Multi-instance toolbox

mil_tools 1.2.0

A Matlab toolbox for the description, processing and classification of compound data

March 16, 2016

D.M.J. Tax

Contents

Chapter 1

Introduction

In pattern recognition the standard approach is to encode each object by a fixed set of features. For some applications this approach is very challenging. The object may be very complex, and to extract a single feature vector from the object may be a very limited representation.
Take for instance the classification of an image. An image may contain several interesting sub parts, or objects. To classify an image as 'contains a face' or as 'does not contain a face' one often performs a (rough) segmentation or detection, and classifies each individual detection. When one of the detections is classified as a face, the image is classified as 'contains a face'. In most cases the classifier is trained on labeled detections, i.e. detections are manually classified as 'face' or 'non-face'.
In multi-instance learning the task is to classify such compound objects, without relying on a (manual) labeling of training segments. It is required that the classifier itself detects the 'face' and 'non-face' segments, without manual intervention. When the classifier is presented a new image, the classifier has to classify all individual segments, and combine these outputs into a final output for the image.
In the terminology of multi-instance learning, the compound object, or the image, is called a *bag*. The compound object (for instance, image) contains several sub-parts (image regions) that are called *instances*. All instances are assumed to be represented by a single feature vector (all with the features). That means that each bag is represented by a set of feature vectors. Furthermore, each bag has a label, 'positive' or 'negative'. The labels of the instances are unknown. The task of a multi-instance classifier is to find the bag label, based on the set of instance feature vectors.

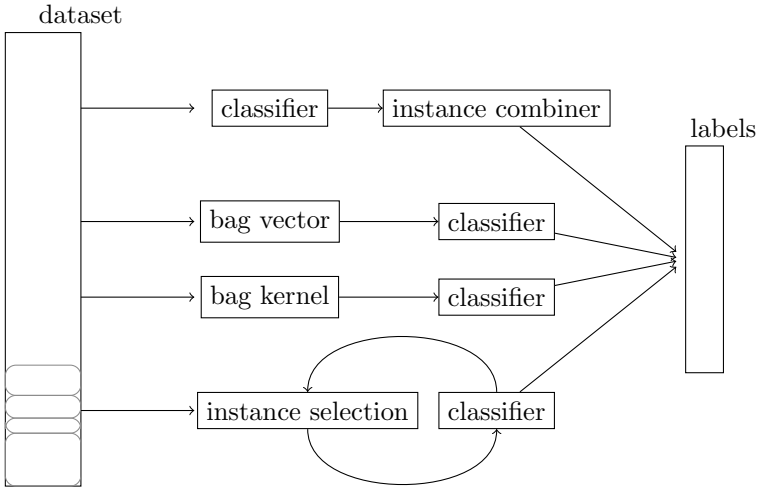

Figure 1.1: General overview of MIL classifiers.

In Figure 1.1 a schematic overview of possible classifier configurations is shown. On the left we start with a dataset, consisting of a large collection of instances (feature vectors) that are organised in bags. For each of the bags one output label should be predicted. This is indicated by the column on the right of the figure. Note that the height of the label matrix is less than the heigh of the instance matrix; there are less bags than there are instances in the dataset.
Now fundamentally three, or four, different approaches can be taken. The first one, indicated in the top of the figure, is to classify each instance individually by a standard classifier, and then combine the instance outputs into an overall output for each bag. Any standard classifier can be used, but the instance combiner should perform the hard task to combine the unreliable instance output labels into a trustworthy bag label.
In the second and third approach the bags of instances are directly 'reduced' to a standard representation. It could be a feature vector (characterizing some statistics on the bags) or a kernel matrix (measuring the similarity between different bags). Any classifier can then be applied to this bag representation.
Finally, in the last approach the informative instance(s) of each bag are selected. Given the informative instances (and possibly removing or relabeling the uninformative ones), a standard classifier is trained. This can be performed in an iterative fashion: the selection of the instances is typically done based on the output of the classifier in a previous iteration.
In the coming chapters of this manual, realisations of these four different approaches will be given.

1.1 General remarks

For all functions in this toolbox some help is available. Just type

```
>> help genmil
```

to get information on the function `genmil`. An overview of the (most important) functions, see the file `Contents.m`.
This toolbox is written to simplify my own work and experiments. It is not industrial-grade robust software for critical applications. Please let me know if there are bugs or inconsistencies. If you want to have additional classifiers implemented, please try to do it yourself, and send me the code, then I can integrate it.

Chapter 2

MIL tools

This MIL toolbox is an extension to Prtools, a Pattern Recognition toolbox for Matlab. So in order to use the functionality of the MIL toolbox, you have to have Prtools. Furthermore, for some evaluation procedures, in particular the Area under the ROC curve, the code of dd_tools is used. Dd_tools is the Data Description toolbox, another extension to Prtools[1].

2.1 The MIL dataset

To implement the multi-instance learning in Prtools, it is required that the standard dataset is extended with an additional identifier that indicates to which bag an instance belongs.
While in the standard Prtools only a dataset and labels have to be given, for a multi-instance dataset also bag identifiers have to be defined. With the use of the command `genmil` this can be done:

```
>> dat = rand(20,2);
>> lab = genlab([10 10]);     % label the first 10 objs 1, the rest 2
>> a = prdataset(dat,lab);    % standard prtools dataset
>> bagid = genlab([5 5 5 5]);
>> a = genmil(dat,lab,bagid)  % MIL dataset with 4 bags, 5 instances each
```

On the command line should appear: a 里的data 即有自己的label 也有自己的bagid

```
20 by 2 dataset with 2 classes: [10  10]
```

The variable `a` now contains a dataset, where each instance is represented by a feature vector. For each of the instances it is also know to which bag it belongs. To inspect how many bags are present, use the command `mildisp`:

```
>> mildisp(a)
20 by 2 MIL dataset with 4 bags: [0 pos, 4 neg]
```

In this example we see that four bags have been found, zero positives and four negatives. In the MIL toolbox it is typically assumed that the labels of the classes are `positive` and `negative`. In Prtools you can change the names of the classes by using `setlablist`:

```
>> b = setlablist(a,{'positive','negative'});
>> mildisp(b)
20 by 2 MIL dataset with 4 bags: [2 pos, 2 neg]
```

Another way of doing this, is to use the function `positive_class`:

[1] Available at `http://prlab.tudelft.nl/david-tax/dd_tools.html`.

```
>> b = positive_class(a,1);
>> mildisp(b)
20 by 2 MIL dataset with 4 bags: [2 pos, 2 neg]
```

Note that in this example each instance has its own label. This label is consistent with the bag label. All instances in a positive bags have a positive label, and all instances in a negative bag have a negative label. This does not always have to be the case. In the strict Multiple Instance Learning setting, one single positive instance in a bag will make the whole bag positive. When all instances are negative, the bag will be labeled positive.
This is shown in the next piece of code:

```
>> dat = rand(20,2);
>> lab = [1;2;2;2;2; 1;2;2;2;2; 2;2;2;2;2; 2;2;2;2;2];
>> baglab = genlab([5 5 5 5]);
>> a = genmil(dat,lab,baglab);
>> a = positive_class(a,1);
>> mildisp(a)
20 by 2 MIL dataset with 4 bags: [2 pos, 2 neg]
```

In the second line only two out of the 20 instances are labeled positive, and that resulted in two positive bags.
The rule that defines how instance labels are propagated to bag labels, can be given with a forth input argument of `genmil`. The possible combination rules are defined in `milcombine`, and the default is `'presence'` (if there is one instance positive, the whole bag is positive). One can change it to, for instance, a majority vote rule:

```
>> a = genmil(dat,lab,baglab,'majority');
>> a = positive_class(a,1);
>> mildisp(a)
20 by 2 MIL dataset with 4 bags: [0 pos, 4 neg]
```

In our example it would mean that none of the bags will be positive anymore. For other combination rules, use `help milcombine`.
The MIL dataset that you just created, is a standard Prtools dataset, and therefore all Prtools classifiers can use them. But they will only look at the instance labels, and the notion of the bags, and bag labels is unknown to them. In order to use that, you need to use classifier from the MIL toolbox.
Finally, there are a few functions to create artificial MIL datasets:

`gendatmilc`	concept MIL problem
`gendatmild`	difficult MIL problem
`gendatmilg`	Gaussian MIL problem
`gendatmilm`	Maron MIL problem
`gendatmilr`	rotated distribution MIL problem
`gendatmilw`	width distribution MIL problem

2.2 Working with bags of instances

When a multi-instance dataset is constructed, you can extract the individual bags and store it in a cell array using:

```
>> [bags, baglab] = getbags(a);
```

The variable `bags` is a cell-array that contains in each cell an individual bag. Furthermore, in `baglab` the label of each bag is returned. This bag label is derived from the labels of the instances in this bag using the `milcombine` function.
In many learning situations you want to have the positive and negative bags split. In these situation it may be useful to use `positive_bags`:

```
>> [pos_bags, neg_bags] = getpositivebags(a);
```

To obtain the bag identifier, that is the bag number an instance belongs to, use `getbagid`.
If you want to split a MIL dataset into a training and a test set, you have to be careful not to split a bag in two. To avoid that, the function `gendatmil` is created. Similarly, if you want to randomize the order of the bags, use `milrandomize`.
Finally, when you want to combine two MIL datasets `a1` and `a2` into one, you cannot just concatenate them like `b=[a1;a2]` because bag labels can be confused. The bag called `1` from `a1` can then not be distinguished from bag `1` from `a2`. Therefore you have to use `milmerge`.

2.3 Training a classifier

Now we have data, and a way to split it in a training set and a test set, we can train a MIL classifier. A typical script looks like:

```
>> a = gendatmilg([20 20]);  % 20 positive and 20 negative bags
>> [x,z] = gendatmil(a,0.7); % use 70% for training
>> w = milboostc(x);
>> z*w*labeld
```

Note, that the labels that are returned on the last line, are the predicted labels for the test bags. Therefore the number of predicted labels is less than the number of rows in `z`.
There are several MIL classifiers implemented, and they will be discussed in chapter 4.

Chapter 3

Evaluation

3.1 Visual inspection, scatterplots and decision boundaries

To get a feeling or intuition of a dataset and a classifier, a scatterplot with a decision boundary is often very insightful. For that, Prtools has the functions `scatterd` and `plotc` available. This only works for 2-dimensional datasets and classifiers that work on 2-dimensional data.
For a MIL problem, there is the additional problem that instances are organized in bags, and classifiers output bag labels. When the standard function `scatterd` is applied to a MIL dataset, a scatterplot of the instances is shown, but it is unclear which instances belong to one bag. One attempt to make that visible is to connect all instances of one bag with straight lines to a center. This is implemented by `scattermil`:

```
>> a = gendatmilg([20 20]);  % 20 positive and 20 negative bags
>> scattermil(a)
```

A similar problem appears when the decision boundary of a MIL classifier is requested: MIL classifiers in principle deal with bags of instances, not with individual instances. In order to still plot a decision boundary, it is assumed that all the 2-dimensional vectors are individual bags. That means that all bags only contain a single instance. Because this may not be very realistic, a warning is given when a MIL classifier is plotted using `plotc`:

```
>> scattermil(a)
>> w = milboostc(a);
>> plotc(w)
Warning: No bag identifiers present: each obj is a bag.
```

3.2 Classification error, ROC curve and crossvalidation

To get to the performance of a MIL classifier, the predictions of the classifier on bags should be compared to the true labels of the bags. This can be done by the standard Prtools function `testc`:

```
>> a = gendatmilg([20 20]);  % 20 positive and 20 negative bags
>> [x,z] = gendatmil(a,0.7); % use 70% for training
>> w = milboostc(x);
>> z*w*testc
```

When a MIL dataset is mapped through a MIL classifier, the resulting output dataset `d = a*w` only contains the output per bag. For each bag dataset `d` contains one line, containing the posterior probabilities per class (the positive and negative class). At this point dataset `d` is indistinguisable from a normal Prtools dataset, and `testc` can directly be applied.

For situations that the classes are very unbalanced, or you want to take a wide range of misclassification costs into account, often Receiver Operating Characteristic (ROC) curves are used. For the output of MIL classifiers the ROC curve can be computed like:

```
>> w = milboostc(x);
>> r = milroc(z*w)
```

This ROC curve `r` can then be used in `plotroc` and `dd_auc`. These two functions are available in Dd_tools, and they plot the ROC curve, and compute the Area under the ROC curve, respectively. In practice, you are not interested in the performance on the training set, but you want to evaluate it on independent test data. In order to make most use of the data, crossvalidation is used. For MIL problems the data should be split according to the bags. This can be done using `milcrossval`. The implementation is such that you, the user, still has to make a loop over de different folds. The information which objects are used for testing in which fold, is stored in an additional variable (`I` in the next example). A typical implementation looks like:

```
a = gendatmilg([50 50]);   % get data
nrfolds = 10;              % define nr. of folds
perf = zeros(nrfolds,1);   % storage for results
I = nrfolds;               % index variable for crossvalidation
for i=1:nrfolds
   [x,z,I] = milcrossval(a,I);    % split in train and test
   w = milboostc(x);              % train on train set
   out = z*w;                     % get test output
   perf(i) = dd_auc(out*milroc);  % AUC performance
end
```

Chapter 4

Classifiers

In Figure 1.1 an overview of MIL classification approaches is shown. The top row shows the 'naive' approach, where a standard classifier is directly trained on the individual instances. The next two rows extract a bag-level representation, and use that as input for a standard classifier. The last approach contain the 'real' MIL approaches, that typically involve a selection mechanism to select the informative set of instances from each bag. For each of the approaches some classifiers are present in the toolbox. They will be discussed in the coming sections.

4.1 Naive approach

In the construction of the MIL dataset, in principle each instance in a bag is labeled (see 2.1). Given these labeled instances, a standard Prtools classifier can be trained. In order to get an output label per bag, the instance predictions have to be combined. For this, the function `milcombine` is defined. In order to train the sequence of Prtools classifier and combination rule, you can first define an untrained mapping `u`. This untrained mapping can be trained in one step, and then applied to new data:

```
u = loglc*milcombine;     % untrained sequence of classifier and combiner
a = gendatmilg([50 150]); % some dataset
[x,z] = gendatmil(a,0.7); % split in train and test set
w = x*u;                  % train on train set
out = z*w;                % evaluate on test set
out*labeld
```

The function `milcombine` defines how the bag label is derived from the instance predictions. Several rules are defined (`'presence'` is the default):

`'presence'`	indicate the presence of the positive class
`'first'`	just copy the first label
`'majority'`	take the majority class
`'vote'`	identical to 'majority'
`'noisyor'`	noisy OR
`'sumlog'`	take the sum of the log(p)'s (similar to the product comb.)
`'average'`	average the outcomes of the bag
`'mean'`	identical to 'average'
`F=0.1`	take the F-th quantile fraction of the positives

So, if we want to use the quadratic classifier, and want to get a positive bag when at least 10% of the instances are classified as positive, you have to define the following mapping:

```
u = qdc*classc*milcombine([],0.1);
```

An all-in-one MIL classifier that does the same, is called `simple_mil`.

4.2 Bag representations

There are several procedures defined to extract a feature vector from a bag of instances. The first, straightforward way is to use `milvector`, where basic statistics are computed, as the mean vector, the minimum and maximum feature values, the covariance matrix, or even the number of instances in a bag.
This function is a mapping, so it can be combined with other mappings to form a sequence of mappings. For instance, to compute the mean vector per bag, and train an LDA classifier on top of that, you can define the mapping:

```
u = milvector([],'m')*ldc;
```

A slightly more advanced approach to obtain a fixed-length vector representation of bags, is by using a *Bag of Words* (BoW) approach. This approach originates from natural language processing, where a document is represented by a vector of word counts. The word order is therefore lost. To apply this approach to general MIL datasets, the collection of 'words' should be defined first. In this toolbox, these 'words' are defined as the cluster centers obtained from a mixture of Gaussians.[1] Next, for each bag, the instances are assigned to the closest cluster center (or, word). This results in a histogram over the word for each bag.
This procedure is implemented in `bowm`. The number of clusters, or words `K` should be defined beforehand. Running this code:

```
K = 30;
u = bowm([],K)*ldc;
```

will result in an `K`-dimensional dataset. Typically, a hard assignment is not very good, and it is better to use the soft assignments:

```
u = bowm([],K,'soft')*ldc;
```

An alternative way of representing a bag by a single vector, is by defining a kernel or similarity, between bags. Several kernels are defined in the function `milproxm`. These kernels can then be used in kernel machines, like a support vector classifier. For this the function `sv_mil` is defined:

```
C = 10;            % tradeoff parameter in support vector classifier
u = scalem([],'variance') * sv_mil([],C,milproxm([],'minmin'));
```

In this example, I used the `'minmin'` kernel, that computes all pairwise (euclidean) distances between all instances of two kernels, and then uses the minimum distance. Because the euclidean distance is sensitive to the scaling of the features, I first rescale the data such that all features have a variance of 1. This is achieved by the mapping `scalem` in front of the `sv_mil`.
Other kernels are also implemented:

[1] The mixture of Gaussians implementation of Dd_tools is used. You can download that from `http://prlab.tudelft.nl/david-tax/dd_tools.html` .

'minmin'	Minimum of minimum distances between inst.
'summin'	Sum of minimum distances between inst.
'meanmin'	Mean of minimum distances between inst.
'meanmean'	Mean of mean distances between inst.
'mahalanobis'	Mahalanobis distance between bags
'hausdorff'	(maximum) Hausdorff distance between bags
'emd'	Earth mover's distance (requires emd_mex!)
'linass'	Linear Assignment distance
'miRBF'	MI-kernel by Gartner,Flach,Kowalczyk,Smola, basically just summing the pairwise instance kernels (here we use the RBF by default)
'mmdiscr'	Maximum mean discrepancy, from Gretton, Borgwardt, Rasch, Schoelkopf and Smola
'miGraph'	miGraph kernel. This requires two additional parameters in KPAR: KPAR[1] indicates the threshold on the maximim distance between instances (in order to allow an edge between the two instances), KPAR[2] indicates the $\gamma = 1/\sigma^2$ in the RBF kernel between instances.
'rwk'	Random Walk graph kernel. KPAR[1] is defined as in miGraph. KPAR[2] indicates gamma in the RBF kernel between nodes. KPAR[3] indicates λ in infinite sum over walks ($0 < \lambda < 1$).
'spk'	Shortest Path graph kernel. KPAR[1] is defined as in miGraph. KPAR[2] and KPAR[3] indicate gamma parameters in the RBF kernels between nodes and between edges. KPAR[4] indicates the trade-off between nodes and edges.

4.3 MIL classifiers

The real MIL classifiers, that perform a selection of interesting instances from training bags, covers a wide range of different classifiers. Below is a short list of implemented MIL classifiers:

apr_mil	the very first MIL classifier, fitting an axis-parallel rectangular decision boundary[DLLP97]
maxDD_mil	maximum Diverse Density, fitting a concept in a probabilistic way[MLP98]
emdd_mil	EM version of maximum Diverse Density[ZG02]
misvm	iterative SVM that selects a fraction of the most positive instances from positive bags[ATH03]
spec_mil	generalized version of misvm in which the SVM can be replace by any classifier
milboostc	Boosting approach to MIL[BDTB08]
miles	the MILES classifier (Multiple Instance Learning via Embedded Instance Selection) [CBW06]

The training and evaluation of these classifiers is identical to standard Prtools classifiers:

```
>> a = gendatmilg;
>> [x,z] = gendatmil(a,0.7);
>> u = scalem([],'variance')*miles([],1,'r',2);
>> w = x*u;
>> perf = dd_auc(z*w*milroc)
```

In this example I train a MILES classifier on 70% of the data. In the MILES classifier a radial basis kernel is used, which depends on the euclidean distance between instances. Because the euclidean distance is depending on the (relative) scales of the features, I rescale the feature space before I train the MILES. This rescaling renormalizes the features such that each feature has a variance of 1, and this is performed by scalem.
To find out what parameters to set, use the help-functionality.

Chapter 5

Remarks, issues

5.1 Instance labels and bag labels

One of the potential advantages of MIL classifiers is, that they are not only capable of predicting bag labels, but they may also be able to recover labels of instances. This holds particularly for the 'real' MIL classifiers, as mentioned in section 4.3. One way to get the instance labels back from a MIL classifier, is to remove the bag information that is stored in the dataset. This can be done by the function `unmil`.
In principle, the MIL classifier should complain that it cannot extract the bags. In most MIL classifiers, fortunately, the evaluation is started by a call to `genmil`. This will test for the presence of bags in the dataset. If bags are not defined, it will define each instance as an individual bag, and a warning will be given:

```
>> a = gendatmilg;
>> w = milboostc;
>> b = unmil(a);
>> b*w*labeld
Warning: No bag identifiers present: each obj is a bag.
> In genmil at 65
  In milboostc at 92
  In prmap at 232
  In prmapping.mtimes at 14

ans =

positive
negative
negative
...
```

5.2 Very large datasets and batches

Prtools has the feature that when very large datasets are used, this dataset can be processed in batches. In particular for evaluating a classifier on large datasets, it can be trivially done, because it is typically assumed that all data is iid sampled from some distribution. For MIL problems we cannot sample instances at random, because they are organized in bags. This batch processing is therefore not possible for MIL datasets, and should be shut off. This is done in all MIL classifiers by defining inside each mapping:

```
w = setbatch(w,0);
```

So if you have defined your own MIL classifier, and want to apply it to very large datasets, don't forget to unset the batch processing.

Bibliography

[ATH03] S. Andrews, I. Tsochantaridis, and T. Hofmann. Support vector machines for multiple-instance learning. In *Advances in Neural Information Processing Systems 15*, pages 561–568. MIT Press, 2003.

[BDTB08] Boris Babenko, Piotr Dollar, Zhuowen Tu, and Serge Belongie. Simultaneous learning and alignment: Multi-instance and multi-pose learning. In *Workshop on Faces in 'Real-Life' Images: Detection, Alignment, and Recognition*, 2008.

[CBW06] Y. Chen, J. Bi, and J.Z. Wang. MILES: Multiple-instance learning via embedded instance selection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 28(12):1931–1947, 2006.

[DLLP97] T.G. Dietterich, R.H. Lathrop, and T. Lozano-Perez. Solving the multiple instance problem with axis-parallel rectangles. *Artificial Intelligence*, 89(1-2):31–71, 1997.

[MLP98] O. Maron and T. Lozano-Pérez. A framework for multiple-instance learning. In *Advances in Neural Information Processing Systems*, volume 10, pages 570–576. MIT Press, 1998.

[ZG02] Q. Zhang and S. Goldman. EM-DD: An improved multiple-instance learning technique. In *Advances in Neural Information Processing Systems*, volume 14. MIT Press, 2002.