Notes on "Multiple Instance Learning: review, taxonomy and comparative study"

Jaume Amores

Contents

1 Introduction

This report provides additional notes on the survey "Multiple Instance Learning: review, taxonomy and comparative study".

2 IS methods

2.1 IS methods following SMI assumption

2.1.1 Diverse Density algorithm

In [1] the authors propose the Diverse Density algorithm, similar in spirit to the APR algorithm. Here, the instance-level classifier $f(\vec{x})$ is expressed as:

$$f(\vec{x};\vec{t},W) = \exp\left((\vec{x}-\vec{t})^T W(\vec{x}-\vec{t})\right), \tag{1}$$

where the parameters to infer are the so-called target point $\vec{t}$ and the diagonal matrix W that provides a scaling for each dimension of the instance space. We can see that the idea is similar to the one of APR. In the latter case, the instance $\vec{x}$ is considered positive if it falls inside the region defined by $\mathcal{R}$. Here, $\vec{x}$ is positive if it falls inside a region defined by $\vec{t}$ and W (i.e., a region centered at $\vec{t}$ and with size determined by W, as can be seen from Eq. 1). The target point $\vec{t}$ is estimated as the one in the instance space which maximizes the Diverse Density (DD) measure. This measure provides a high value for those points in the instance space that are close to at least one instance of each positive bag and far away from all the instances of negative ones (see [2] for the mathematical formulation). Again, the max rule (Eq. **??**) is used for obtaining the bag-level classifier $F(X)$ in terms of $f(\vec{x})$.

2.1.2 MI-SVM algorithm

In the MI-SVM method [3], the authors propose a IS classifier $f(\vec{x};\Theta)$, where Θ are parameters learned by SVM. In order to estimate the SVM, they propose an iterative gradient-descent-like approach. Initially, the SVM parameters are estimated by using a training set of instances where those falling in positive bags receive positive labels and those in negative bags are considered negative [1] Given this initial training set, SVM is estimated and used to

[1] There is a second strategy that consists of taking only one positive instance per positive bag. This positive instance is in fact the average of all the instances in the bag.

re-label the instances in the training set as follows: for each positive bag, the instance $\vec{x}$ with highest score $f(\vec{x};\Theta)$ is considered positive and the rest are discarded. For each negative bag, all the instances are considered negative. The new training set of instances is then utilized to re-estimate a new SVM classifier, which again is utilized to identify a new training set, and so on until convergence. In section 2.1.4 we describe more in detail the type of algorithm followed in this approach.

2.1.3 EM-DD algorithm

In the Expectation-Maximization Diverse Density (EM-DD) algorithm [4], the authors propose a similar iterative approach for estimating the parameters $(\vec{t}, W)$ of the DD instance-level classifier defined in Eq. 1. The authors show that this algorithm obtains better results than the classical DD algorithm of [1].

2.1.4 Algorithmic structure of MI-SVM and EM-DD

The methods of the Standard MI paradigm make use of an instance classifier $f_\theta(\vec{x}) \in [0,1]$, where θ are the parameters of the classifier. This instance classifier is used in order to follow the Standard MI assumption: a bag X is classified as positive if any of its instances $\vec{x} \in X$ is classified as positive. This is achieved if we express the bag classifier $F(X)$ in terms of $f_\theta(\vec{x})$ as follows:

$$F(X) = \max_{\vec{x} \in X} f_\theta(\vec{x}) \tag{2}$$

Note that, using this combination rule [2], if there is any $\vec{x} \in X$ that is positive (i.e., $f_\theta(\vec{x}) > 0.5$) then the bag X receives a positive label i.e., $F(X) > 0.5$.

In order to obtain the instance classifier $f_\theta(\vec{x})$, we need a training set of instances and their associated labels. However, we only have a training set $\mathcal{T}$ of bags, as explained in section 3 of the paper. In order to solve this problem, we describe here an iterative algorithm followed by two of the most well-known Standard MI algorithms: EM-DD [5] (which is an evolution of the well-known DD algorithm of Maron et al. [2]), and MI-SVM [3], both of them evaluated in our experimental study. Let $\mathcal{I}$ be the training set of instances we want to obtain, and let $\mathcal{I}^-$ and $\mathcal{I}^+$ denote the sets containing the negative and positive instances respectively of $\mathcal{I}$. In order to obtain

[2] Another possibility is to use the sum-rule: $F(X) = \frac{1}{|X|} \sum_{\vec{x}_i \in X} f_\theta(\vec{x}_i)$.

1. Obtain an initial instance classifier f_θ.
2. Let $X_1^-, \dots, X_N^-$ be the negative bags of the training set $\mathcal{T}$. Define:

$$\mathcal{I}^- = X_1^- \cup X_2^- \cup \ldots \cup X_N^- \tag{3}$$

3. Iterate until convergence:

    3.1 Let $X_1^+, \dots, X_M^+$ be the positive bags of $\mathcal{T}$. Define:

    $$\mathcal{I}^+ = \{\vec{z}_1, \dots, \vec{z}_M\}, \text{ where } \vec{z}_i = \arg\max_{\vec{x} \in X_i^+} f_\theta(\vec{x}) \tag{4}$$

    3.2 Learn a new instance classifier f_θ by using a training set with negative instances in $\mathcal{I}^-$ and positive instances in $\mathcal{I}^+$.

Figure 1: Algorithm for learning the instance classifier f_θ

these sets, both EM-DD [5] and MI-SVM [3] use the algorithm of Fig. 1. We explain below each step of this algorithm using the MI-SVM method as an example.

Let $X_1^+, \dots, X_M^+$ be the positive bags and let $X_1^-, \dots, X_N^-$ be the negative bags of the training set $\mathcal{T}$. In step 1 of Fig. 1 we obtain an initial classifier f_θ. In order to do so, MI-SVM uses an initial training set where $\mathcal{I}^+ = X_1^+ \cup X_2^+ \cup \ldots \cup X_M^+$ and $\mathcal{I}^- = X_1^- \cup X_2^- \cup \ldots \cup X_N^-$. In other words, we consider as positive instances $\mathcal{I}^+$ all the instances of the positive bags of $\mathcal{T}$, and as negative instances $\mathcal{I}^-$ all the instances of the negative bags. Based on this setting, we train an initial SVM classifier in order to obtain f_θ.

In step 2 of Fig. 1 we define the set of negative instances $\mathcal{I}^-$, which contains all the instances of all the negative bags. The set $\mathcal{I}^-$ is always defined like this because, in the Standard MI assumption, we suppose that all the instances of all the negative bags are negative.

In step 3.1 we define the set of positive instances $\mathcal{I}^+$. According to the Standard MI assumption, each positive bag must have at least one positive instance. For each positive bag of $\mathcal{T}$, we take the instance that is most likely to be positive according to f_θ. In step 3.2., we learn a new instance classifier f_θ based on the training set of instances $\mathcal{I}^+ \cup \mathcal{I}^-$. Note that the step 3.1.

uses the result of step 3.2. (i.e., it uses the learned classifier f_θ), and the step 3.2. uses the result of step 3.1 (i.e., it uses the positive instances $\mathcal{I}^+$), so that both steps 3.1. and 3.2. can be iterated until convergence.

In the EM-DD method [5], instead of using SVM they maximize the Diverse Density (DD) as defined in [2]. The DD measure provides a high value for those points in the instance space that are close to the positive instances in $\mathcal{I}^+$ and far away from the negative instances in $\mathcal{I}^-$ (see [2] for the mathematical formulation). In this case, the classifier f_θ has parameters $\theta = \{\vec{t}, \vec{w}\}$, where $\vec{t} \in \mathbb{R}^d$ is the point with highest DD, and $\vec{w} \in \mathbb{R}^d$ is a vector of weights for each dimension of the instance space. With these parameters, the classifier f_θ is defined as:

$$f_\theta(\vec{x}) = \exp\left(-\sum_{k=1}^{d} w_k^2 (x_k - t_k)^2\right). \tag{5}$$

Regarding the initialization (step 1 of the algorithm), the classifier f_θ is obtained by choosing initial values for the parameters $\theta = \{\vec{t}, \vec{w}\}$. In order to do so, the algorithm of Fig. 1 is run several times, where each time the vector $\vec{t}$ is initialized to one of the instances $\vec{x}$ of one of the positive bags in $\mathcal{T}$, and the vector $\vec{w}$ is initialized to a vector of ones. The parameters that, at the end of the algorithm, lead to a classifier f_θ with highest DD are the ones that are chosen. We refer to [5] for more details.

2.2 IS methods following the Collective assumption

2.2.1 Weighted Collective methods

A generalization of the previous approach is to allow a different weight for each instance. This generalization gives rise to the weighted Collective assumption, as identified by Foulds and Frank [6]. The only one algorithm in the MIL literature that implements this idea is the *Iterative Framework for Learning Instance Weights* (IFLIW) [7], which is based on the Wrapper MI algorithm explained before. The idea of the IFLIW algorithm is to start with the same weighting scheme as in Wrapper MI, and learn an instance classifier $f(\vec{x})$ based on it. The resulting function $f(\vec{x})$ is considered as an approximation to the probability that the instance $\vec{x}$ is positive, denoted as $Pr(+|\vec{x})$. Based on this, the weight $w(\vec{x})$ is updated as follows:

$$w(\vec{x}) = w(\vec{x}) \times \exp(\text{infogain}(Pr(+|\vec{x}), Pr(+))), \tag{6}$$

where $Pr(+)$ denotes the a priori probability that an instance is positive, and infogain is the information gain of $Pr(+|\vec{x})$ relative to $Pr(+)$. After the weights $w(\vec{x})$ are updated, we use them to learn a new classification function $f(\vec{x})$ with the Wrapper MI algorithm. This in turn represents a new approximation to the probability $Pr(+|\vec{x})$, so that we can update again the weights $w(\vec{x})$ using Eq. 6. This way the algorithm is iterated until convergence. Finally, the bag classifier F is computed using a weighted version of Eq. 4 of the paper:

$$F(X) = \frac{1}{\sum_{\vec{x}\in X} w(\vec{x})} \sum_{\vec{x}\in X} w(\vec{x}) f(\vec{x}) \tag{7}$$

3 Vocabulary-based methods

3.1 Histogram-based methods

Let us review additional algorithms that fall into the histogram-based subparadigm of the vocabulary-based family.

3.1.1 Kernel Code-Book

In the H-BoW method we use hard-assignment for obtaining the vocabulary V and for obtaining the mapping function $\mathcal{M}$. The Kernel Code-Book (KCB) method [8] also uses hard-assignment for obtaining the vocabulary V, like in H-BoW, but it uses soft-assignment for the mapping function $\mathcal{M}$. Let us see both components in turn.

Regarding the vocabulary V, the KCB method is exactly the same as the H-BoW. Therefore, it uses the KM clustering algorithm with hard-assignment (each instance is assigned to exactly one cluster). Apart from this, each class C_j is represented by a prototype $\vec{p}_j$, just like in H-BoW.

Regarding the mapping $\mathcal{M}$, we use soft-assignment: an instance $\vec{x}$ might belong to different classes, with a different degree of membership for each class. This is achieved by defining the function f_j in Eq. 10 of the paper as follows:

$$f_j(\vec{x}) = \frac{H_\sigma\left(\|\vec{x} - \vec{p}_j\|\right)}{\sum_{k=1}^{K} H_\sigma\left(\|\vec{x} - \vec{p}_k\|\right)} \tag{8}$$

where $H_\sigma(z) = \frac{1}{\sqrt{2\pi}\sigma} \exp\left(-\frac{z^2}{2\sigma^2}\right)$. Note that f_j is no longer a crisp function, like it was in Eq. 11 of the paper. Now $f_j(\vec{x}) \in [0, 1]$ is a real-valued function which provides the degree of membership of the instance $\vec{x}$ to the class C_j. This degree of membership depends on the similarity between $\vec{x}$ and the j-th prototype $\vec{p}_j$, normalized by the similarity of $\vec{x}$ to the rest of prototypes. The scale σ of the kernel H_σ is estimated by cross-validation (see [8]). The constant Z in Eq. 10 of the paper is set to $Z = |X|$ in order to normalize the histogram.

3.1.2 Bag-of-Words with Gaussian Mixture Models

We present now an algorithm that uses soft-assignment in both the clustering and the mapping functions. The idea is to estimate a Gaussian Mixture Model with Expectation-Maximization (GMM-EM). We describe now how the Vocabulary V and mapping function $\mathcal{M}$ are obtained.

The Vocabulary V is obtained by running the GMM-EM algorithm [9] on the set $\mathcal{D}$ containing all the instances (defined above). It outputs a mixture of K Gaussian classes, where the j-th class C_j is represented by the set of parameters $\theta_j = \{\vec{\mu}_j, \Sigma_j, \alpha_j\}$ which are stored in the Vocabulary. The parameters $\vec{\mu}_j$ and Σ_j are the mean and covariance of the Gaussian class, and the parameter α_j is its relative weight in the mixture, see for example [9].

The mapping function $\mathcal{M}$ is expressed using Eq. 10 of the paper , where the function f_j is defined now as $f_j(\vec{x}) = Pr(C_j|\vec{x})$, i.e., the probability that the instance $\vec{x}$ belongs to the j-th Gaussian cluster C_j, expressed as:

$$Pr(C_j|\vec{x}) = \frac{\alpha_j \mathcal{N}(\vec{x}; \vec{\mu}_j, \Sigma_j)}{\sum_{k=1}^{K} \alpha_k \mathcal{N}(\vec{x}; \vec{\mu}_k, \Sigma_k)},$$

where $\mathcal{N}(\vec{x}; \vec{\mu}_j, \Sigma_j)$ is the classical normal distribution with mean $\vec{\mu}_j$ and covariance Σ_j. Regarding the constant Z in Eq. 10 of the paper, it is set to normalize the histogram $\vec{v}$ so that $\sum_j v_j = 1$.

As we can see, this approach uses a probabilistic framework in order to provide a soft-assignment, where each instance $\vec{x}$ is assigned to class C_j with probability $Pr(C_j|\vec{x})$.

The GMM-EM algorithm estimates more parameters than KM, especially the covariance matrix Σ_j that contains $d \times d$ elements where d is the number of dimensions. Even if we approximate this matrix by forcing it to be diagonal, the algorithm still needs a large number of instances to reliably estimate these parameters, and it is less robust than KM when the number of dimensions d is large and the number of instances is small.

3.1.3 Bag-of-Words with Decision Trees

This method was proposed by Weidmann et al. [10], and the only one difference with the H-BoW method is in the clustering algorithm. Instead of using unsupervised clustering such as KM, the authors propose to use a Decision Tree [11] (DT) as clustering algorithm. Note that a DT is a supervised classifier, so that in order to build it we need a training set of instances and their corresponding labels. For this purpose, we use the collection of instances from all the bags of the training set, and each instance receives the same label (positive / negative) as the bag where it belongs. By doing so, the DT clusters the instances of the training set into different nodes of the tree, by considering both the features of the instances and their labels. Thus, we obtain a clustering procedure that groups together instances falling into a contiguous region of the instance space and having the same label.

A peculiarity of this algorithm is that the clusters are not disjoint, i.e., the set of instances reaching one node of the tree is a sub-set of the set of instances reaching its parent in the tree. In other words, an instance $\vec{x}$ belongs to several clusters: one for each node it reaches while traversing the tree from the root to the leaf.

Regarding the mapping function $\mathcal{M}$, it is the same as the one defined in Eq. 10 of the paper, where now the instance classification function f_j is defined by the DT classifier, i.e., $f_j(\vec{x}) = 1$ iff $\vec{x}$ reaches the j-th node of the tree. Note that now we have $\sum_j f_j(\vec{x}) > 1$, because the instance $\vec{x}$ falls into more than one cluster (class), as explained before. Regarding the constant Z in Eq. 10 of the paper, it is set to normalize the histogram $\vec{v}$ so that $\sum_j v_j = 1$.

Regarding the supervised classifier $\mathcal{G}$, the authors [10] propose to use AdaBoost with decision stumps.

3.1.4 Weidmann's hierarchy

This hierarchy consists of three paradigms in increasing order of generality: the presence-based, the threshold-based and the count-based paradigms (see Fig. 7 of the paper). We review here very briefly the presence-based paradigm, which is the less general one, and the count-based paradigm, which is the most general one. A more detailed explanation can be found in [10, 6].

Let $\mathcal{C} = \{C_1, \ldots, C_M\}$ be a set of classes of instances. Using the notation of [6], let $\Delta(X, C_j)$ be a function that counts the number of instances in the bag X that belong to class C_j. Note that Δ is nothing else than a histogram as the one used in the H-BoW method. The presence-based paradigm states that a bag X is positive if $\Delta(X, C_j) > 1 \ \forall j = 1, \dots, M$, i.e., if there is at least one instance belonging to each of the classes in $\mathcal{C}$. In [6] the authors report the Constructive Clustering Ensemble (CCE) method [12] as following the presence-based paradigm. The CCE algorithm uses an unsupervised clustering method to obtain K clusters of instances. Then, it obtains a histogram such as the one described above for every bag, and they binarize it to obtain a boolean feature vector where each element corresponds to a cluster. Then, a standard supervised learning algorithm (such as SVM or AdaBoost) is trained using these boolean feature vectors. This corresponds exactly to the H-BoW algorithm of section 7.3.1 of the paper, except that the histograms are binarized here. In our experiments, we saw that using binary histograms has a worse performance than using the original non-binarized histogram.

The count-based paradigm is very similar to the presence-based paradigm, but now we impose a more general condition: X is positive if and only if $t_j \leq \Delta(X, C_j) \leq z_j \ \forall j = 1, \dots, M$. In practice, the optimal thresholds t_j and z_j are obtained by the classifier. This is implemented in [10] by using the algorithm explained in section 3.1.3, where the classifier is AdaBoost with decision stumps. A decision stump obtains a threshold t_j or z_j depending on the polarity, which is a parameter that is estimated as part of the learning procedure.

3.2 Distance-based methods

Let us now review additional algorithms that fall into the histogram-based sub-paradigm of the vocabulary-based family.

3.2.1 GMIL and count-based GMIL

The Generalized Multiple Instance Learning (GMIL) algorithm [13] explicitly enumerates all possible axis-parallel boxes. It maps the bag X into a boolean vector where the j-th element is set to 1 if at least one instance from the bag falls into the j-th box. We see now that this method falls into the distance-based sub-paradigm.

The vocabulary V is formed with concepts that are axis-parallel boxes of the instance space. Each concept C_j is parameterized by the center $\vec{p}_j$ and size

r_j of the box, i.e., $\theta_j = \{\vec{p}_j, r_j\}$, where $\vec{p}_j \in \mathbb{R}^d$ is a point in the instance space. In practice, in order to make the algorithm computationally feasible, those boxes covering the same instances of the training set are grouped together, and only one representative box for each group is considered [13].

The mapping function is slightly different from Eq. 12 of the paper:

$$v_j = \begin{cases} 1 & \text{if } \min_{\vec{x}_i \in X} \|\vec{x}_i - \vec{p}_j\|_\infty \leq r_j \\ 0 & \text{otherwise} \end{cases} \tag{9}$$

Thus, the j-th element is 1 if any instance falls into the j-th box.

Regarding the standard supervised classifier $\mathcal{G}$, the authors [13] propose the Winnow learner, which learns efficiently in very large binary spaces.

The count-based GMIL algorithm is an extension of the GMIL algorithm. It performs a similar boolean mapping, but here the j-th element v_j is set to one if there are at least t_j and no more than z_j instances from X falling into the j-th box. A drawback of these algorithms is the high computational demand, even with the speed up introduced in [14], which makes it infeasible for big databases such as the ones used in our study.

3.2.2 Sparse Coding

In this algorithm [15] each concept C_j of the vocabulary V is represented by a prototype vector $\vec{p}_j$, which is called "basis vector". For each instance $\vec{x}$, the method obtains a *sparse* vector of coefficients $\vec{z} = (z_1, z_2, \ldots, z_K)$ such that $\vec{x} \approx \sum_{j=1}^{K} z_j \vec{p}_j$. Note that z_j provides the representational power of the j-th prototype $\vec{p}_j$ in the approximation of $\vec{x}$. Let us define the function $s_j(\vec{x}) = z_j$, providing the importance of the j-th concept from V in the representation of $\vec{x}$.

Let $\mathcal{D} = \vec{x}_1, \ldots, \vec{x}_T$ be all the instances from all the bags of the training set. The Sparse Coding (SC) algorithm described in [15] obtains the prototype vectors $\vec{p}_1, \ldots, \vec{p}_K$ defining our vocabulary V based on the set $\mathcal{D}$. Furthermore, given any instance $\vec{x}$ it provides an algorithm that obtains the vector of coefficients $\vec{z} = (z_1, \ldots, z_K)$ corresponding to $\vec{x}$, so that $\vec{x} \approx \sum_{j=1}^{K} z_j \vec{p}_j$.

Based on the vocabulary V, Yang et al. [16] propose to use a mapping function $\mathcal{M}$ expressed as Eq. 13 of the paper, where remember that we define $s_j(\vec{x}) = z_j$, i.e., $s_j(\vec{x})$ returns the j-th coefficient z_j, which provides the importance of the j-th prototype $\vec{p}_j$ in the representation of $\vec{x}$. In our experiments, however, we found that we obtain better performance if we use a slightly different mapping function: $\mathcal{M} = \vec{m} \circ \vec{\mu} \circ \vec{\nu}$, where $\vec{m}, \vec{\mu}, \vec{\nu} \in \mathbb{R}^K$ and $m_j = \max_{\vec{x}_i \in X} s_j(\vec{x}_i)$, $\mu_j = \frac{1}{|X|} \sum_{\vec{x}_i \in X} s_j(\vec{x}_i)$, $\nu_j = \min_{\vec{x}_i \in X} s_j(\vec{x}_i)$, for $j = 1, \ldots, K$.

4 Implementation of vocabulary-based methods

Let us express here in algorithmic form the details of the implementation used for the vocabulary-based approaches.

```
1. Let $\{s_1, \ldots, s_Q\}$ be the set of vocabulary sizes. Let $R$ be the number of vocabularies that we compute for each size. Let $V_{r,s_q}$ be the $r$-th vocabulary of size $s_q$, where $r = 1, \ldots, R$, and $q = 1, \ldots, Q$. Let $\mathcal{T} = \{(X_1, y_1), \ldots, (X_M, y_M)\}$ be our training set with $M$ bags, where $y_i$ is the label of bag $X_i$.

2. For $q = 1, \ldots, Q$:

    2.1 For $r = 1, \ldots, R$:
        2.1.1 Let $\vec{v}_i^{(r,s_q)} = \mathcal{M}(X_i, V_{r,s_q})$, be the result of mapping the bag $X_i \in \mathcal{T}$ using the vocabulary $V_{r,s_q}$.
    2.2 Let $\vec{w}_i^{(s_q)} = \vec{v}_i^{(1,s_q)} \circ \ldots \circ \vec{v}_i^{(R,s_q)}$ be the concatenation of the vectors obtained with the $R$ vocabularies of size $s_q$.
    2.3 Train a classifier $\mathcal{G}_{s_q}$ using the training set $\{(\vec{w}_i^{(s_q)}, y_i)\}_{i=1}^{M}$.

3. Given a new bag $X$ to be classified:

    3.1 For $q = 1, \ldots, Q$:
        3.1.1 Let $\vec{v}^{(r,s_q)} = \mathcal{M}(X, V_{r,s_q})$, be the mapping of $X$ using $V_{r,s_q}$.
        3.1.2 Let $\vec{w}^{(s_q)} = \vec{v}^{(1,s_q)} \circ \ldots \circ \vec{v}^{(R,s_q)}$
        3.1.3 Apply the classifier $\mathcal{G}_{s_q}$ to the vector $\vec{w}^{(s_q)}$
    3.2 Compute the final score $F(X)$ by averaging over the different sizes:

        $$F(X) = \frac{1}{Q} \sum_{q=1}^{Q} \mathcal{G}_{s_q}(\vec{w}^{(s_q)})$$
```

5 D-BoW algorithm by Auer and Ortner

In this section we show that the Auer and Ortner's method [17] is a D-BoW algorithm. The idea of Auer and Ortner is to use AdaBoost in order to classify bags. Thus, the bag classifier $F(X)$ is defined as $F(X) = \sum_t \alpha_t h_t(X)$., where $h_t(X)$ is the t-th weak classifier of the ensemble. The t-th weak classifier is defined using a ball in instance space, with center $\vec{c}_t \in \mathbb{R}^d$ and radius r_t. The function $h_t(X)$ classifies the bag X as positive if any of its instances falls within the ball, which is expressed as follows:

$$h_t(X) = \begin{cases} 1 & \text{if } \min_{\vec{x} \in X} \|\vec{x} - \vec{c}_t\| < r_t \\ 0 & \text{otherwise} \end{cases} \tag{10}$$

In practice, the ball associated with h_t is restricted to have a center $\vec{c}_t$ that is one of the instances of the positive bags of the training set. Let $V = \{\vec{p}_1, \ldots, \vec{p}_K\}$ be the set gathering all the instances from positive bags. Note that V is nothing else than a vocabulary, where the j-th concept is represented by $\vec{p}_j$, as explained in last section. As we have said, the center $\vec{c}_t$ is one of the instances in V, i.e., $\vec{c}_t = \vec{p}_{I_t}$ where $I_t \in \{1, \ldots, K\}$. Taking this into account, it is straightforward to see that Eq. 10 can be expressed as:

$$h_t(X) = g_t(\vec{v}) = \begin{cases} 1 & \text{if } v_{I_t} < r_t \\ 0 & \text{otherwise} \end{cases} \tag{11}$$

where $\vec{v} = \mathcal{M}(X, V)$ is the vector resulting from mapping the bag X using Eq. 12 of the paper. The distance function $d_j(\vec{x})$ might be the Euclidean distance $d_j(\vec{x}) = \|\vec{x} - \vec{p}_j\|$, as in Eq. 10, or it might be any other distance (in [17] the authors evaluate several metrics).

We can see that the function $g_t(\vec{v})$ in Eq. 11 is a standard decision stump, see for example [18]. The intrinsic parameters learned by this classifier is the index I_t and the threshold r_t. Most authors use an additional parameter that is the polarity p_t of the classifier (see [18]), which here is fixed to 1. Taking all this into account, we can conclude that the method of Auer and Ortner [17] is a D-BoW algorithm, where the bags X are mapped to vectors $\vec{v}$ as explained before and where we apply a standard supervised classifier such as AdaBoost with decision stumps.

Indeed, mapping the bags X to vectors $\vec{v}$ is suggested by Auer and Ortner [17] in order to implement their method, although the authors do not establish the relationship of their method with other D-BoW or Vocabulary-based algorithms.

6 Complexity of the learning algorithms

Almost all of the MIL methods analyzed use a standard supervised classifier, with very few exceptions. In particular, almost all the methods implemented in our survey use the SVM classifier with the standard RBF kernel. Therefore, the learning complexity is similar for most of the methods. In this sense, the difference in performance of the three paradigms (IS, BS and ES) does not lie in the specific learning algorithm, but in the way the information is extracted and introduced to SVM.

In particular, if we look at table 1 of the paper, we see that 11 out of the 14 implemented methods use the same standard SVM learning algorithm. These methods are: SbMI and MI-SVM from the IS paradigm, m. Hauss + SVM, EMD+SVM, Gartner+SVM and Chamfer+SVM from the BS paradigm, and Simple MI, H-BoW (KM), H-BoW (EM), D-BoW and IMK from the ES paradigm.

The difference between these methods lies in the way the information is introduced to the SVM learner. For example, the IS methods build a training set of instances, while the ES methods build a training set of feature vectors, where each feature vector represents an original bag. Thus, the difference between the IS and ES methods lie in the way the method builds a training set and introduces it to SVM. The complexity of the learning algorithm is the same or similar, and it is not the explaining factor of the difference in performance.

The exception is the EM-DD method, where the learner is not the standard SVM. Indeed, in the EM-DD method the model obtained is a simple Gaussian, with a mean and a diagonal covariance matrix. In this case, the complexity of the learner is much lower than the one of the rest of the methods. However, for other IS methods such as MI-SVM, the learner is SVM with a RBF kernel. Interestingly, the EM-DD method obtains higher accuracy than the MI-SVM method across the majority of the databases (see Fig. 12 (a)), despite the fact that the learned model is a simple Gaussian in the case of EM-DD.

Apart from this, some BS methods are implemented with both the the K-NN and the SVM learning algorithm [3]. This was done for measuring the relative impact of the learning algorithm as compared with the impact of the

[3]Examples of this are the m.Hauss+c.KNN and the m.Hauss.+SVM, as shown in table 1 of the paper.

kernel or distance function employed. However, in order to compare methods across different paradigms, we used the SVM learning algorithm for obtaining a fair comparison. Thus, the performance of the different paradigms is not affected by the complexity of the learning algorithm. Instead, the performance is affected by the way the information is extracted and introduced into the learning algorithm.

References

[1] O. Maron, T. Lozano-Pérez., A framework for multiple-instance learning., in: NIPS, 1998.

[2] M. Oded, Learning from ambiguity, Ph.D. thesis, MIT (May 1998).

[3] S. Andrews, I. Tsochantaridis, T. Hofmann, Support vector machines for multiple-instance learning, in: NIPS, 2003.

[4] Q. Zhang, S. A. Goldman, EM-DD: An improved multiple-instance learning technique, in: NIPS, 2001.

[5] Q. Zhang, S. A. Goldman, EM-DD: An improved multiple-instance learning technique, in: NIPS, 2001.

[6] J. Foulds, E. Frank, A review of multi-instance learning assumptions, The Knowledge Engineering Review 25 (1) (2010) 1–25.

[7] J. Foulds, Learning instance weights in multi-instance learning, Master's thesis, University of Waikato (2008).

[8] J. C. van Gemert, C. Veenman, A. Smeulders, J. Geusebroek, Visual word ambiguity, IEEE TPAMI 32 (2010) 1271–1283.

[9] R. O. Duda, P. E. Hart, D. G. Stock, Pattern Classification, John Wiley and Sons Inc, 2001.

[10] N. Weidmann, E. Frank, B. Pfahringer, A two-level learning method for generalized multi-instance problems, in: ECML, 2003, pp. 468–479.

[11] J. R. Quinlan, C4.5: programs for machine learning, Morgan Kaufmann Publishers Inc., 1993.

[12] Z.-H. Zhou, M.-L. Zhang, Solving multi-instance problems with classifier ensemble based on constructive clustering, Knowledge and Information Systems 11 (2007) 155–170.

[13] S. Scott, J. Z., J. Brown, On generalized multiple-instance learning, International Journal of Computational Intelligence and Applications.

[14] Q. Tao, S. Scott, N. V. Vinodchandran, T. T. Osugi, SVM-based generalized multiple-instance learning via approximate box counting, in: ICML, 2004.

[15] H. Lee, A. Battle, R. Raina, A. Y. Ng, Efficient sparse coding algorithms, in: NIPS, 2007.

[16] J. Yang, K. Yu, Y. Gong, T. Huang, Linear spatial pyramid matching using sparse coding for image classification, in: IEEE CVPR, 2009.

[17] P. Auer, R. Ortner, A boosting approach to multiple instance learning, in: Proc. ECCV, 2004.

[18] P. Viola, M. J. Jones, Robust-real time face detection, IJCV 57 (2) (2004) 137–154.