Multiple Instance Learning with Query Bags

Boris Babenko
UC San Diego
bbabenko@cs.ucsd.edu

Piotr Dollár
California Institute of Technology
pdollar@caltech.edu

Serge Belongie
UC San Diego
sjb@cs.ucsd.edu

Abstract

In many machine learning applications, precisely labeled data is either burdensome or impossible to collect. Multiple Instance Learning (MIL), in which training data is provided in the form of labeled bags rather than labeled instances, is one approach for dealing with ambiguously labeled data. In this paper we argue that in many applications of MIL (*e.g*. image, audio, text, bioinformatics) a single bag actually consists of a large or infinite number of instances, such as all points on a low dimensional manifold. For practical reasons, these bags get subsampled before training. Instead, we introduce a MIL formulation which directly models the underlying structure of these bags. We propose and analyze the *query bag model*, in which instances are obtained by repeatedly querying an oracle in a way that can capture *relationships* between instances. We show that sampling more instances results in better classification performance, which motivates us to develop algorithmic strategies for sampling many instances while maintaining efficiency.

1 Introduction

Traditional supervised learning requires example/label pairs during training. However, in many domains precisely labeled data is either burdensome or impossible to collect. *E.g*., less effort is required to specify what digits are present in an image than to accurately delineate their location, a problem first studied by Keeler et al. [15]. There may also be inherent ambiguity during labeling. The multiple instance learning framework (MIL), introduced by Dietterich et al. [10], provides a general paradigm for weakly supervised learning. Instead of example/label pairs, MIL requires unordered sets of instances, or *bags*, and labels are assigned to each bag, rather than each instance. A bag is labeled positive if it contains at least one positive instance. In recent years MIL has received significant attention and numerous algorithms and applications have been proposed [19, 1, 24, 4, 23, 7].

In many MIL applications a bag is generated by taking an object, *e.g*. an image or audio wave, and splitting it into overlapping pieces, which serve as the instances. However, both the existing MIL theory and algorithms are underdeveloped for this scenario. *E.g*., a typical data model used in classic MIL theory papers [2, 17] assumes that bags are finite and all instances are i.i.d., which is not appropriate for many of the datasets we consider here. While existing MIL algorithms do not always explicitly make these assumptions, they do not offer a principled way of dealing with large or infinite bags. Instead, instances are sampled prior to training. Note that a sampled positive bag may may actually contain no positive instances. Furthermore, by separating sampling from training, the two phases cannot be jointly optimized.

These observations inspire us to propose the **query bag model**[1] for MIL. In this model, instances are obtained by repeatedly querying an oracle in a way that can capture *relationships* between instances (*e.g*. in many applications all instances in a bag lie on a low dimensional manifold). We also propose to integrate the query bag model directly into existing MIL algorithms. To do so we turn to the filtering paradigm, where the data is drawn from a continuous stream or oracle [6], and extend these ideas to MIL with query bags. We show extensive experiments to validate the proposed approach.

[1] Note that although the terminology is similar to [11], our model and assumptions are quite different.

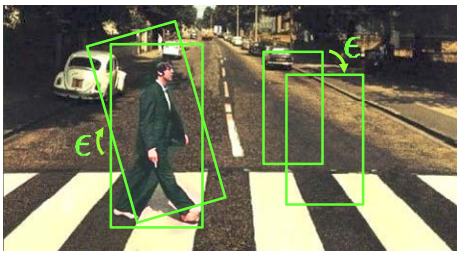


Figure 1: Object detection is a typical MIL application. In this example a pedestrian is known to exist in the image but at an unknown location. The image can be broken up into multiple overlapping patches using a sliding window, at multiple scales and orientations, to create a positive *bag*. Note that the number of instances is essentially infinite, since a sliding window can change location by some small ϵ. If we subsample the bag by randomly selecting a limited number of patches, we run the chance of missing the object of interest; moreover, relationships between instances are discarded. Instead, in this work we propose strategies that more directly take advantage of such bag structure.

There are numerous MIL applications where our model is appropriate. In computer vision, an image that contains an object at an unknown position and scale can be partitioned into overlapping sub-images, one of which will contain the object, either by sliding a rectangular window [23] or through segmentation [1]. In computer audition, prior to processing an audio wave can be partitioned either in the spatial domain [18] or in the frequency domain [21]. In text processing, a document can be partitioned into many pieces by using a sliding window [1]. Similarly, in bioinformatics, a label is often assigned to a long sequence even though a short sub-sequence is responsible for a positive label [20].

In these scenarios the number of instances in a bag can become prohibitively large or even infinite (*e.g.* if sub-pixel image window locations are used). The query bag model is well suited for the above scenarios; in fact, while surveying the literature we found very few applications for which it was *not* well suited (*e.g.* [10]).

2 The Query Bag Model for MIL

We begin with a review of supervised learning and MIL with the **classic bag model**. In standard supervised learning data consists of instances $\{x_1, ., x_n\}$, $x_i \in \mathcal{X}$, and labels $\{y_1, ., y_n\}$, $y_i \in \mathcal{Y}$, where the instance/label pairs are usually assumed to be drawn i.i.d. from some distribution $(x_i, y_i) \sim \mathcal{D}_{x,y}$. The goal is to learn a classification function $h : \mathcal{X} \to \mathcal{Y}$ that generalizes well to unseen data. In MIL with the classic bag model data is given in the form $\{X_1, ., X_n\}$, where each $X_i = \{x_{i1}, ., x_{im}\}$ is a bag, with associated labels $\{Y_1, ., Y_n\}$, $Y_i \in \{0, 1\}$. For notational simplicity we assume all bags have the same cardinality m. A bag is positive if one or more of its instances are positive according to an unknown classifier h^*:

$$Y_i = \max_{j \in \{1...m\}} \{h^*(x_{ij})\} \tag{1}$$

Alternatively, to model noisy instance labels, we can write $Y_i = \max\{y_{ij}\}$, where y_{ij} need not equal $h^*(x_{ij})$. Typically, the instances/label pairs are assumed to be drawn i.i.d. from some distribution $(x_{ij}, y_{ij}) \sim \mathcal{D}_{x,y}$, much like in the standard supervised setting.

In the **query bag model** model, rather than being given n bags with m instances each, we model each bag as an oracle that can be queried to obtain an arbitrary number of instances per bag. Furthermore, we design the query model so we can ask for *nearby* instances.

Each query bag i is represented by an object $o_i \in \mathcal{O}$ (*e.g.* a large image). To sample an instance from o_i, we must specify a location parameter $\alpha \in \mathcal{A}$ (*e.g.* coordinates of an image patch). A sample $x_{ij} \in \mathcal{X}$ is then generated using a query function $\mathcal{Q} : \mathcal{O} \times \mathcal{A} \to \mathcal{X}$. We write $x_{ij} = \mathcal{Q}(o_i, \alpha_{ij})$. Given certain assumptions about $\mathcal{Q}$, formalized below, we can query $\mathcal{Q}(o_i, \alpha + \epsilon)$ for small $\epsilon \in \mathcal{A}$ to get an instance near $\mathcal{Q}(o_i, \alpha)$. The query function $\mathcal{Q}$ and the associated spaces are problem specific, examples are given in Sec. 2.1.

We can express the set of all instances in a bag i using $X_i = \{x | x = \mathcal{Q}(o_i, \alpha), \alpha \in \mathcal{A}\}$. Note that unlike in the classic bag model, this set may be potentially infinite. Likewise, we can define the bag label for query bags in a manner analogous to Eqn. 1:

$$Y_i = \max_{\alpha \in \mathcal{A}} \{h^*\big(\mathcal{Q}(o_i, \alpha)\big)\}, \tag{2}$$

where h^* is again an instance classifier that determines the bag labels. In addition, for each bag i, we associate a distribution $\mathcal{D}_i$ over $\mathcal{A}$ that provides some prior information on where positive instances are likely to be located. Most often no prior knowledge is available and this distribution is uniform.

Let $\tilde{p}$ be the probability that $h^*(\mathcal{Q}(o_i, \alpha_{ij})) = 1$ given that bag i is positive and $\alpha_{ij} \sim \mathcal{D}_i$. $\tilde{p}$ is a measure of the expected informativeness of $\mathcal{D}_i$ and helps determine the difficulty of the resulting

MIL problem. Given m independent draws from a positive bag, the probability that all m instances are negative is $(1-\tilde{p})^m$. Thus, to determine a bag's label with confidence δ, we must query m instances, where:

$$m \geq \log(1-\delta)/\log(1-\tilde{p}) \tag{3}$$

Given query bags, we can convert them to a suitable format for standard MIL algorithms by simply querying m instances per bag prior to training. We can define a sample of the i^{th} bag, $X_i = \{x_{i1}, ., x_{im}\}$ where each x_{ij} is generated using $\mathcal{Q}(o_i, \alpha_{ij})$, where $\alpha_{ij} \sim \mathcal{D}_i$. Note that by sampling, there is a chance that for a positive bag, Eqn. 1 will no longer hold. This observation suggests that sampling more instances for each bag is advantageous. We study this in more detail in Sec. 3.

One intuitive interpretation of the query bag model is that all instances in a bag i are noisy observations of some **key** instance x_i^*. Let $\alpha_i^* = \text{argmax}_\alpha\{h^*\big(\mathcal{Q}(o_i, \alpha)\big)\}$ and $x_i^* = \mathcal{Q}(o_i, \alpha_i^*)$. We call x_i^* the key to bag i – the key need *not be unique* and is arbitrary for negative bags (since $\forall\alpha, h^*(\mathcal{Q}(o_i, \alpha)) = 0$). Regardless, every instance x_{ij} in a bag can be written as being some $\epsilon_{ij} \in \mathcal{A}$ away from the key instance: $x_{ij} = \mathcal{Q}(o_i, \alpha_i^* + \epsilon_{ij})$. If the ϵ_{ij} were known we could recover the α_i^* and thus the keys x_i^* and the problem would reduce to the standard supervised case. Instead, we can query noisy observations, where $\mathcal{D}_i$ is the noise distribution (specifically $\epsilon_{ij} \sim \mathcal{D}_i - \alpha_i^*$).

To make the model more tractable, we assume $\mathcal{Q}$ is well behaved. Typically, we expect $\mathcal{Q}$ to at least be continuous w.r.t. $\alpha \in \mathcal{A}$, and often we can additionally assume that $\mathcal{Q}$ is differentiable. In most of the examples that follow $\mathcal{A} = \mathbb{R}^d$ and $\mathcal{X} = \mathbb{R}^D$ where $d \leq D$; note that under these conditions, a bag defines a d-dimensional *manifold* in $\mathbb{R}^D$ (*cf*. Fig. 2). Knowing that $\mathcal{Q}$ meets these requirements can be useful. *E.g*. given a classifier $h : \mathbb{R}^D \to [0, 1]$ that is also differentiable, we can attempt to find the maximum of $h(\mathcal{Q}(o_i, \alpha))$ w.r.t. α using gradient descent. This also allows us to query the oracle for *nearby* instances. Given $x_{ij} = \mathcal{Q}(o_i, \alpha_{ij})$, we can query an instance $x'_{ij} = \mathcal{Q}(o_i, \alpha_{ij} + \epsilon)$ such that for small $\epsilon \in \mathbb{R}^d$, $||x_{ij} - x'_{ij}||_2$ is small (see Sec. 4).

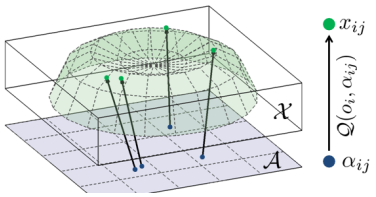


Figure 2: Data generation model.

Datasets where instances lie on low dimensional manifolds have been studied extensively and many interesting techniques have been proposed to take advantage of their structure. These range from techniques for estimating distances [22] to algorithms that traverse the manifold during training [9]. We draw inspiration from this work and propose some effective filtering strategies that take advantage of the manifold structure in Sec. 4.

2.1 Examples of Query Bags

We now consider some concrete, illustrative examples of query bags. These examples are by no means comprehensive, rather they are meant to clarify the model and allow us to perform a number of carefully controlled experiments in Sec. 3.1. For each model we define $\mathcal{Q} : \mathcal{O} \times \mathcal{A} \to \mathcal{X}$, and also $\mathcal{O}, \mathcal{A}$ and $\mathcal{X}$. Remaining details are deferred to the experiments.

Line Bags: A simple case of bags that lie on a manifold is when instances fall on a line. Let $o_i = \{u_i, v_i\}$ where $u_i, v_i \in \mathbb{R}^D$, $\mathcal{A} = \mathbb{R}^1$, and the query function be $\mathcal{Q}(o_i, \alpha) = u_i + \alpha v_i$. In other words, $\mathcal{Q}(o_i, \cdot)$ defines a line that extends in direction v_i and passes through point u_i. Each instance $x_{ij} \in \mathcal{X} = \mathbb{R}^D$ is a point on that line.

Hypercube Bags: Under this model each bag i is defined to be a hypercube $\mathbb{R}^D$ (or a square in $\mathbb{R}^2$) with side length $2r$ and center u_i. Let $o_i = \{u_i, r\}$ where $u_i \in \mathbb{R}^D$ and $r \in \mathbb{R}$, and let $\mathcal{A} = [-1, 1]^D$. The query function is $\mathcal{Q}(o_i, \alpha) = u_i + \alpha \cdot r$.

Image Translation Bags: Let $\mathcal{I}$ represent a large image and $x = \mathcal{I}(\alpha)$ a $p \times p$ patch cropped from image $\mathcal{I}$ centered at location $\alpha \in \mathbb{R}^2$. Each image corresponds to a bag and patches cropped from the image are instances, *e.g*. for face detection, α_i^* for a positive bag i would specify the coordinates of a face. More formally, $A = \mathbb{R}^2$, $\mathcal{X} = \mathbb{R}^{p \times p}$, $o_i = \mathcal{I}_i$ and $\mathcal{Q}(o_i, \alpha) = \mathcal{I}_i(\alpha)$.

The query bag model is appropriate for many typical MIL applications. *E.g*., it is straightforward to extend image bags, defined above, to include other image transformations, such as rotation and scale change, by increasing the dimension of $\mathcal{A}$ to account for all degrees of freedom and updating $\mathcal{Q}$ accordingly. Also, recall the various applications discussed in Sec. 1. For both bioinformatics

and text processing, $\mathcal{A}$ could encode the position and size of a sliding window; for audio processing, $\mathcal{A}$ could encode the frequency bandwidth. Further details are omitted for space.

3 Implications of Bag Size

For the classic bag model the notion of bag size is clearly defined by the number of instances in the bag, which stays constant for any given bag. On the other hand, for our query bag model the number of instances in a bag, m, is variable since it depends on the number of instances we decide to sample. Therefore, one interpretation of bag size in our model is how much we choose to sample the bags. Alternatively, we could consider the distribution $\mathcal{D}_i$ to define bag size. For example, consider the image bag shown in Fig. 1. $\mathcal{D}_i$ in this case could be uniform over the entire image, or over a smaller region around the pedestrian (this depends on the data and is out of our control). The bag in the former case is effectively larger than the latter case. Next we perform some experiments studying the effects of these interpretations of bag size on the performance of a MIL algorithm.

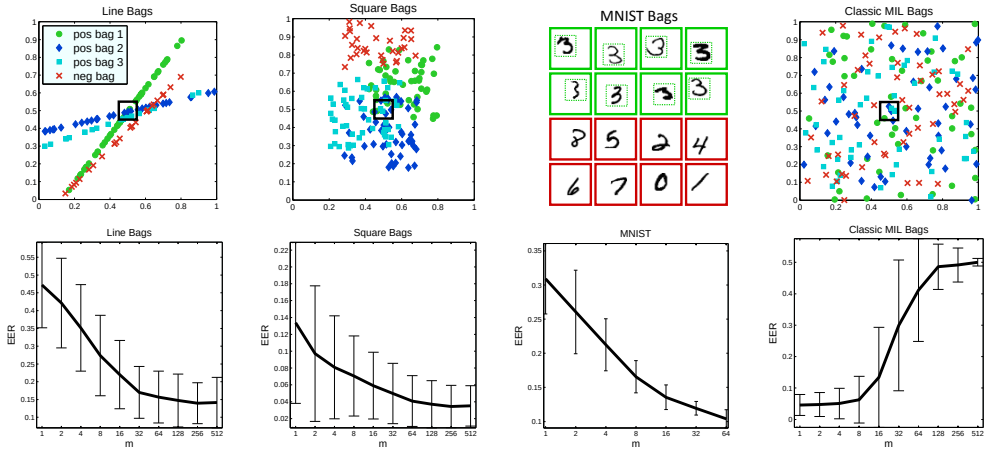


Figure 3: **Error versus** m. (TOP) illustrations of the datasets; for synthetic datasets the black square designates the positive region, and point color/symbol indicates bag membership. (BOTTOM) Equal error rate (EER) vs. m. See text for details.

3.1 Experiments: Error versus m

We begin with a set of experiments that study the performance of a MIL algorithm as we increase the number of instances we sample from each bag. In all experiments we use MIL-BOOST, proposed by Viola et al. [23]. In Sec. 4 we propose some modifications so MIL-BOOST can work directly with the query bag model; however, for the experiments that follow we use the original algorithm. To do this we sample a fixed number of instances m per bag prior to training. We expect performance of other MIL algorithms to be qualitatively similar.

We measure the effect of varying m on 2 synthetic datasets and 1 real dataset: (1) line bags, (2) hypercube bags, and (3) image translation bags (with the MNIST image dataset). Thorough details on how each dataset was generated are given below. All errors reported are bag errors. We estimate the predicted bag label by sampling 100 instances per bag.

The datasets and plots showing performance averaged over 25 trials are shown in the first three columns of Fig. 3. As expected, the results show that *as m increases error decreases*. Furthermore the MNIST results closely resemble the two synthetic cases, suggesting the query bag model correctly captures the properties of such data. Since our model and these experiments suggest that sampling more instances is advantageous, we must consider the computational consequences. In Sec 4 we propose strategies that allow us to sample many instances while being efficient.

Details for synthetic experiments: The first 2 experiments involve 2D point datasets ($\mathcal{X} = \mathbb{R}^2$). In each case $h^*(x_{ij}) = 1$ if x_{ij} falls in the square region spanning $[0.45, 0.55]^2$ (the 'positive region'). For both training and testing 50 positive and 50 negative bags are used. Decision stumps served as the weak classifiers. Remaining details follow: (1) For each line bag, u_i is sample uniformly from the positive region for positive bags and from outside it for negative bags; v_i is sampled uniformly from the unit circle (making sure that the resulting line for negative bags does not pass through the positive region). $\mathcal{D}_i = \mathcal{N}(0, 1)$ for each i. (2) For each hypercube bag, u_i is sampled uniformly

from $[0.25, 0.75]^2$ for positive bags and from outside of this region (but inside $[0, 1]^2$) for negative bags. $\mathcal{D}_i$ is uniform over $\mathcal{A} = [-1, 1]^2$ and $r = 0.2$. See Fig. 3, top, for example bags.

Details for MNIST digit experiment: The final experiment was based on a variation of the MNIST handwritten digits [16]. Each digit is originally a 28×28 image; we pad each with an 8 pixel border and randomly translate the digit within the resulting 44×44 image. Each resulting bag $\mathcal{I}_i$ contains 256 instances (28×28 patches), one of which is the original image (see Fig. 3). No prior knowledge is assumed, and thus $\mathcal{D}_i$ is uniform. We arbitrarily labeled bags containing '3's as positive and the rest negative. Decision stumps over Haar features served as the weak classifiers as in [23]. We use 100 positive and negative bags for training, and the rest of the data for testing.

3.2 Experiments: Classic Bags

We now briefly review the behavior of classic bags with respect to bag size. Using the classic bag model, Blum and Kalai proposed a PAC algorithm [5] with sample complexity $\tilde{O}\left(D^2 m/\varepsilon^2\right)$ (where D is dimensionality of $\mathcal{X}$, m is the number of instances per bag, and ε is the error). Previous results reported sample complexity that included even higher powers of m [2, 17]. All of these results suggest MIL becomes more difficult as the bag size increases. Given the independence assumption these results are intuitive – the larger the bag, the more difficult it is to identify positive instances in positive bags. Note that unlike in our model, here we have no control over the value of m.

For comparison, we perform an experiment similar to the ones in the previous section. We generate synthetic 'classic' bags, modeled on the experiment described in Fig. 2 of [19]. We use a setup similar to the previous synthetic experiments; we generate bags by uniformly sampling m points from $[0, 1]^2$ and assign the bag a positive label if any of these points fall into the positive region. The results are shown in the last column of Fig. 3; as is predicted by the PAC bounds, the error goes up as bag size increases. The more important point, however, is that this bag model is not appropriate for MIL datasets like the ones descried in Sec. 1.

3.3 Experiments: Error versus $\tilde{p}$

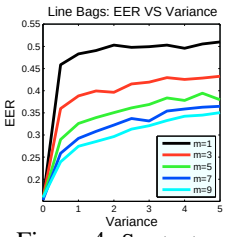


Figure 4: See text.

Finally, we study the alternate interpretation of bag size for query bags. Recall that that the distributions $\mathcal{D}_i$ provide prior information on where positive instances are likely to be located and $\tilde{p}$ quantifies their expected informativeness. Eqn. 3 suggests that for a fixed bag size m a lower value for $\tilde{p}$ would make learning more difficult, as sampled positive bags are less likely to contain positive instances. To investigate this, we repeat the line bag experiment, setting $\mathcal{D}_i = \mathcal{N}(0, \sigma)$ for each i and varying σ. Note that in this particular setup, $\sigma = 0$ implies $\tilde{p} = 1$, and as σ increases $\tilde{p}$ approaches 0 (as the bag becomes 'bigger'). Results for multiple values of m and σ are shown in Fig. 4; indeed increasing σ degrades performance while increasing m improves it. Note that in real applications we can control m, but the value of $\tilde{p}$ is fixed and dependent on the data.

4 Filtering Instances

In the previous sections we saw that sampling more instances for bags improves the performance of a trained classifier. We now present some strategies for reaping the benefits of sampling many instances, while maintaining efficiency.

When training a classifier, we would like to minimize the empirical error over bags. Substituting h for h^* in Eqn. 2, we can write this as:

$$h = \operatorname*{argmin}_h \sum_{i=1}^{n} \mathbf{1}[\max_{\alpha \in \mathcal{A}} \left(h(\mathcal{Q}(o_i, \alpha))\right) \neq Y_i] \tag{4}$$

This is a challenging optimization for a number of reasons. One of the key difficulties is that finding the maximum in terms of α may be intractable, especially if $\mathcal{A}$ is infinite. Moreover, for practical reasons we wish to deal with only limited amounts of data at a time.

Recently, Bradley and Schapire [6] introduced a boosting algorithm called FilterBoost which learns from a continuous source of data. Boosting is well adapted to this scenario because training happens in stages – weak classifiers are trained sequentially. FilterBoost alternates between training an additional weak classifier and querying the oracle for more data, using the the latest version of the overall classifier to evaluate the weights of queried instances.

MILBoost()

Input: $\{o_1, ., o_n\}$,$\{Y_1, ., Y_n\}$,$\{\mathcal{D}_1, ., \mathcal{D}_n\}$

for $t = 1$ to T **do**
  Call **FilterInstances**(i) for $i = 1 \ldots n$
  Compute weights $w_{ij} = \frac{\partial \mathcal{L}}{\partial H_t}(x_{ij})$
  Train weak classifier h_t
    $h_t = \text{argmax}_h \sum_{ij} (w_{ij} h(x_{ij}))$
  Find a_t via line search to maximize $\mathcal{L}$
    $a_t = \text{argmax}_a \mathcal{L}(H_{t-1} + a h_t)$
  Update strong classifier $H_t \leftarrow H_{t-1} + a_t h_t$
end for

FilterInstances()

Parameters: MEM, SRCH, m, ϵ, R, F

if $t > 1$ & $\text{mod}(t, F) \neq 0$ **then return**
if MEM & $t > 1$ **then** $r = m + 1$ **else** $r = 1$
$\alpha_{ij} = D_i(), x_{ij} = \mathcal{Q}(o_i, \alpha_{ij})$, for $j = r \ldots R$
Compute $p(x_{ij})$ for each j, keep top m instances
if SRCH **then**
  for $j = 1 : m$ **do**
    if $p(\mathcal{Q}(o_i, \alpha_{ij} + \epsilon)) > p(x_{ij})$ **then**
      $\alpha_{ij} = \alpha_{ij} + \epsilon, x_{ij} = \mathcal{Q}(o_i, \alpha_{ij})$
    end if
  end for
end if

Inspired by this work, we make an analogous extension of MIL for the case of query bags. In our case, however, we query the oracle for additional instances from a given bag i, rather than requesting additional labeled data (recall that for MIL labels are required only for bags). Therefore, as opposed to FilterBoost, the number of labels required does *not* depend on the number of queries to the oracle.

We focus on the MIL-BOOST algorithm, which was originally proposed in [23] and extended in [3]. We begin with a brief review of MIL-BOOST, and then describe our extensions to filter instances during training.

4.1 MIL-BOOST Review

MIL-BOOST trains a classifier of the form $H(x) = \sum_{t=1}^{T} a_t h_t(x)$, where $h_t : \mathcal{X} \rightarrow \{-1, 1\}$ is a weak classifier, such as a decision stump, and a_t is a scalar weight. Using Friedman's gradient boosting framework [14], we can train this classifier to optimize the log likelihood of the data (since 0/1 loss is generally difficult to optimize). For shorthand we define $p_i \equiv p(Y_i = 1|o_i)$, the probability that bag i is positive. We can write down the log likelihood (defined over bag, not instance, probabilities) as:

$$\mathcal{L}(H) = \sum_{i=1}^{n} [Y_i \log p_i + (1 - Y_i) \log(1 - p_i)] \tag{5}$$

We will model the probability of an instance x being positive as $p(x) = \sigma\big(H(x)\big)$, where $\sigma(v) = (1 + \exp\{-v\})^{-1}$ is the sigmoid function. What remains is to define the probability of a bag p_i as a function of its instances. In our model with query bags we could write this as:

$$p_i = \max_{\alpha \in \mathcal{A}} \{p(\mathcal{Q}(o_i, \alpha))\} \tag{6}$$

The above is analogous to Eqn. 2. However, as mentioned before, the $\max$ over α is difficult to deal with. If we subsample our bag to get m instances $\{x_{i1}, ., x_{im}\}$, we can re-write the above as $p_i^m = \max_{j \in \{1,\ldots,m\}} \{p(x_{ij})\}$. Furthermore, we replace the $\max$ with a differentiable approximation such as Noisy-OR ([3] proposes several options for this). To optimize Eqn. 5, we perform gradient descent in function space. The MIL-BOOST algorithm is summarized above; for details see [23, 3].

4.2 Filtering Strategies

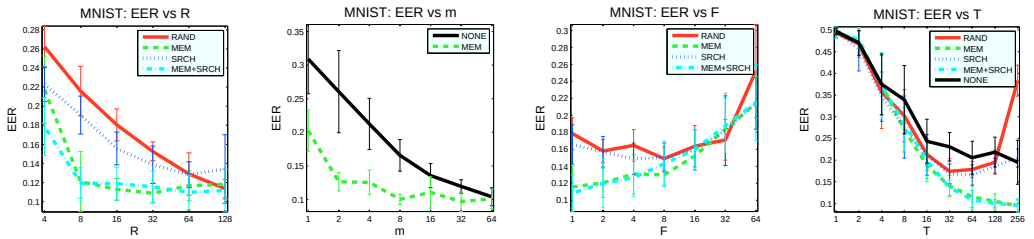
Previously, we observed that when data obeys the query bag model, having more instances per bag is advantageous. Since the number of candidate instances per bag can be quite large, even infinite, for practical reasons we are limited to a relatively small number of instances per bag for use during training. Instead of using a fixed set, we propose querying the data oracle for new instances during each boosting iteration. In each iteration, we assume we have the computational resources to train the weak classifiers with m instances per bag. Our goal is to optimize Eqn. 5. To get a good estimate of the likelihood, filtering is used to select m instances x_{ij} for each bag that have high probability $p(x_{ij})$ given the current classifier $H_t = \sum_k^t a_k h_k$. For negative bags this is similar to traditional bootstrapping – we want to select the hardest negative examples. For positive bags we want to get the most "correct" examples. Some cost is incurred for querying the oracle (*e.g.* cropping a patch out of an image) and evaluating $p(x_{ij})$. Assume we have the computational resources to evaluate the probability of R instances per bag and that we can filter instances once in every F iterations of boosting. Given these constraints, we propose the following filtering strategies.

Random Sampling (RAND): The simplest filtering strategy is to query $R > m$ samples and keep the m with the highest probability, resulting in $O(nR)$ classifier evaluations (n is the number of bags). Samples are queried using $\alpha_{ij} \sim \mathcal{D}_i$ followed by $x_{ij} = \mathcal{Q}(o_i, \alpha_{ij})$.

Memory (MEM): Instead of sampling a fresh batch of R instances per bag in each iteration, we can retain the m instances from the previous iteration, sample $(R-m)$ new instances, and then keep the best m of the combined set (resulting in $O(nR)$ evaluations as before). The classifier changes between iterations; nevertheless, memory allows high probability instances to accumulate over time. **Search** (SRCH): Recall that in many scenarios we expect that given $x_{ij} = \mathcal{Q}(o_i, \alpha_{ij})$ and $x'_{ij} = \mathcal{Q}(o_i, \alpha_{ij}+\epsilon)$, $\|x_{ij}-x'_{ij}\|_2$ is small for small $\epsilon \in \mathcal{A}$. Although H need not be smooth in the technical sense, it is likely that $\left|p(x'_{ij}) - p(x_{ij})\right|$ is also small for nearby instances. Thus, given a high probability instance x_{ij}, we can search for nearby x'_{ij} such that $p(x'_{ij}) > p(x_{ij})$. A straightforward way to operationalize this search is to test c nearby locations for each instance at offsets $\{\epsilon_1, ., \epsilon_c\} \in \mathcal{A}$, and keep the neighbor with the highest probability. Note that this incurs an additional cost of $O(nmc)$ classifier evaluations.

We summarize the three strategies (random sampling, memory, and search) in Algorithm 3. Memory (MEM) and search (SRCH) can be turned on/off, and we can adjust the amount (R) and filtering period (F). We emphasize that more sophisticated filtering strategies could be developed, *e.g.* a true gradient descent strategy to take advantage of underlying manifold structure of the instances as opposed to the steepest descent type search described above. However, our goal is to demonstrate that even simple filtering strategies can result in significant performance gains.

4.3 Filtering Experiments



(A) (B) (C) (D)

Figure 5: Plots showing performance for various filtering strategies (see text) for the MNIST dataset. In each plot we show the equal error rate (EER) plotted against four different parameters: **(A)** R, amount of sampling; **(B)** m, number of instance per bag during training (only MEM shown); **(C)** F, the filtering period (low F = frequent); and **(D)** T, the number of weak classifiers.

MNIST For this experiment we used the MNIST handwritten digit dataset described in Sec. 3.1. $\mathcal{A} = \mathbb{R}^2$ encodes the 2D pixel coordinates of the center of an image patch and $\mathcal{D}_i$ is set to uniform as we have no prior information as to where the digit may be located. For SRCH we used four values of $\epsilon = (\pm 1, 0), (0, \pm 1)$.

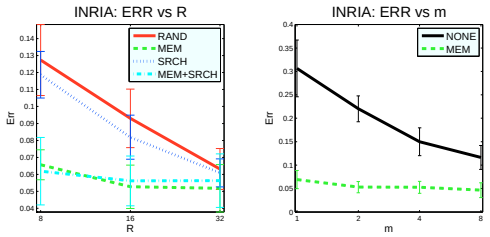
Our goal is to measure the effectiveness of various combinations of filtering strategies. We report the equal error rate (EER), (the point where false positive equal false negatives), repeating each experiment 10-25 times depending on the measured variance, and plotting the averaged results with standard deviation bars. We measured filtering performance with MEM on/off and SRCH on/off, while sweeping through the parameters R (sampling amount), m (bag size), F (filtering period) and T (number boosting iterations). In each experiment we keep 3 of the 4 parameters fixed at their default values of $16, 4, 1, 64$, respectively, and sweep through the fourth. Where appropriate, we also include performance without filtering (NONE). Results are shown in Fig. 5.

In Fig. 5(A) we show the effect of altering R, the number of instances queried per bag per iteration. Both MEM strategies converge to low error when $R = 8$, while the other strategies take up to $R = 128$. SRCH is beneficial for random sampling but makes a smaller difference when MEM is turned on. Fig. 5(B) shows the advantage of filtering (MEM) versus sampling bags and keeping them fixed during training. Both strategies converge to low error as m increases, however, with filtering an *eighth* of the instances ($m = 8$ *vs.* $m = 64$) are required during training. Fig. 5(C) shows in more detail the effects of filtering period F (lower values of F result in more frequent sampling). Frequent filtering is beneficial for all strategies, however the improvement is most significant for MEM strategies (see Sec. 4.4 for discussion).

In Fig. 5(D) we plot error *vs.* the number of boosting iterations T. These results are particularly interesting, as without MEM error actually *increases* for large values of T. This is true also for

training error (not shown). This behavior is counterintuitive as boosting is know to have excellent generalization, with error decreasing as T increases [13]. We observed that this behavior is not as severe given larger R (results not shown). Essentially, as the classifier becomes more refined its response becomes more peaked and random sampling with low R may not yield any high probability instances for a given bag, thus preventing the classifier from converging. Using memory alleviates this problem by guaranteeing high probability instances are retained (more details in Sec. 4.4).

INRIA Pedestrians The MNIST dataset is convenient because it is not particularly large or difficult and it allowed us to study the behavior of the algorithm in a well controlled manner. To ensure that the general trends of this behavior hold for a more challenging dataset, we repeat similar experiments with the INRIA pedestrian dataset [8]. This dataset is currently a standard benchmark for pedestrian detection, and is used to evaluate many recent state of the art systems (*e.g.* [12]). We use a setup analogous to the MNIST experiments – we resized these images to be 80×40 pixels in size, and then padded both the negative and positive images by 30 pixels in each direction by replicating the border. The number of possible instances for each bag is therefore 3721. Unlike the MNIST dataset, here it would be impossible to sample the bags exhaustively and store this in memory[2]. We use 500 positive and 500 negative bags for training. The default parameters used are as follows: $T = 128, F = 8, m = 4, R = 16$. Results are shown in Fig. 6. We see that the general trends we observed with the MNIST data appear in these results as well.



(A) (B)

Figure 6: Plots showing performance for various filtering strategies (see text) for the INRIA dataset. In each plot we show the miss rate at a false positive rate of 10% plotted against two parameters: **(A)** R, amount of sampling; **(B)** m, number of instance per bag during training.

4.4 Analysis

We now consider possible explanations for the results discussed above. Let $\mathcal{L}^m(H)$ be defined as in Eqn. 5, but with p_i^m replacing p_i. It is easy to show that as $m \to \infty$, $\mathcal{L}^m(H) \to \mathcal{L}(H)$ for a fixed classifier H. This follows from the fact that $p_i^m \to p_i$. This yet again suggests that training with more sampled instances is advantageous.

Now let us consider some of the filtering strategies proposed above, momentarily assuming the classifier H is fixed during training (obviously this is not the case in the algorithm). For RAND, we are drawing novel instances in each iteration, and there is no guarantee that at time step $t+1$ our estimate of the likelihood $\mathcal{L}^m(H)$ would be any closer to $\mathcal{L}(H)$ than at time t. With MEM, however, during every time step we are effectively evaluating an additional $(R-m)$ new instances per bag. Therefore, as $t \to \infty$, $\mathcal{L}^m(H) \to \mathcal{L}(H)$.

In practice, however, at every time step t the classifier H changes slightly. This makes it difficult to prove convergence for MIL-BOOST with filtering. In particular, note that for two different classifiers H and $\tilde{H}$, $\mathcal{L}^m(H) > \mathcal{L}^m(\tilde{H})$ does *not* imply $\mathcal{L}(H) > \mathcal{L}(\tilde{H})$. In other words, if the boosting algorithms finds a local maximum of $\mathcal{L}^m$, it is not necessarily the local maximum of the true log likelihood $\mathcal{L}$. However, the above analysis provides some intuition as to which filtering strategies should work, in particular, it helps explain why MEM is essential.

5 Conclusions

In this work we argued that the majority of MIL applications in recent literature have diverged from the assumptions and data generation models associated with the original MIL formulation. We presented the **query bag model**, which more accurately fits the data in these applications. We showed that sampling more instances for each bag is advantageous and proposed a number of filtering strategies for dealing with a large number of samples. These strategies open the door to effectively dealing with a range of MIL applications in computer vision, audition, text, bioinformatics, and other domains that previously required heavy sampling of bags and thus resulted in suboptimal performance. We envision developing more sophisticated techniques for specific domains thus extending the effectiveness and applicability of the MIL framework.

[2]Storing the 2500 dimensional feature vectors for this many instances would require over 60GB of memory.

References

[1] S. Andrews, T. Hofmann, and I. Tsochantaridis. Multiple instance learning with generalized support vector machines. *A.I.*, pages 943–944, 2002.

[2] P. Auer. On learning from multi-instance examples: Empirical evaluation of a theoretical approach. *ICML*, 1997.

[3] B. Babenko, P. Dollár, Z. Tu, and S. Belongie. Simultaneous learning and alignment: Multi-instance and multi-pose learning. In *Faces in Real-Life Images*, 2008.

[4] J. Bi, Y. Chen, and J. Wang. A sparse support vector machine approach to region-based image categorization. In *CVPR*, 2005.

[5] A. Blum and A. Kalai. A Note on Learning from Multiple-Instance Examples. *Mach. Learning*, 30(1):23–29, 1998.

[6] J. Bradley and R. Schapire. FilterBoost: Regression and Classification on Large Datasets. In *NIPS*, 2007.

[7] R. Bunescu and R. Mooney. Multiple instance learning for sparse positive bags. In *ICML*, 2007.

[8] N. Dalal and B. Triggs. Histograms of oriented gradients for human detection. In *CVPR*, 2005.

[9] T. Dietterich, A. Jain, R. Lathrop, and T. Lozano-Perez. A Comparison of Dynamic Reposing and Tangent Distance for Drug Activity Prediction. In *NIPS*, 1994.

[10] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Perez. Solving the multiple-instance problem with axis parallel rectangles. *A.I.*, 1997.

[11] D. Dooly, S. Goldman, and S. Kwek. Real-valued multiple-instance learning with queries. *Journal of Computer and System Sciences*, 72:1–15, 2006.

[12] P. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*, 2008.

[13] Y. Freund and R. Schapire. A decision-theoretic generalization of on-line learning and an application to boosting. *Journal of Comp. and Sys. Sci.*, 55:119–139, 1997.

[14] J. Friedman. Greedy function approximation: A gradient boosting machine. *Ann. of Stat.*, 29(5):1189–1232, 2001.

[15] J. D. Keeler, D. E. Rumelhart, and W. K. Leow. Integrated segmentation and recognition of hand-printed numerals. In *NIPS*, 1990.

[16] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[17] P. Long and L. Tan. PAC Learning Axis-aligned Rectangles with Respect to Product Distributions from Multiple-Instance Examples. *Mach. Learning*, 30(1):7–21, 1998.

[18] M. Mandel and D. Ellis. Multiple-instance learning for music information retrieval. In *ISMIR*, 2008.

[19] O. Maron and T. Lozano-Perez. A framework for multiple-instance learning. In *NIPS*, 1998.

[20] S. Ray and M. Craven. Supervised versus multiple instance learning: an empirical comparison. *ICML*, 2005.

[21] L. K. Saul, M. G. Rahim, and J. B. Allen. A statistical model for robust integration of narrowband cues in speech. *Computer Speech and Language*, 15:175–194, 2001.

[22] P. Simard, Y. LeCun, and D. J. Efficient pattern recognition using a new transformation distance. In *NIPS*, 1993.

[23] P. Viola, J. C. Platt, and C. Zhang. Multiple instance boosting for object detection. In *NIPS*, 2005.

[24] Q. Zhang and S. Goldman. EM-DD: An improved multiple-instance learning technique. In *NIPS*, volume 14, pages 1073–1080, 2002.