algorithm in the proposed scheduler, we use First-Fit (FF) or First-Fit-Decreasing (FFD) bin packing algorithms, and set the maximum binary search iteration $K = 10$ unless stated otherwise. To obtain more stable estimation for each configuration, we repeat each experiment 3 times and report average performance.

To evaluate the scalability of the proposed scheduler, Figure 8 compares the learning speedup and its confidence interval against a growing number of nodes for each individual concept. The scalability of the baseline scheduler quickly goes down after running on more than 8 nodes. Such parallelization inefficiency is not due to the increasing overhead of disk I/O and network traffic. This can be confirmed by the runtime statistics of map functions shown in Table 3. We can observe that the average mapper runtime is almost equivalent for all configurations, and thus no slowdown happens in each base model learning process. Instead, as mentioned in Section 5, the slowdown stems from the non-informative scheduling algorithm in presence of task heterogeneity. In contrast, the MultiFit schedulers can provide significant improvement in terms of scalability for all



(a) Human (b) Individual (c) Outdoors (d) Vehicles (e) Urban Scene

(f) Group of People (g) Sport (h) Landmark (i) Land Vehicle (j) Nature

Figure 8: Comparing the learning scalability of three scheduling methods, i.e., Baseline, MultiFit-FFD and MultiFit-FF on each of the 10 most frequent concepts. The number of nodes grows from 1 to 16.

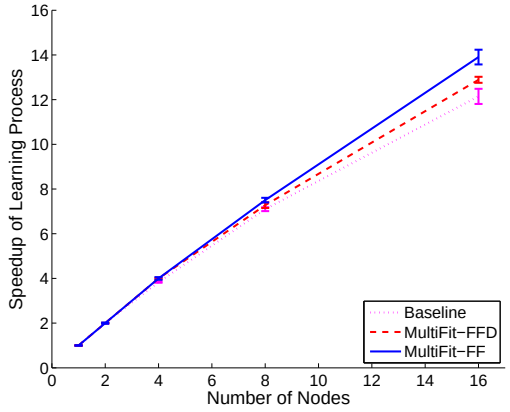


Figure 9: Comparing the scalability of three scheduling methods when learning all 10 concepts.

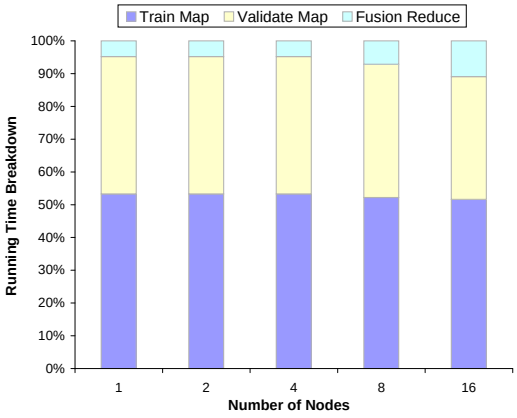


Figure 10: Runtime breakdown for each map and reduce process against the number of nodes.

concepts. For example, for concept "Human", the 16-node learning speedup improves from 8.5 using baseline to 12.5 using MultiFit, although it is not yet linear due to inherent runtime variance, limited reduce functions, and task overhead. These figures also show Multi-FF can consistently outperform MultiFit-FFD, because Multi-FF can provide a more balanced task assignment.

Fortunately, the larger the data collection and the concept list are, the fewer the unparallelizable components, and hence a higher scalability can be achieved. Figure 9 plots a similar comparison when learning all 10 concepts together, where MultiFit-FF can achieve a closer-to-linear speedup of 14.1 with more data available. For more detailed analysis, Figure 10 breaks down this MapReduce process into three parts, i.e., training map time, validation map time, and fusion reduce time. It shows that most of the computational time is spent in either map tasks. However, the reduce tasks become more non-negligible with larger number of nodes available, because currently the number of reduce functions is limited by the number of concepts, i.e., 10. This can be addressed by splitting and re-balancing the reduce functions, but we leave it for future work.

Figure 11 examines the effect of varying the number of map tasks from 32 (default) to 1960 (total number of base models) using the baseline Hadoop scheduler. It shows that the best task size for the baseline scheduler is around 128 to 256, based on the trade-off between scheduling flexibility and task overhead. However, even with the best task size, its runtime is still higher than MultiFit. This again confirms the superiority of the proposed scheduler.

Finally, it is worth pointing out that compared with baseline SVMs, combining the 10-fold speedup from RB-SBag and 14-fold speedup from MultiFit-FF can lead to a more than 140 times faster modeling process on 16 nodes. Such a speedup is difficult to achieve by using only traditional parallel machine learning algorithms.

7. CONCLUSIONS

In this paper, we proposed the Robust Subspace Bagging algorithm (RB-SBag) and its MapReduce implementation for data-intensive semantic concept modeling. To im-

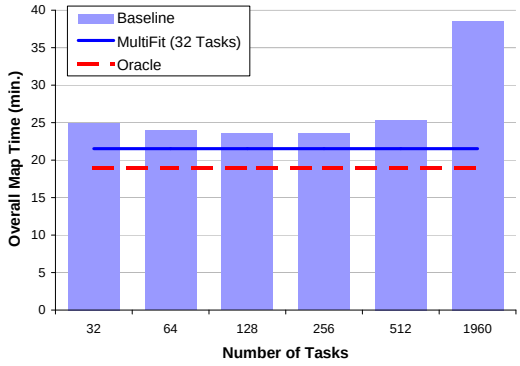


Figure 11: The overall map-task runtime against the number of map tasks for different schedulers.

prove modeling robustness and learning efficiency, RS-SBag combines random subspace bagging together with forward model selection to automatically select the most effective base models and merge them into a composite classifier. To our best knowledge, this model is the first attempt to combine data sampling and model selection into a unified framework. Compared with the state-of-art SVM-based modeling methods, our experiments on a quarter-million image collection and several standard TRECVID benchmarks show that RB-SBag can enjoy a learning speedup by an order of magnitude, without suffering from the risk of overfitting.

The ensemble structure of RB-SBag also allows us to easily transform it into a two-stage MapReduce process. To improve node balancing and task placement, we also proposed a runtime model and developed a MultiFit-based task scheduling algorithm to balance the estimated durations of heterogenous map tasks. Our implementation on a 16-node Hadoop cluster shows that the proposed task scheduler can provide a significantly better scalability than the baseline scheduler in presence of task heterogeneity. Putting everything together, the proposed approaches can provide a robust learning performance with a more than 140x speedup in comparison with the baseline SVMs.

8. REFERENCES

[1] Hadoop. http://hadoop.apache.org/.

[2] Hadoop wiki. http://wiki.apache.org/hadoop/PoweredBy.

[3] L. Breiman. Bagging predictors. *Machine Learning*, 24(2):123–140, 1996.

[4] L. Breiman. Random forests. *Machine Learning*, 45(1):5–32, 2001.

[5] R. E. Bryant. Data-intensive supercomputing: The case for disc. Technical report, School of Computer Science, Carnegie Mellon University, 2007.

[6] R. Caruana, A. Niculescu-Mizil, G. Crew, and A. Ksikes. Ensemble selection from libraries of models. In *Intl. Conf. of Machine Learning*, 2004.

[7] E. Y. Chang, K. Zhu, H. Wang, H. Bai, J. Li, and Z. Qiu. Psvm: Parallelizing support vector machines on distributed computers. In *Advances in Neural Information Processing Systems*, volume 20, 2007.

[8] C. Chu, S. Kim, Y. Lin, Y. Yu, G. Bradski, A. Ng, and K. Olukotun. Map-Reduce for machine learning on multicore. In *Advances in Neural Information Processing Systems: Proceedings of the 2006 Conference*, page 281. MIT Press, 2007.

[9] E. G. Coffman, M. R. Garey, and D. S. Johnson. An application of bin-packing to multiprocessor scheduling. *SIAM Journal on Computing*, 7(1):1–17, 1978.

[10] R. Collobert, S. Bengio, and Y. Bengio. A parallel mixture of svms for very large scale problems. *Neural Computation*, 14(5):1105–1114, 2002.

[11] J. Dean and S. Ghemawat. Mapreduce: simplified data processing on large clusters. *Commun. ACM*, 51(1):107–113, 2008.

[12] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[13] H. Graf, E. Cosatto, L. Bottou, I. Dourdanovic, and V. Vapnik. Parallel support vector machines: The cascade SVM. *Advances in Neural Information Processing systems*, 17(521-528):2, 2005.

[14] A. G. Hauptmann. Large analytics library and scalable concept ontology for multimedia research (libscom). `http://www.informedia.cs.cmu.edu/analyticsLibrary`.

[15] T. K. Ho. The random subspace method for constructing decision forests. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(8):832–844, 1998.

[16] Y.-G. Jiang, C.-W. Ngo, and J. Yang. Towards optimal bag-of-features for object categorization and semantic video retrieval. In *Proceedings of the 6th ACM Intl. Conf. on Image and video retrieval*, pages 494–501, 2007.

[17] T. Joachims. Making large-scale support vector machine learning practical. In A. S. B. Schölkopf, C. Burges, editor, *Advances in Kernel Methods: Support Vector Machines*. MIT Press, Cambridge, MA, 1998.

[18] Y. Lu, L. Zhang, Q. Tian, and W.-Y. Ma. What Are the High-Level Concepts with Small Semantic Gaps? . In *CVPR08*, 2008.

[19] G. Martínez-Mu noz and A. Suárez. Pruning in ordered bagging ensembles. In *Proceedings of the 23rd Intl. Conf. on Machine Learning*, pages 609–616, 2006.

[20] M. R. Naphade and J. R. Smith. On the detection of semantic concepts at trecvid. In *Proceedings of the 12th annual ACM international conference on Multimedia*, pages 660–667, New York, NY, USA, 2004.

[21] P. Over, T. Ianeva, W. Kraaij, and A. F. Smeaton. Trecvid 2006 overview. In *NIST TRECVID-2006*, 2006.

[22] N. Panda, E. Y. Chang, and G. Wu. Concept boundary detection for speeding up svms. In *Proceedings of the 23rd Intl. Conf. on Machine Learning*, pages 681–688, 2006.

[23] Same Author. N/A. In *N/A*, 2007.

[24] D. Tao, X. Tang, X. Li, and X. Wu. Asymmetric bagging and random subspace for support vector machines-based relevance feedback in image retrieval. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(7):1088–1099, 2006.

[25] A. Torralba, R. Fergus, and W. T. Freeman. 80 million tiny images: A large data set for nonparametric object and scene recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 30(11):1958–1970, 2008.

[26] C. Wang, F. Jing, L. Zhang, and H.-J. Zhang. Scalable search-based image annotation of personal images. In *Proceedings of the 8th ACM Intl. Workshop on Multimedia Information Retrieval*, pages 269–278, 2006.

[27] R. Yan, J. Tesic, and J. R. Smith. Model-shared subspace boosting for multi-label classification. In *Proceedings of the 13th ACM SIGKDD Intl. Conf. on Knowledge discovery and data mining*, pages 834–843, 2007.

[28] M. Zaharia, A. Konwinski, A. D. Joseph, R. H. Katz, and I. Stoica. Improving mapreduce performance in heterogeneous environments. Technical Report UCB/EECS-2008-99, EECS Department, University of California, Berkeley, Aug 2008.

[29] G. Zanghirati and L. Zanni. A parallel solver for large quadratic programs in training support vector machines. *Parallel Computing*, 29(4):535–551, 2003.