Multiple Instance Learning using EMDD

Tejas Khairnar
Arizona State University
Email: tkhairna@asu.edu

Vivin Paliath
Arizona State University
Email: tkhairna@asu.edu

Abstract—**In this paper, we investigate and implement the Expectation Maximization Diverse Density (EM-DD) [1] algorithm, and compare its performance to other, similar algorithms. EM-DD is an approach that has seen significant success in Multiple-Instances Learning problems and is a generalization of the supervised-learning classification-problem. Our goal is to demonstrate an understanding of this algorithm by providing a valid implementation and measuring its performance against various data sets. We will also compare the performance of the EM-DD algorithm against the MatLab MIL-toolkit implementation.**

I. Introduction

The Multiple Instances Learning (MIL) problem [2] is a variation of the general supervised-learning classification-problem. In the standard version of this problem, labeled, *individual* instances presented one at a time until the classifier is able to learn how to distinguish positive and negative instances. However, in practice, it is not always easy to identify discrete, positive and negative instances; in some cases it may only be known that a set of instances either contains or does not contain a positive instance. MIL aims to solve this variation of the classification problem. In MIL, instead of presenting a single instance, a bag of instances is presented to the algorithm, along with a label that describes the bag as either positive or negative. A bag is given a positive label if it contains at least one positive instance, and a negative label if it does not contain any positive instances. The goal of the MIL problem then, is to classify unseen bags or instances as either positive or negative, based on the training data. For this project, we specifically focus on the EM-DD algorithm and use it to solve the MIL problem.

II. Implementation

We successfully implemented the EM-DD algorithm using the Python programming-language and verified its correctness against a simple, clearly-separable synthetic data-set. This synthetic data-set consists of 100 bags with 10 instances each. Positive instances are represented by the vector $[1, 0, 1, 1, 0]^T$ and negative instances are represnted by the vector $[0, 1, 0, 0, 0]$. There were 50 positive bags and 50 negative bags. Positive bags were initialized with a random number of positive instances, with no more than a maximum of 5. Our implementation is able to clearly classify the instances, which demonstrates its correctness. As our implementation is in Python, we used the SciPy library to perform mathematical-operations. To minimize the criterion function, we used `scipy.optimize.minimize` with the limited-memory variant of BFGS search [3], using a calculated gradient.

While implementing the algorithm, the initial issues that we ran into were due to unfamiliarity with SciPy, and were quickly resolved. Later we also noticed that our classification performance was not as good as we expected. Initially, we were using `scipy.optimize.minimize` with a numerically-estimated gradient. To address performance problems, the documentation suggested the use of an explicitly-defined gradient function. We followed the suggestion and provided such a function, which seemed to not only improve performance, but also significantly decreased training-time. Another reason for the discrepancy was that by default, the EM-DD algorithm as implemented in the MILL toolkit, uses the average of the estimated target and scale vectors from each cross-validation run during prediction. In contrast, our implementation used the target and scale vector corresponding to the best diverse-density. By changing our implementation to instead use the average of the results, we were able to further increase performance.

Since the EM-DD algorithm as described uses 10-fold cross-validation, we employed the same method as well. However the algorithm, as described, does not use the entire training-set during each cross-validation run. Instead, it randomly chooses up to k positive bags from the training set and then uses each instance as a hypothesis. Unfortunately, no information was provided regarding an ideal value of k. With some experimentation, we settled on using $k = 10$; using larger values of k, or even setting k to the total number of positive instances in the training-set for that validation run, only seemed to increase the training-time without any appreciable improvement in accuracy. An issue we noticed here, was that the accuracy performance reported in the original paper is based on the average accuracy of the classifier against when run against each validation set in the 10-fold cross-validation. In contrast, we were reporting our accuracy based only on classification against the *entire* set. By changing our accuracy calculation to be more in line with the original implementation, we started seeing results closer to what we expected, even to the point where it seems that our reported average-accuracy (across 10-fold cross-validation) seemed to be better than the implementation in the MILL toolkit.

Our implementation of the EM-DD algorithm has some differences when compared to the implementation in the MILL toolkit. In the EM portion, the MILL toolkit stops iteration when the difference between the current density and the previous one falls below a certain threshold. On each iteration, it will compare the previous density to the current one, and then set the value of the previous density to the current one in preparation for the next iteration. In contrast, our implementation will only update the value of the previous density if the current density is better. In our implementation, the iteration will terminate if the difference is below a certain threshold, or if there is no change in density. We also calculate precision and recall in addition to accuracy. Furthermore, we also employ 10-fold cross-validation like the original implementation, to calculate our average accuracy; the MILL toolkit implementation does not seem to employ a similar technique. In general, our Python implementation, while correct, does not seem to approach the level of accuracy as described in the original paper, although in some instances it seems to perform better than the MILL implementation. A more thorough explanation of the discrepancy, and a rationale are provided in the results section. A portion of the output from our implementation is also shown in Figure 1.

Fig. 1. Program output from `emdd.py`

```
Partition 1:
    Density: 46.214994685717684
    Accuracy: 0.8
    Precision: 0.8333333333333334
    Recall: 0.8333333333333334

Partition 2:
    Density: 44.53627762779977
    Accuracy: 1.0
    Precision: 1.0
    Recall: 1.0

Partition 3:
    Density: 45.81991329148556
    Accuracy: 0.8
    Precision: 0.5
    Recall: 0.5

Partition 4:
    Density: 40.59239723412051
    Accuracy: 0.9
    Precision: 1.0
    Recall: 0.8

Partition 5:
    Density: 46.02459327281305
    Accuracy: 0.7
    Precision: 0.6
    Recall: 0.75

Partition 6:
    Density: 45.61426356661195
    Accuracy: 0.8
    Precision: 1.0
    Recall: 0.7142857142857143

Partition 7:
    Density: 41.683017597801886
    Accuracy: 0.8
    Precision: 0.8
    Recall: 0.8

Partition 8:
    Density: 43.78349979801756
    Accuracy: 0.9
    Precision: 0.8
    Recall: 1.0

Partition 9:
    Density: 45.34380248285251
    Accuracy: 0.9
    Precision: 1.0
    Recall: 0.8571428571428571

Partition 10:
    Density: 48.9783103825846
    Accuracy: 1.0
    Precision: 1
    Recall: 0

Across 10-fold cross-validation:
Average accuracy: 0.8600000000000001
Average precision: 0.8533333333333333
Average recall: 0.7254761904761905

Classifying against test set

Threshold 0.5
Total bags 92
Total positive bags 47
Total negative bags 45
Total predicted positive bags 64
Total predicted negative bags 28
True positives 44
False positives 20
True negatives 25
False negatives 3
Accuracy 0.75
Precision 0.6875
Recall 0.9361702127659575
```

III. Discussion and Analysis

In this section, we examine aspects of the EM-DD algorithm, including how and why it works, any possible limitations it may have, and also look at the problem of feature and instance selection. In addition, we will investigate other methods to solve the MIL problem and compare them to EM-DD. While evaluating the performances of these other algorithms against our EM-DD implementation, we only used the MUSK1 data-set as other data-sets were taking an extremely-long time. However, we did use other data-sets to compare our implementation of the EM-DD algorithm against the one provided in the MILL toolkit.

A. EM-DD

Before looking at the EM-DD algorithm, it is useful to look at the definition of diverse density. If we assume that there is a hypothetical point t in the feature space that identifies the ”true concept”, the goal of DD is to identify the region that has positive instances from the most number of *different* positive bags, while also being far away from negative instances. Hence the diverse-density problem involves maximizing the probability $P(x = t \mid B_1^+, \dots, B_n^+, B_1^-, \dots, B_m^-)$, where B_i^+ is a positive bag and B_j^- is a negative bag. This is effectively equivalent to maximizing the following [4]:

$$\arg\max_x \prod_i P(x = t \mid B_i^+) \prod_i P(x = t \mid B_i^-) \tag{1}$$

From this general definition of the maximum diverse-density, a noisy-or model is used to define the terms in the products. Hence the probability all points did not miss the target is $P(x = t \mid B_i^+) = P(x = t \mid B_{i1}^+, B_{i2}^+, \dots) = 1 - \prod_j (1 - P(x = t \mid B_{ij}^+))$ and likewise $P(x = t \mid B_i^-) = \prod_j (1 - P(x = t \mid B_{ij}^-))$. The probability of a particular instance being the potential target-instance is then based on the distance between them: $P(x = t \mid B_{ij}) = \exp(- \|B_{ij} - x\|^2)$. The intuition is that if one of the instances in a positive bag is close to x, then $P(x = t \mid B_i^+)$ is high. Hence, if all positive bags have an instance close to x and no negative ones do, the diverse density of x will be high.

It is easy to see why this approach is successful; observe that while the diverse density at the intersection of n bags is exponentially higher than at $n-1$ bags, the presence of a single negative instance is enough to drive down the diverse density. Hence, the optimal solution is one that includes the maximum number of positive instances and no negative-instances, which is exactly what we want.

The diverse density formulation addresses the problem of feature-selection by taking into account the relevance of features by learning a scaling vector s in addition to the location x that maximizes the diverse density ($\|B_{ij} - x\|^2 = \sum_k s_k^2(B_{ijk} - x_k)^2$). This allows the algorithm to disregard irrelevant features and consider important ones, especially considering that it uses the Euclidean distance as a metric for ”closeness”. Therefore, the EM-DD algorithm simultaneously solves the problem of feature-selection and classification.

In the standard diverse-density forumlation, the authors use gradient ascent with multiple starting-points to identify the target and scaling vector. But in the EM-DD formulation, the positive instance that corresponds to the bag label is viewed as a missing attribute that can be estimated using the expectation-maximization approach [1]. The algorithm starts with an initial guess of the target point t using instances from positive bags. It then repeatedly performs the *E*-step and the *M*-step to search for the maximum-likelihood hypothesis. In the *E*-step, the algorithm needs to select the appropriate instances from the bag. It identifies the most-likely instance from each bag that is responsible for the label of the bag, based on the *current* target t. In the *M*-step, the algorithm uses a gradient-ascent search to find the new target t' that maximizes the diverse-density at h. After the maximization set, t is set to t' and the algorithm returns to the first step and runs until convergence.

While the EM-DD algorithm performs well, it does have some limitations - it is not able to guarantee a globally-optimal solution and can get stuck in local minima. In addition, the algorithm requires multiple runs with different starting points, due to which the training process can be time-consuming. Another issue with EM-DD is that performance can be affected by the initial guess of the scaling vectors [5]. The EM-DD algorithm also does not *directly* classify a test-bag as positive or negative; instead it learns a thresold between positive and negative instances [5].

The issue relating to selecting a good initial scaling-vector could be solved by using various feature-selection methods, such as Principal Component Analysis (the scaling vector could be initialized based on the eigenvalues for each corresponding-feature), hypothesis testing, or through clustering. While we haven’t verified this, perhaps it might be possible to select hypotheses-instances on each cross-validation run by using the target learned from the previous cross-validation run; this could cut down training time by eliminating instances from the selected bags that are not close to the target (i.e., negative instances).

B. *Iterated-discrim APR*

The iterated-discrim APR algorithm is an Axis-Parallel-Rectangle algorithm for solving the multiple-instance learning problem [2]. It starts with a point in the feature-space and then grows a box with the aim of finding the smallest box that covers at least one instance from each postive-bag and no instances from negative bags. It has the same goal as the EM-DD algorithm, but uses a different approach. After identifying this smallest box, it is further expanded using some statistical techniques. The DD and the EM-DD algorithms are improvements over this technique, and avoid some of the potentially-hard computational-problems associated that heuristics in the iterated-discrim algorithm require. This algorithm’s performance in comparison to our EM-DD implementation is shown in the results section for the MUSK1 data-set.

TABLE I
Comparison of accuracy on MUSK1 data-set between EM-DD implementations and other MIL algorithms

Algorithm	Accuracy
EM-DD (paper implementation)	96.8%
EM-DD (our implementation)	86%
EM-DD (MILL implementation)	84.9%
Iterated-discrim APR	91.3%
SVM	80.5%
Citation-kNN	90.4%

TABLE II
Comparison of accuracy between Python and MILL implementation of EM-DD on multiple data-sets

Data Set	Accuracy (Python)	Accuracy (MILL)
MUSK1	86%	84.9%
Synthetic data-set 1	79%	73.25%
Synthetic data-set 4	76%	82.4%
Diabetic-retinopathy data-set	83%	55.05%

C. *SVM*

Support Vector Machine [6] is a non-parameteric machine-learning algorithm. It works by solving for a decision boundary that maximizes the ”margin” between the two classes. This boundary is the midpoint between two ”support vectors”, which pass through those positive and negative instances that are closest to the midpoint, and equidistant from it. As a result, the algorithm does not need to retain all information about the training data and has a small number of parameters, which are from a subset of the training data itself. We compared this algorithm’s performance to our EM-DD implementation against the MUSK1 data set; our findings are shown in the results section.

D. *Citation-kNN*

Citation-kNN [7] is an approach that adapts the standard kNN algorithm to the multiple-instance learning-problem. While regular kNN classifies a point p in the feature space based on the class of the majority of k nearest-neighbors, citation-kNN takes into account the not only the neighbors of p, but also those neighbors that count p as *their* neighbor. Hence the neighbors that *reference* or *cite* that same point could also be taken into account. We used the MUSK1 data-set to compare the performance of this algorithm against our EM-DD implementation. The findings are shown in the results section.

TABLE III
Accuracy for Python EM-DD implementation on all data-sets

Data Set	Accuracy
MUSK1	86%
MUSK2	80.99%
Synthetic data-set 1	79%
Synthetic data-set 4	76%
Diabetic-retinopathy data	83%
Elephant	92%
Tiger	85%
Fox	78%

TABLE IV
Precision for Python EM-DD implementation on all data-sets

Data Set	Precision
MUSK1	85.33%
MUSK2	89.66%
Synthetic data-set 1	85.5%
Synthetic data-set 4	80.5%
Diabetic-retinopathy data	86.73%
Elephant	94.4%
Tiger	89.33%
Fox	79.4%

TABLE V
Recall for Python EM-DD implementation on all data-sets

Data Set	Recall
MUSK1	72.54%
MUSK2	57%
Synthetic data-set 1	70.73%
Synthetic data-set 4	74.75%
Diabetic-retinopathy data	85.9%
Elephant	90.25%
Tiger	79.16%
Fox	73.8%

TABLE VI
Accuracy for Python EM-DD implementation on all data-sets, over the entire set

Data set	Accuracy
MUSK1	75%
MUSK2	64.7%
Synthetic data-set 1	50%
Synthetic data-set 4	50%
Diabetic-retinopathy data	61.3%
Elephant	59.5%
Fox	52.5%
Tiger	50.5%

IV. Results

We compared the performance of our EM-DD implementation against MILL toolkit's implementation for the following data-sets:

- Synthetic Data [8]
- MUSK1 [9] and MUSK2 [10]
- Diabetic Retinopathy Data [8]
- Elephant, Fox, and Tiger [11]

The synthetic data mentioned above is a data-set that was provided to us, and is different from the degenerate, clearly-separable data-set that we used to establish the correctness of our implementation. The provided synthetic data-set is meant to be a ”control” data-set that we can use to easily compare our algorithm's performance to the MILL implementation. MUSK1 and MUSK2 data-sets contain features that describe the conformation of a particular kind of molecule, some of which are ”musks” (a certain kind of aromatic compound) and others are not. The diabetic-retinopathy data-set contains scans of individuals with and without diabetic retinopathy, which is a complication arising from diabetes, that causes damage to the retina. The Elephant, Fox, and Tiger data-sets each consist of images; each bag contains multiple instances that are sections of a single image. Some of the images in the data-set are images of the corresponding animal, while others are not.

To compare the performance of our implementation against other MIL algorithms, we only used the MUSK1 data-set. This was due to a lack of time, as some of these algorithms took an especially-long time to run. We also ran into various errors while trying these data-sets against the MILL toolkit implementations of these algorithms. The problems were difficult to solve due to our lack of familiarity with MatLab. Since use of the MUSK1 data-set did not seem to present any such problems, we decided to restrict our comparisons to just that data set; the results can be seen in table I.

To evaluate the performance of our algorithm against the MILL toolkit, we used synthetic data, MUSK1, and diabetic-retinopathy data. We wanted to perform a more complete comparison across all data-sets. Unfortunately, the MILL implementation of the EM-DD algorithm took an exceptionally-long time to run on the MUSK2, Elephant, Fox, and Tiger data-sets without seeming to complete, and in some cases would cause MatLab to freeze. The results of the comparison between our EM-DD implementation and the MILL one can

TABLE VII
PRECISION FOR PYTHON EM-DD IMPLEMENTATION ON ALL DATA-SETS, OVER THE ENTIRE SET

Data set	Precision
MUSK1	68.75%
MUSK2	52.94%
Synthetic data-set 1	50%
Synthetic data-set 4	50%
Diabetic-retinopathy data	61.3%
Elephant	55.25%
Fox	51.31%
Tiger	50.26%

TABLE VIII
RECALL FOR PYTHON EM-DD IMPLEMENTATION ON ALL DATA-SETS, OVER THE ENTIRE SET

Data set	Recall
MUSK1	93.61%
MUSK2	69.23%
Synthetic data-set 1	100%
Synthetic data-set 4	100%
Diabetic-retinopathy data	100%
Elephant	100%
Fox	98%
Tiger	97%

be seen in table II.

While our implementation seemed to outperform the MILL implementation in some instances, it does not approach the accuracy described in the original paper. It is also important to note that the accuracy reported is not absolute - we observed that there seemed to be an upper and lower-bound for the accuracy that seemed to be dependent on the k bags that were chosen before the EM portion of the algorithm. Furthermore, we also shuffle the bags before we divide them into partitions for 10-fold cross-validation, and this could slightly affect the results as well. A possibility for future investigation would be to record accuracy over multiple runs to examine how it varies, and to see if it might be possible to establish an upper and lower bound. Regarding the accuracy with respect to the original implementation, we feel that it might be due to shortcomings in `scipy.optimize.minimize`. In contrast to MatLab's `fpmincon`, the SciPy optimizer lacks certain heuristics and optimizations that its MatLab counterpart uses. However, we have been able to verify that our implementation is essentially correct by testing it on a degenerate, clearly-separable set. Hence, one possible-solution to increase accuracy might be to use an external, more-performan solver, or investigate the use of MatLab bindings in SciPy.

To evaluate the overall performance of our algorithm, we ran it against all of the data sets. The average accuracy reported across 10-fold cross-validation can be seen in table III. We also calculated the average precision and recall of the implementation against all the data-sets, and the results can be seen in tables IV and V. Interestingly, there seemed to be a large disparity in performance in some cases, when we ran the the classifier against the *entire* data-set. In contrast to the 10-fold averaged cross-validation results, accuracy seemed to be much worse. Note that the target and scale used in the classification in this case is the averaged target and scale from the cross-validation results. For the MUSK1 and MUSK2 data sets, test-data was provided and so we were able to measure our performance against those. For the other data-sets, there was no test-data and so we evaluated the performance of the classifier against the entire test-data set.

Given our experience with our machine-learning algorithms, we expected the performance to be at least as good as the average result from the 10-fold cross validation, since the classification is against the test set. However, we were surprised to see that this was not necessarily the case. We feel that this might be due to the fact that the classifier learns a threshold between the positive and negative instances, and that this threshold might not generalize appropriately over the entire set. This could be seen when looking at the individual cross-validation results - in some cases accuracy was very low, depending on the bags in the validation set for that particular cross-validation run. In contrast, the MILL implementation has much better accuracy when performing against the entire data-set. This again, might be due to the fact that `scipy.optimize.minimize` does not perform as well as MatLab's `fpmincon`.

A future avenue of exploration to verify this hypothesis, would be to modify the MILL implementation to calculate the average accuracy across 10-fold cross-validation and compare the results to our implementation; we suspect that the performance will be better than our averaged results. The accuracy, precision, and recall across the complete sets of data are shown in tables VI, VII, and VIII. Note that while the recall on some sets is 100%, this is not necessarily a good result; this can be seen when looking at the precision (50%) in those cases. What this means is that the classifier classifies every instance as positive, which leads to 100% recall, but poor precision. Another possibility for future investigation is to examine the role of the classification-threshold a little more in these situations; the MILL implementation uses a threshold of 50% to classify an instance as positive or negative. If a particular instance's probability of ”closeness” as based on the density is higher than this threshold, it is classified as positive. It may be that in some cases, the threshold of classification for positive instances is much higher than 50% and so using this higher value might provide better results.

V. Conclusion

We were able to successfully implement the EM-DD algorithm and compare its performance against the MILL toolkit implementation. In addition, we were also able to compare the performance of our EM-DD implementation against other algorithms that aim to solve the MIL problem, as well as the EM-DD implementation in the MIL toolkit. We note that our average accuracy across 10-fold cross validation is better than the MIL implementation. However, the MIL implementation performs better across the entire data-set compared to ours. To address these discrepencies and to understand the reason behind them, we feel that it would be useful to perform additional experiments, use a better minimizer, and also examine the role of the threshold used in classifying instances. While our implementation's performance does not match the results in the original paper, we note that this might also be due to limitations in the optimization method used by `scipy.optimize.minimize`, which does not perform as well as MatLab's `fmincon`. Furthermore, we were able to verify that our implementation is essentially correct by testing it against a degenerate, easily-separable, synthetic data-set. Through our implementations, experiments, and comparisons, we feel that we have also been able to demonstrate sufficient understanding and appreciation of this particular machine-learning approach.

References

[1] Q. Zhang and S. A. Goldman, "Em-dd: An improved multiple-instance learning technique," in *Advances in neural information processing systems*, 2001, pp. 1073–1080.

[2] T. G. Dietterich, R. H. Lathrop, and T. Lozano-Pérez, "Solving the multiple instance problem with axis-parallel rectangles," *Artificial intelligence*, vol. 89, no. 1, pp. 31–71, 1997.

[3] R. H. Byrd, P. Lu, J. Nocedal, and C. Zhu, "A limited memory algorithm for bound constrained optimization," *SIAM Journal on Scientific Computing*, vol. 16, no. 5, pp. 1190–1208, 1995.

[4] O. Maron and T. Lozano-Pérez, "A framework for multiple-instance learning," *Advances in neural information processing systems*, pp. 570–576, 1998.

[5] S. R. Cholleti, S. A. Goldman, and R. Rahmani, "Mi-winnow: A new multiple-instance learning algorithm," in *2006 18th IEEE International Conference on Tools with Artificial Intelligence (ICTAI'06)*. IEEE, 2006, pp. 336–346.

[6] C. Cortes and V. Vapnik, "Support-vector networks," *Machine learning*, vol. 20, no. 3, pp. 273–297, 1995.

[7] J. Wang and J.-D. Zucker, "Solving multiple-instance problem: A lazy learning approach," 2000.

[8] Ragav Venkatesan, "Synthetic Dataset for MIL," https://github.com/ragavvenkatesan/np-mil/tree/master/data.

[9] UCI Machine Learning Repository, "MUSK1 Data Set," https://archive.ics.uci.edu/ml/datasets/Musk+(Version+1).

[10] ——, "MUSK2 Data Set," https://archive.ics.uci.edu/ml/datasets/Musk+(Version+2).

[11] S. Andrews, I. Tsochantaridis, and T. Hofmann, "Support vector machines for multiple-instance learning," *Advances in neural information processing systems*, vol. 15, pp. 561–568, 2002.