Multiple Instance Learning: Algorithms and Applications

Boris Babenko
Dept. of Computer Science and Engineering
University of California, San Diego

Abstract

Traditional supervised learning requires a training data set that consists of inputs and corresponding labels. In many applications, however, it is difficult or even impossible to accurately and consistently assign labels to inputs. A relatively new learning paradigm called Multiple Instance Learning allows the training of a classifier from ambiguously labeled data. This paradigm has been receiving much attention in the last several years, and has many useful applications in a number of domains (e.g. computer vision, computer audition, bioinformatics, text processing). In this report we review several representative algorithms that have been proposed to solve this problem. Furthermore, we discuss a number of existing and potential applications, and how well the currently available algorithms address the problems presented by these applications.

1 Introduction

In traditional supervised learning a training dataset, consisting of input and output/label pairs, is used to construct a classifier that can predict outputs/labels for novel inputs [1, 2]. An enormous amount of theoretical and practical work has gone into developing robust algorithms for this problem. However, the requirement of input/label pairs in the training data is surprisingly prohibitive in certain applications, and more recent research has focused on developing more flexible paradigms for learning. In this paper we focus on the Multiple Instance Learning (MIL) paradigm, which has been emerging as a useful tool in a number of application domains. In this paradigm the data is assumed to have some ambiguity in how the labels are assigned. In particular, rather than providing the learning algorithm with input/label pairs, labels are assigned to *sets* or *bags* of inputs. Restricting the labels to be binary, the MIL assumption is that every positive bag contains at least one positive input. The true input labels can be thought of as latent variables, because they are not known during training.

Consider the following simple example (adapted from [3]) of a MIL problem, illustrated in Fig. 1. There are several faculty members, and each owns a key chain that contains a few keys. You know that some of these faculty members are able to enter a certain room, and some aren't. The task is then to predict whether a certain key or a certain key chain can get you into this room. To solve this we need to find the key that all the "positive" key chains have in common. Note that

Figure 1: Keychain example: see text for details.

if we can correctly identify this key, we can also correctly classify an entire key chain - either it contains the required key, or it doesn't.

The spirit of MIL emerged as early as 1990 when Keeler *et al*. designed a neural network algorithm that would find the best segmentation of a hand written digit during training [4]. In other words, the digit label was assigned to every possible (block) segmentation of a digit image, rather than to a tightly cropped image. The idea was further developed, and the actual term was coined by Dietterich *et al*. in [3]. The problem that inspired this work was that of drug discovery, where some property of the molecule has to be predicted from its shape statistics. The ambiguity comes into play because each molecule can twist and bend into several different distinct shapes, and it is not known which shape is responsible for the particular property (see Figure 4). In Section 3 we will see many more interesting applications that fit into the MIL paradigm.

Although MIL has received an increasing amount of attention in recent years, the problem is still fairly undeveloped and there are many interesting open questions. The goal of this report is to provide a review of some representative algorithms for MIL, as well as some interesting applications. Finally, we discuss how well the existing algorithms address the problems presented by the applications and point out potential avenues for future research.

2 Algorithms

The purpose of this section is to formally define the MIL problem, and review some popular algorithms for solving MIL. For the purpose of clarity, it is useful to divide all existing methods into a hierarchy, but, as is usually the case, this is impossible to do as various methods overlap in various ways. As with supervised learning, the training procedure for MIL will require two basic ingredients: a cost function (*e.g*. 0/1 loss, likelihood), and a method of finding a classifier that optimizes that cost function (*e.g*. gradient descent, heuristic search). We chose to structure this review by splitting the methods up according to the former criteria, but it should be noted that there are methods with different cost functions that are actually very similar in terms of their optimization procedures. Next we define the MIL problem formally, review the most basic way of solving MIL as well as some PAC algorithm and theoretic results; finally, we conclude with a review of maximum

likelihood and maximum margin MIL algorithms.

2.1 Problem Statement

We begin by formally stating the problem, and establishing the notation that will be used in the rest of this section. Since it will be useful to compare MIL to standard supervised learning, and since most MIL algorithms are derived from well known supervised learning algorithms, we first establish notation for supervised learning. The training data consists of instance examples $\{x_1, x_2 \ldots, x_n\}$ and labels $\{y_1, y_2 \ldots, y_n\}$, where $x_i \in \mathcal{X}$ and $y_i \in \mathcal{Y}$. Typically, $\mathcal{X} = \mathbb{R}^d$ is the $d-$dimensional Euclidean space, and $\mathcal{Y} = \{0, 1\}$. The goal of supervised learning is to train a classifier function $h(X) : \mathcal{X} \rightarrow \mathcal{Y}$ that will accurately predict labels y for novel instances x. On the other hand, in Multiple Instance Learning the training set consists of bags $\{X_1, X_2 \ldots, X_n\}$ and bag labels $\{y_1, y_2 \ldots, y_n\}$, where $X_i = \{x_{i1}, x_{i2} \ldots, x_{im}\}$, $x_{ij} \in \mathcal{X}$ and $y_i \in \{0, 1\}$ (for convenience we also define $y' = (2y - 1) \in \{-1, +1\}$). Note that the MIL problem is defined only for binary cases (although extensions have also been explored [5, 6]). For the sake of simplicity, we will assume that $\mathcal{X} = \mathbb{R}^d$. Furthermore, we will assume that all bags are of a fixed size m, although in most algorithms this is not necessary.

The basic assumption of MIL, as mentioned before, is that a bag is positive if at least one of the instances in that bag is positive (the true positive instance inside a positive bag is sometimes referred to as the "witness" or the "key"). In other words, we assume that instance labels $y_{ij} \in \{0, 1\}$ exist for each instance, but they are not known during training. We can think of these as latent variables. The MIL assumption can be then summarized as follows:

$$y_i = \begin{cases} 1 & \text{if } \exists j \text{ s.t. } y_{ij} = 1 \\ 0 & \text{otherwise} \end{cases} \tag{1}$$

Another way of writing this is:

$$y_i = \max_j \{y_{ij}\} \tag{2}$$

This is a bit less intuitive, but will prove to be useful later.

The goal of MIL is to either train an instance classifier $h(X) : \mathcal{X} \rightarrow \mathcal{Y}$ or a bag classifier $H(X) : \mathcal{X}^m \rightarrow \mathcal{Y}$. However, note that a bag classifier can be easily constructed with the correct instance classifier: $H(X_i) = \max_j h(x_{ij})$. Therefore, with few exceptions [7, 8], most MIL algorithms aim to learn the instance classifier.

2.2 Naive MIL

Perhaps the simplest way of solving the MIL problem is to generate instance labels y_{ij} equal to the bag labels y_i. For negative bags, the generated instance labels will be correct, because all instances in a negative bag are negative. However, positive bags may contain both positive and negative instances, and this approach will inevitably lead us to assigning positive labels to negative instances.

$x_i \in \mathcal{X}$	i^{th} instance (supervised)
$X_i = \{x_{i1}, x_{i2} \ldots, x_{im}\}$	i^{th} bag with m instances x_{ij} (MIL)
$y_i \in \mathcal{Y}$	label of i^{th} instance (supervised) or i^{th} bag (MIL)
y_{ij}	true label of j^{th} instance in i^{th} bag
y'	$(2y - 1)$
n	instances (supervised) or number of bags (MIL)
m	number of instances per bag
d	dimensionality
$h(x)$	instance classifier
$H(X)$	bag classifier

Table 1: A summary of notation.

Still the question remains: can a standard supervised learning algorithm perform comparably to a MIL algorithm? Many papers have investigated this question, and have shown for a variety of datasets and algorithms (both supervised and MIL) that using the MIL framework results in better performance [3, 9, 10, 11]. MIL algorithms tend to outperform this "naive MIL" approach in practice by a large margin. Nevertheless, this is an important scenario to consider, and we will see that an approach similar to this leads to interesting theoretical results.

2.3 PAC Algorithms and Bounds

Early work on MIL focused on simple algorithms, such as axis aligned rectangles, for which theoretical results can be derived more readily [12, 13]. In particular, several researchers focused on developing Probably Approximately Correct (PAC) [14] learning algorithms, and showing sample complexity bounds. The basic idea behind PAC is to prove that given some training data, a certain learning algorithm will produce an accurate classifier with high probability. Formally, for small ϵ and δ, $\mathbb{P}[\mathbf{err}(h) > \epsilon] < \delta$, where **err**$(h)$ is the error of classifier h on unseen data. Using this framework one can show roughly how much data will be needed to achieve an error of ϵ with high probability – this is often referred to as the sample complexity. To arrive at these results, these algorithms generally need to make some assumptions about the data. For the case of supervised learning, the most common assumption is that the instances are drawn i.i.d. from some distribution. For MIL there is a similar assumption: we assume that all instances are drawn i.i.d. from some distribution, and then placed randomly into bags. This means that the instances in one bag have no special relationship. This is a very important assumption, and we will discuss when it does and does not apply in Section 4.

Blum [15] proposed a surprisingly simple algorithm that reveals an interesting relationship between supervised learning and MIL. The basic idea of his algorithm is as follows: for negative bags, assign a negative instance label to all instances; for positive bags, choose one instance per bag at random, and assign a positive instance label. Finally, plug these instance/label pairs into

a noise tolerant learning algorithm [16]. This is similar to the naive MIL approach we discussed in the previous section. The key to this algorithm is that the positive instances will always have a correct label. However, the negative instances may be labeled positive (if they came from a positive bag). Therefore, the data set that we produce will have asymmetric label noise. With this reduction, Blum shows that any concept class that is PAC learnable with noise is also PAC learnable for MIL. The best sample complexity bound is also due to Blum. This algorithm is a more sophisticated extension of the above, and uses the Statistical Query framework [16] (that was used to prove PAC bounds for noise-tolerant learning by Kearns *et al.*). The sample complexity bound of this MIL algorithm is $\tilde{O}(d^2 m/\epsilon^2)$. Note that this implies that learning becomes more difficult as ϵ decreases, as the dimensionality d increases, and as the bag size m increases.

Although Blum's algorithm achieved the best known PAC bounds for MIL, it is clearly not a practical algorithm. For large bag sizes m we would essentially be throwing away vast amounts of data. However, this algorithm is interesting to study because it highlights the relationship between supervised learning and MIL. The algorithms we review in the sections below generally have fewer known theoretical properties, but are more practical and have been shown to perform well on difficult data.

2.4 Maximum Likelihood

The first category of practical MIL algorithms that we will discuss aims to train a classifier that maximizes the likelihood of the data. Maximum likelihood methods are popular in supervised learning, and we begin with a breis review. For each instance x_i and a classifier h we let $p_i \equiv \mathbb{P}(y_i = 1|x_i, h)$ be the posterior probability of that instance. The exact definition of this entity will depend on the classifier. The log likelihood[1] is defined as follows:

$$\mathcal{L} = \sum_{i|y_i=1} \log(p_i) + \sum_{i|y_i=0} \log(1 - p_i) \tag{3}$$

To adapt this loss function for MIL we need to remove the dependence on instance labels, as these are not known during training. Instead, we let $p_i \equiv \mathbb{P}(y_i = 1|X_i, h)$ be the posterior probability of the bag, and $p_{ij} \equiv \mathbb{P}(y_{ij} = 1|x_{ij}, h)$ be the posterior probability of an instance. The equation for log likelihood remains the same, but note that p_i is now the bag probability. As before, the exact definition of p_{ij} will depend on the classifier.

Finally, we need to connect the bag probability p_i to the probabilities of its instances p_{ij}. A natural way of defining this relationship is:

$$p_i = \max_j \{p_{ij}\} \tag{4}$$

The above can be considered a probabilistic equivalent of Eqn. 2. An important observation is that the $\max$ operator is not differentiable, which means that the log likelihood for MIL is not

[1] Maximizing the likelihood can cause numerical errors, so it is common to instead maximize the log likelihood, which is equivalent because $\log$ is monotonically increasing.

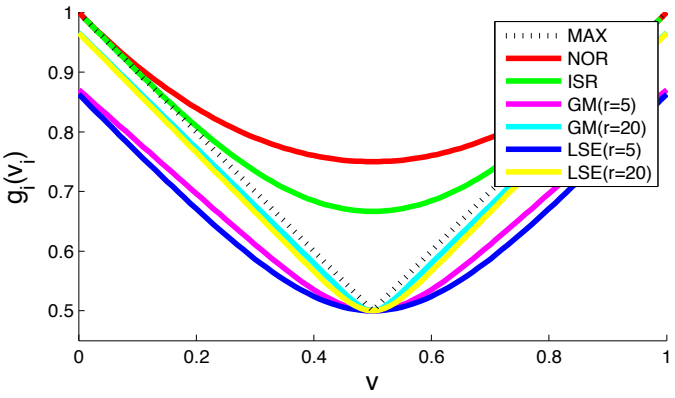

Figure 2: Various models $g_\ell(v_\ell)$ applied to $v_\ell = (v, 1-v)$.

differentiable. This presents a considerable problem as most maximum likelihood methods in supervised learning take advantage of the smoothness of the loss function to perform some form of gradient descent. Luckily, several differentiable approximations to the $\max$ operator exist in the literature. We continue with a review of these approximations.

2.4.1 Differentiable Max Approximations

We now present an overview of differentiable approximations of the *max*, which we refer to as the *softmax*. The general idea is to approximate the max over $\{v_1, v_2 \ldots, v_m\}$ by a differentiable function $g_\ell(v_\ell)$, such that:

$$g_\ell(v_\ell) \approx \max_\ell(v_\ell) = v_* \tag{5}$$

$$\frac{\partial g_\ell(v_\ell)}{\partial v_i} \approx \frac{\mathbf{1}(v_i = v_*)}{\sum_\ell \mathbf{1}(v_\ell = v_*)} \tag{6}$$

Intuitively, if v_i is the unique max, then changing v_i changes the max by the same amount, otherwise changing v_i does not affect the max.

A number of approximations for g have been proposed. We summarize the choices used here in Table 2: a variant of log-sum-exponential (LSE) [17, 18], generalized mean (GM), noisy-or (NOR) [10], and the ISR model [4, 10]. For experimental comparisons of these different choices within MIL we refer the reader to [19]. In Fig. 2 we show the different models applied to $(v, 1-v)$ for $v \in [0, 1]$.

LSE and GM each have a parameter r the controls their sharpness and accuracy; $g_\ell(v_\ell) \to v_*$ as $r \to \infty$ (note that large r can lead to numerical instability). For LSE one can show that $v_* - \log(m)/r \leq g_\ell(v_\ell) \leq v_*$ [17] and for GM that $(1/m)^{1/r} v_* \leq g_\ell(v_\ell) \leq v_*$, where $m = |v_\ell|$. NOR and ISR are only defined over $[0, 1]$. Both have probabilistic interpretations and work well in

practice; however, these models are best suited for small m as $g_\ell(v_\ell) \to 1$ as $m \to \infty$. Finally, all models are exact for $m = 1$, and if $\forall \ell \; v_\ell \in [0, 1]$, then $0 \leq g_\ell(v_\ell) \leq 1$ for all models.

	$g_\ell(v_\ell)$	$\partial g_\ell(v_\ell)/\partial v_i$	domain
LSE	$\frac{1}{r} \ln \left(\frac{1}{m} \sum_\ell \exp(r v_\ell) \right)$	$\frac{\exp(r v_i)}{\sum_\ell \exp(r v_\ell)}$	$[-\infty, \infty]$
GM	$\left(\frac{1}{m} \sum_\ell v_\ell^r \right)^{\frac{1}{r}}$	$g_\ell(v_\ell) \frac{v_i^{r-1}}{\sum_\ell v_\ell^r}$	$[0, \infty]$
NOR	$1 - \prod_\ell (1 - v_\ell)$	$\frac{1 - g_\ell(v_\ell)}{1 - v_i}$	$[0, 1]$
ISR	$\frac{\sum_\ell v'_\ell}{1 + \sum_\ell v'_\ell}$, $v'_\ell = \frac{v_\ell}{1 - v_\ell}$	$\left(\frac{1 - g_\ell(v_\ell)}{1 - v_i} \right)^2$	$[0, 1]$

Table 2: Four softmax approximations $g_\ell(v_\ell) \approx \max_\ell(v_\ell)$.

Now we are ready to go over a few specific maximum likelihood algorithms for MIL. For each, we will define the classifier h and the instance probability p_{ij} in terms of h.

2.4.2 Logistic Regression

Logistic regression [2] is a popular probabilistic model for supervised learning, and using the framework we have laid out so far it is fairly straightforward to adapt it to MIL [9]. The classifier will consist of just one vector in the input space: $h = \{w\}, w \in \mathbb{R}^d$. Using the sigmoid function:

$$\sigma(x) = \frac{1}{1 + \exp\{-x\}} \tag{7}$$

we define the instance probabilities as follows:

$$p_{ij} = \sigma\Big(w \cdot x_{ij} \Big) \tag{8}$$

We can think of the above as a probabilistic version of linear regression. Note that the range of the sigmoid is between 0 and 1, so plugging the dot product $(w \cdot x_{ij})$ into this function produces a valid probability value. Another interpretation of logistic regression is that w is a normal vector to a separating hyperplane, and the probability of an instance depends on the distance between it and the hyperplane. While in standard logistic regression we would directly plug these probabilities into the likelihood cost function, for MIL we must first compute the bag probabilities p_i and then plug the resulting values into the likelihood. If we use one of the max approximations discussed above, this likelihood will be differentiable with respect to w. We can therefore use a standard unconstrained optimization procedure (*e.g.* gradient descent) to find a good value for w. A similar, though more sophisticated, model was proposed in [20].

2.4.3 Diverse Density

Maron *et al*. [21] proposed an algorithm called Diverse Density for MIL. Unlike many of the other MIL algorithm, this method does not have a close relative in supervised learning (though it is reminiscent of nearest neighbor). The classifier again consists of just one vector in the input space: $h = \{w\}, w \in \mathbb{R}^d$, which we will call the target point. Rather than defining a hyperplane, however, this vector specifies a point in the input space that is "most positive". The probability of an instance x_{ij} depends on the distance between it and this target point:

$$p_{ij} = \exp\{-||x_{ij} - w||^2\} \tag{9}$$

In other words, the positive region in the space is defined by a Gaussian centered at the point w. Having defined this, we can again plug the instance probabilities into one of the max approximations, and then into the likelihood ([21] uses the NOR model). Gradient descent can be used to find an optimal w, and Maron *et al*. suggest to use points in the positive bags to initialize (because presumably the vector w will lie close to at least one instance in every positive bag). Several extensions can be incorporated into this framework. First, we can easily include a weighting of the features, rather than using basic Euclidean distance:

$$p_{ij} = \exp\{-\sum_k s_k(x_{ijk} - w_k)^2\} \tag{10}$$

It is then straight forward to modify the gradient descent to find both w and s. Furthermore, rather than having just one target point w, it is also straight forward to modify this algorithm to find several target points. The instance probabilities can then be calculated by taking a max over all the target points. This makes the classifier more powerful, and enables it to learn positive classes that are more sophisticated (essentially a mixture of Gaussians).

The EM-DD [22] algorithm uses the same cost function as the above, but the optimization technique is different. In this algorithm, the authors chose not to use a soft approximation of $\max$. As we mentioned before, this leads to a non-differentiable likelihood function. Zhang *et al*. therefore propose the following simple heuristic algorithm to find the optimal target point w. In the spirit of EM, the algorithm iterates over two steps: (1) use the current estimate of the classifier to pick the most probable point from each positive bag, and (2) find a new target point w by maximizing likelihood over all negative instances and the positive instances from the previous step (can use any gradient descent variant to do this since the $\max$ is no longer a problem). In other words, step (2) reduces to standard supervised learning, rather than MIL. We will see a similar search heuristic used in one of the maximum margin algorithms in Section 2.5

2.4.4 Boosting

Boosting [23] is a popular, powerful tool in statistical machine learning. Many different boosting algorithms for standard supervised learning exist in the literature, and these algorithms have been shown to be very effective in many applications. The basic idea behind boosting is to take a simple

learning algorithm, train several simple classifiers, and combine them. Typically, the combination is done via weighted sum, the classifier is defined as: $h(x) = \text{sign}\big(\sum_{t=1}^{T} \alpha_t h_t(x)\big)$, where $h_t \in \mathcal{H}$ is a "weak classifier" (sometimes called the "base classifier"), and α_t are the positive weights.

Viola *et al*. proposed a boosting algorithm for solving MIL in [10]. They define the probability of an instance as

$$p_{ij} = \sigma\Big(\sum_{t} \alpha_t h_t(x_{ij})\Big) \tag{11}$$

This is similar to posterior probability estimates for AdaBoost and LogitBoost [24]. They then use the NOR and ISR max approximations to define the bag probabilities p_i. Using the gradient boosting framework [25, 26], they derive an boosting procedure that optimizes the bag likelihood. Gradient boosting works by training one weak classifier at a time, and treating each of them as a gradient descent step in function space. To do this, we first compute the gradient $\partial\mathcal{L}/\partial h$ and then find a weak classifier h_t that is as close as possible to this gradient (note that it may be impossible to move exactly in the direction of the gradient because we are limited in our choice of h_t). The weight α_t can then be found via line search.

2.4.5 Other Gradient Descent algorithms

Other gradient descent based MIL algorithms can be framed in a manner similar to the above, by picking various differentiable loss functions for supervised learning, and plugging in the max approximation to compute the loss of a bag. For example, in [18] a neural network algorithm is derived by taking the squared loss function that is generally used in neural nets for supervised learning, and plugging in the LSE max approximation from above. The neural network can then be trained using a variation of the back propagation algorithm.

2.5 Maximum Margin

Due the success of the Support Vector Machine (SVM) algorithm [1], and the various positive theoretical results behind it, maximum margin methods have become extremely popular in machine learning. It is no surprise then that many variations of SVM have been proposed for the MIL problem. In this section we review a few of the most representative algorithms. We begin with a brief review of soft margin SVM.

2.5.1 Review of max margin methods

The basic idea behind SVM is to find a hyperplane in the input space[2] that separates the training data points with as big a margin as possible. The classifier is defined by the hyperplane normal w and the offset b, $h = \{w, b\}$. The margin, γ, is defined as the smallest distance between a positive

[2]For certain formulations of SVM the "kernel trick" can be employed to first map the input instances into a high dimensional feature space.

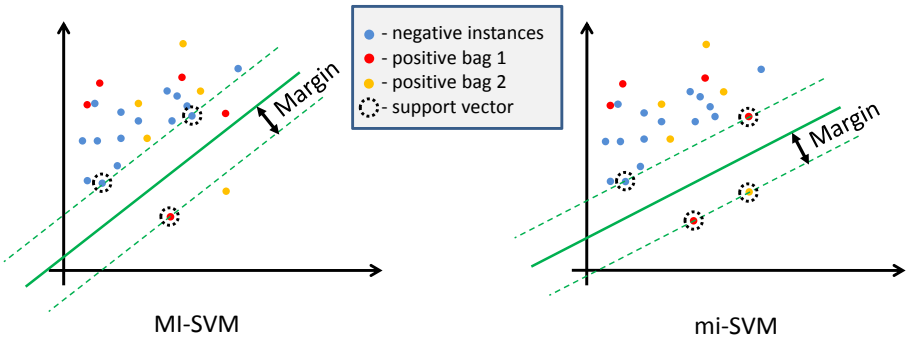

Figure 3: Comparing the solutions of **MI-SVM** and **mi-SVM** for a simple 2D dataset.

or negative point and this hyperplane. Because the data can be scaled arbitrarily, it can be shown that the following optimization is equivalent to maximizing the margin:

$$\begin{aligned} \min_{w,b,\xi} \quad & \tfrac{1}{2}||w||_2^2 + C\sum_i \xi_i \\ \text{s.t.} \quad & y_i(w \cdot x_i + b) \geq 1 - \xi_i \end{aligned} \tag{12}$$

The above can be interpreted as follows: we assume that the margin is at least 1, and shrink the size of the hyperplane normal w. The latter can also be seen as a form of regularization. Because the data may actually not be separable, we also include slack variables ξ_i for each point x_i. The points that are closest to the hyperplane (the ones that lie γ away) are called support vectors.

2.5.2 SVM for MIL

The main question we need to consider when adapting SVM to the MIL problem is how to define the margin. For negative bags, this is simple - all instances in negative bags are known to be negative, so the margin is defined as in the regular case. For positive bags, however, the issue is more complicated. Andrews *et al*. pioneered the work on max margin algorithms for MIL [27, 28]. In this work they propose two different ways of defining the margin, and present two programs that solve for the max margin classifier. In both cases the optimization turns out to be a mixed integer program, so Andrews *et al*. propose simple heuristic algorithms to do this optimization.

Bag margin: identifying the witness

The first type of margin is what we will refer to as the bag margin. As mentioned earlier, for negative instances, the margin is defined as in regular SVM. For positive instances we define the margin of a bag as the maximum distance between the hyperplane and all of its instances. Since we want at least one instance in a positive bag to be positive, we want at least one instance in a positive bag to have a large positive margin. In other words, the maximum over all the instance margins in

the bag must be bigger than one. Incorporating slack variables, we arrive at the following program:

$$\begin{aligned} \min_{w,b,\xi} \quad & \tfrac{1}{2}||w||_2^2 + C\textstyle\sum_{ij}\xi_{ij} \\ \text{s.t.} \quad & (w \cdot x_{ij} + b) \leq -1 + \xi_{ij}, \forall i | y_i = 0 \\ & \max_j (w \cdot x_{ij} + b) \geq 1 - \xi_{ij}, \forall i | y_i = 1 \\ & \xi_{ij} \geq 0 \end{aligned} \tag{13}$$

The second constraint in this optimization is not convex, which makes this a difficult problem to solve. By introducing an extra variable $s(i)$ for each bag, we can convert the above program into a mixed integer program:

$$\begin{aligned} \min_{s(i)} \min_{w,b,\xi} \quad & \tfrac{1}{2}||w||_2^2 + C\textstyle\sum_{ij}\xi_{ij} \\ \text{s.t.} \quad & (w \cdot x_{ij} + b) \leq -1 + \xi_{ij}, \forall i | y_i = 0 \\ & (w \cdot x_{is(i)} + b) \geq 1 - \xi_{ij}, \forall i | y_i = 1 \\ & \xi_{ij} \geq 0 \end{aligned} \tag{14}$$

The intuition of this program is that the variables $s(i)$ are trying to locate the witness instance in each bag. Note that the constraints involve just one instance per positive bag. This means that the potentially negative instance in a positive bag will be ignored. This version of SVM for MIL is called **MI-SVM**.

Mixed integer programs are difficult to solve; therefore, Andrews *et al*. propose a simple heuristic algorithm for solving the above program. The heuristic has the flavor of an EM algorithm: first the values of $s(i)$ are guessed using the current classifier (by choosing the instance in every positive bag with the largest margin), and then the SVM classifier is trained just as in regular supervised learning. These two steps are repeated until convergence, when the values of $s(i)$ stop changing. Note that this optimization procedure is very similar to the one used in EM-DD, although the objective function is different.

Instance margin: identifying the instance labels

In the above formulation, we were trying to recover the latent variable $s(i)$ that identifies the key instance in every positive bag. As mentioned previously, this approach ignores the negative instances in the positive bags. On the other hand, we could attempt to recover the latent variable y_{ij}, which is the label of every instance. Again, for negative bags this is known to always be 0. Once these variables are known, we could train a regular SVM using these instances labels. The

program is summarized below:

$$\begin{aligned} \min_{y_{ij}} \min_{w,b,\xi} \quad & \tfrac{1}{2}||w||_2^2 + C\sum_{ij}\xi_{ij} \\ \text{s.t.} \quad & y_{ij} = 0, \forall i | y_i = 0 \\ & \textstyle\sum_j y_{ij} \geq 1, \forall i | y_i = 1 \\ & y'_{ij}(w \cdot x_{ij} + b) \geq 1 - \xi_{ij} \\ & \xi_{ij} \geq 0 \end{aligned} \tag{15}$$

This variation of SVM for MIL is called **mi-SVM**. This is also a mixed integer program, and is also difficult to solve. As before, Andrews *et al*. propose a heuristic algorithm that is similar to the one we described for **MI-SVM**. The algorithm again iterates over two steps: use the current classifier to compute instance labels y_{ij}, then use these to train a standard SVM. If for a positive bag, none of the instance labels come out as positive, the instance with the largest value of $(w \cdot x_{ij} + b)$ is labeled positive.

Figure 3 shows a simple example that illustrates the differences between **MI-SVM** and **mi-SVM**. The important difference is that in the former approach, the negative instance in the positive bags are essentially ignored, and only one instance in each positive bag contributes to the optimization of the hyperplane; in the latter, the negative instances in the positive bags, as well as multiple positive instances from one bag can be support vectors.

2.5.3 Sparse MIL

As mentioned previously, both the **mi-SVM** and **MI-SVM** involve mixed integer programs which are difficult to solve. Instead, Bunescu *et al*. propose a third alternative for adapting an SVM to the MIL problem [11]. We begin with the program in Equation 13. The first constraint is simple and does not present any problems; the second constraint is the one that causes problems. Our goal is then to replace this constraint with something more manageable.

First, we add the following constraint for all positive bags: $|wx_{ij} + b| \geq 1$. This ensures that every instance in the bag has a large margin, whether it be negative or positive. Now we would like to enforce the fact that every positive bag should have at least one positive instance. Consider the following constraint: $\sum_j (wx_{ij} + b) \geq 1 + -1(m-1) = (2-m)$. If all instances in a bag are negative, then this constraint would be false. Therefore, this constraint ensures that at least one instance in the bag will be positive. One minor issue is that even if one instance in the bag is positive, the constraint might be violated if the margins of the negative points have a large magnitude. In this sense, this constraint is pulling all the negative instances in the bag closer to the hyperplane. However, the first constraint we added ensures that the margin for the negative points

is large enough. Adding the slack variables, we arrive at the following program:

$$\begin{aligned} \min_{w,b,\xi} \quad & \tfrac{1}{2}||w||_2^2 + C\sum_{ij}\xi_{ij} \\ \text{s.t.} \quad & (w \cdot x_{ij} + b) \leq -1 + \xi_{ij}, \forall i | y_i = 0 \\ & \textstyle\sum_j (w \cdot x_{ij}) \geq (2 - m), \forall i | y_i = 1 \\ & |w \cdot x_{ij} + b| \geq 1 \\ & \xi_{ij} \geq 0 \end{aligned} \tag{16}$$

Although this program is not convex, Bunescu *et al*. show how to solve it with a Concave-Convex solver (CCCP), which gives an exact solution unlike a heuristic algorithm. Furthermore, they show that this algorithm works particularly well on problems with sparse positive bags (where there are few positives in every positive bag).

2.5.4 Combining DD and 1-norm SVM

An interesting combination of Diverse Density and SVM was developed in [29, 30]. Recall that the DD classifier consists of a "target" point w, and instances are considered positive if they are close to this point. Bi *et al*. propose to embed all bags in a new feature space, and then train a regular SVM classifier. The intuition behind this feature space is closely linked to the idea behind DD.

Let $x^1, x^2 \ldots x^t$ be the all the instances in positive bags (*e.g.* $x_{ij}, \forall i | y_i = 1$). For each bag we construct the following vector:

$$d_i = \begin{bmatrix} \min_j m(x^1, x_{ij}) \\ \min_j m(x^2, x_{ij}) \\ \vdots \\ \min_j m(x^t, x_{ij}) \end{bmatrix} \tag{17}$$

In other words, in the new feature space, there will be a dimension for every x_{ij} in positive bags, and the value of this dimension will the distance to the closest bag member. Suppose that one of these x^t instances was the target point w (or was very close to the target point). That means that for positive bags, the value of the t^{th} dimension in the new feature space will be small, because there will be at least one point close to that target point. On the other hand, for negative bags, the value will be large. This intuition leads us to believe that this space will be discriminative.

The authors then propose to use a 1-norm SVM instead of the usual L_2 based SVM objective function. This is because the 1-norm SVM often results in a vector w that is more sparse. This is particularly attractive in this scenario because every dimension of w corresponds to an instance from the positive bags. First off the dimensionality will be quite large, so a sparse solution is attractive. In addition, a sparse solution will essentially "pick out" the target point(s).

$$\begin{aligned} \min_{w,b,\xi} \quad & ||w||_1 + C\sum_i \xi_i \\ \text{s.t.} \quad & y_i'(w \cdot d_i + b) \geq 1 - \xi_i \end{aligned} \tag{18}$$

An important note about this method is that it builds a *bag classifier* rather than an *instance classifier*. However, due to the way this problem is structured, we could easily classify an instance by pretending it is a bag of size 1.

2.6 Other approaches

Recall that in MIL, there is a simple relationship between a bag classifier and an instance classifier: $H(X_i) = \max_j h(x_{ij})$. As mentioned before, this relationship allows us to classify bags using an instance classifier. In some applications, however, only a bag classifier is needed, and in some cases it can be argued that training a bag classifier directly, rather than training an instance classifier and using it to classify bags, is an easier problem. A couple of methods have been proposed for this [8, 7], but they generally disregard the MIL assumption that a positive bag contains at least one positive instance, and instead treat the problem as set classification.

3 Applications

In this section we will review a few interesting applications of Multiple Instance Learning. More importantly, we will highlight the fundamental differences in the datasets these applications involve. In particular, there appear to be two distinct types of applications for MIL. We borrow the terminology of [31] and call these two types "polymorphism ambiguity" and "part-whole ambiguity". In the former, an object can have multiple different appearances in input space and it is not known which of these is responsible for the output/label; in the latter, an object can be broken into several pieces/parts, each of which has a representation in input space, and labels are assigned to the whole objects rather than the parts, though only one part is responsible for the label. We continue with a few concrete examples, and in Section 4 we discuss how the differences between these types of applications affect algorithm design.

3.1 Polymorphism Ambiguity

The original application for MIL that was proposed by Dietterich *et al*. [3] is a case of polymorphism ambiguity. In this application, we wish to classify molecules by looking at their shape. Each molecule can twist and bend, and can therefore appear in several distinct shapes (see Fig. 4). It would require significantly more work for a chemist to figure out which shape is responsible for the label (which specifies a certain behavior) of the molecule. Instead, we can collect the measurements for all the different shapes of a molecule, and assign a label to the entire bag of shapes. Dietterich *et al*. also introduced a dataset, that is commonly used to this day, called MUSK. This dataset consists of labels indicating whether a molecule produces a "musky" smell. If successful, these types of molecule classification methods can be of great use in drug discovery.

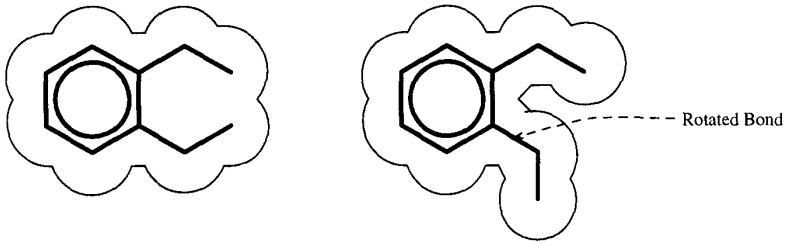

Figure 4: **Polymorphism Ambiguity.** This is a figure from [3] that we reproduce for convenience; it shows an example of one molecule taking on different shapes. Presumably, only one of these two shapes would be responsible for certain behaviors of this molecule.

3.2 Part-whole Ambiguity

The part-whole ambiguity datasets commonly arise in the computer vision and audition domains. In these areas it is common to have applications where labeling an entire image or sound clip is much easier than precisely cropping out an object of interest (*e.g*. a face in an image, a word in a sound clip).

The first example we will go over is object detection. Current methods in computer vision for object detection often train a classifier using a set of carefully cropped training images. At run time, the classifier is applied to every sub-window of a larger image, and non-maximal suppression is usually applied for post processing. Generating these carefully cropped training images, however, can be burdensome. Furthermore, it is difficult for a human labeler to be consistent with the cropping. For this reason, Viola *et al*. argue that placing bounding boxes around objects is an inherently ambiguous task. Instead they propose to gather training data into bags: for each image that is known to contain the object of interest, several bounding boxes (potentially of different scales and orientation) are cropped to generate a bag for MIL. Note that now only image labels are required, rather than precise bounding boxes. An example is shown in Figure 5 (BOTTOM).

A closely related approach in computer vision is to use image segmentation instead of densely sampling sub-windows of an image. As before, labels are given to images even though the labels apply only to one region of the image. For both testing and training the image is broken into several coherent pieces using an image segmentation algorithm. This is useful for cases when the object of interest cannot be easily inscribed in a box (*e.g*. objects that have articulation like humans, animals, etc). Some image statistics are computing for each segment, and the segments of one image constitute one bag for MIL. This type of approach has been explored in [27, 29, 30]. An example is shown in Figure 5 (TOP).

Computer audition is analogous to computer vision in many ways, except the data is 1 dimensional, rather than 2 dimensional. A sound clip can be split into a bag either by using a sliding window in the temporal domain, or by splitting it in the frequency domain such that each instance contains features that describe only a limited range of frequencies[20]. The former is analogous to cropping out object boundaries in computer vision, and the latter can help with noisy audio data.

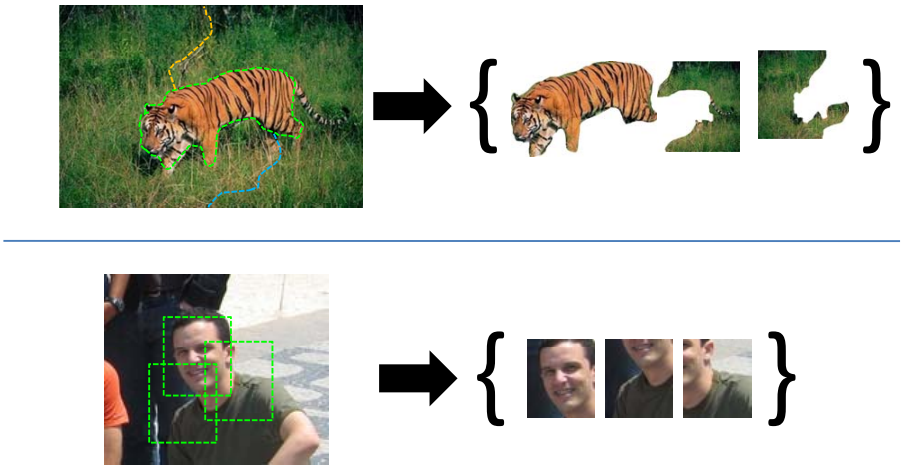

Figure 5: **Part/whole Ambiguity.** These are two examples of a bag being generated by splitting an image into regions with *(TOP)* image segmentation, and *(BOTTOM)* sliding windows. Note that a label that would be assigned to the entire image (*(TOP)* tiger and *(BOTTOM)* face) is attributed to just one of the smaller regions.

Another set of applications comes from Bioinformatics, where biological sequences (either DNA base pairs, or amino acids) need to be classified. A common approach is to use a sliding window across the sequence to form a bag of overlapping sub sequences [32, 9]. The assumption is that only one of the subsequences is responsible for the behavior of the entire DNA sequence or protein.

Finally, document/text classification has been investigated within a MIL framework [27]. A document is general split into paragraphs, or into overlapping pieces consisting of several paragraphs. The features computed for each piece of the document are presumably less noisy than features computed over the entire document; this can lead to more precise classification.

4 Discussion & Future Work

An obvious question to ask is which MIL algorithm is the best. As is the case with supervised learning, the performance of various algorithms depends on the datasets, and no absolute "best" algorithm exists. Unfortunately there have been few comprehensive studies comparing a wide range of algorithms. [9] is one such study, and it concludes that the performance of different algorithms varies depending on the data. Although the MUSK dataset is commonly used to evaluate MIL algorithms, it is difficult to use this dataset to draw conclusions about algorithms because it contains a relatively small amount of data (note that it was introduced in 1997 [3]).

Perhaps the most interesting observation is that there indeed appear to be two distinct classes of applications for MIL (polymorphism and part/whole ambiguities), and yet the algorithm develop-

ment has been independent of this distinction. The datasets belonging to these two different types of ambiguities may behave in very different ways, and it would be interesting to see how these behaviors can be used to our advantage.

One important difference in behavior between these two ambiguities is the effect of bag size, m. Recall from Section 2.3 that the best known sample complexity bound for MIL is $\tilde{O}(d^2 m/\epsilon^2)$ [15]. This implies that increasing the bag size m actually making the problem harder. Of course this bound would only hold if the data assumptions made by this PAC bound hold. In particular, there are two important assumptions made: (1) the instances in a bag are drawn independently, and (2) each positive bag contains at least one positive instances. In the case of polymorphism ambiguity, these seem to be a reasonable assumptions, and the implied behavior tends to be consistent with reality. For example, when dealing with molecules that have a huge number of possible shapes, finding the one that matters becomes difficult. Note that in this scenario the user does not have any control over the bag size - it depends purely on the data and is fixed. Furthermore, if a molecule is labeled positive, then the bag definitely contains a positive instance.

For part/whole ambiguity, however, this assumption does not hold and the data may behave in a completely opposite manner! Consider the case of object detection with sliding windows generating a bag. Sampling an image densely with sliding windows gives us a higher chance of cropping out the "perfect" bounding box around the face in an image. Therefore, in this application, a larger bag may actually make learning easier. Note that in this type of application, the instances are not at all independent. The other key difference is that here we are actually not guaranteed to have a positive instance in a positive bag. For example, consider the case of using image segmentation to generate a bag - what if the segmentation performs poorly and returns nonsense segments? Note that in this scenario the user *does* have control over the bag size, as there is usually way of determining how many *parts* the *whole* object is broken into to generate a bag. The only way to ensure that the MIL assumption holds in this application is to generate bags exhaustively: take every possible segment in an image, crop out every possible bounding box, *etc*. Of course, this would be prohibitive in terms of resources as the amount of data explodes – in some cases the number of *parts* for one object is actually infinite.

In the future, it would be interesting to develop algorithms that take these observations into account. In particular, an algorithm specifically tailored to the part/whole ambiguity datasets would be useful in many vision and audition applications. Furthermore, the theoretical properties of these types of datasets should be explored by making more appropriate assumptions.

Acknowledgements

The author would like to thank Serge Belongie, Sanjoy Dasgupta and Lawrence Saul for their feedback.

References

[1] Vapnik, V.: The Nature of Statistical Learning Theory. Springer-Verlag (1995)

[2] Bishop, C.: Pattern Recognition and Machine Learning (Information Science and Statistics), Secaucus. NJ, USA. Springer, Heidelberg (2006)

[3] Dietterich, T.G., Lathrop, R.H., Lozano-Perez, T.: Solving the multiple-instance problem with axis parallel rectangles. A.I. (1997)

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

[5] Ray, S., Page, D.: Multiple instance regression. In: ICML. (2001) 425–432

[6] Zhou, Z., Zhang, M.: Multi-Instance Multi-Label Learning with Application to Scene Classification. In: NIPS. (2006)

[7] Wang, J., Zucker, J.: Solving the multiple-instance problem: A lazy learning approach. In: ICML. (2000) 1119–1125

[8] Gartner, T., Flach, P., Kowalczyk, A., Smola, A.: Multi-instance kernels. In: ICML. (2002) 179–186

[9] Ray, S., Craven, M.: Supervised versus multiple instance learning: an empirical comparison. ICML (2005) 697–704

[10] Viola, P., Platt, J.C., Zhang, C.: Multiple instance boosting for object detection. In: NIPS. (2005)

[11] Bunescu, R., Mooney, R.: Multiple instance learning for sparse positive bags. In: ICML. (2007) 105–112

[12] Auer, P., Long, P., Srinivasan, A.: Approximating Hyper-Rectangles: Learning and Pseudo-random Sets. Journal of Computer and System Sciences **57**(3) (1998) 376–388

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

[14] Valiant, L.: A theory of the learnable. Communications of the ACM **27**(11) (1984) 1134–1142

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

[16] Kearns, M.: Efficient noise-tolerant learning from statistical queries. Journal of the ACM (JACM) **45**(6) (1998) 983–1006

[17] Boyd, S., Vandenberghe, L.: Convex Optimization. Cambridge Univ. Press (2004)

[18] Ramon, J., De Raedt, L.: Multi instance neural networks. ICML, Workshop on Attribute-Value and Relational Learning (2000)

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

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

[21] Maron, O., Lozano-Perez, T.: A framework for multiple-instance learning. In: NIPS. (1998)

[22] Zhang, Q., Goldman, S.: EM-DD: An improved multiple-instance learning technique. In: NIPS. Volume 14. (2002) 1073–1080

[23] Freund, Y., Schapire, R.: A decision-theoretic generalization of on-line learning and an application to boosting. Journal of Computer and System Sciences **55** (1997) 119–139

[24] Friedman, J., Hastie, T., Tibshirani, R.: Additive logistic regression: a statistical view of boosting. Technical report, Stanford University (1998)

[25] Mason, L., Baxter, J., Bartlett, P., Frean, M.: Boosting algorithms as gradient descent. NIPS **12** (2000) 512–518

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

[27] Andrews, S., Hofmann, T., Tsochantaridis, I.: Multiple instance learning with generalized support vector machines. A.I. (2002) 943–944

[28] Andrews, S.: Learning from Ambiguous Examples. PhD thesis, Brown University (2007)

[29] Chen, Y., Wang, J.: Image categorization by learning and reasoning with regions. JMLR **5** (2004) 913–939

[30] Bi, J., Chen, Y., Wang, J.: A sparse support vector machine approach to region-based image categorization. In: CVPR. (2005)

[31] Andrews, S., Hofmann, T.: Multiple Instance Learning via Disjunctive Programming Boosting. In: NIPS. (2004)

[32] Zhang, Y., Chen, Y., Ji, X.: Motif Discovery as a Multiple-Instance Problem. (2006) 805–809