# Group 3 Writeup for Multiple Instance Learning

December 10, 2021

Ningjia Huang, Tianxi(Sherry) Zhao, Richard Chen, Jingyu(Nora) Tang, Liyuan Huang, Alvin Kong, Golnoush Asaeikheybari [1]

Abstract

Our group looked into Multiple Instance Learning(MIL) problem, a supervised learning task based on training a set of bags where each bag contains multiple feature vectors. Each group member implemented either 1 or 2 algorithms which belongs to 3 main categories: instance-space paradigm, bag-space paradigm, and embedded-space paradigm. The performance of each algorithm and each method is evaluated on 4 datasets. The work allocation is shown in the following table.

| | Algorithm1 | Paper | Algorithm2 | Paper |
|---|---|---|---|---|
| Ningjia Huang | MILES | [3] | CCE | [12] |
| Tianxi(Sherry) Zhao | EMDD | [6] | miVLAD | [9] |
| Jingyu(Nora) Tang | Bag Dissimialrity | [4] | BARTMIP | [10] |
| Liyuan Huang | Citation-kNN | [8] | Bayesian-kNN | [8] |
| Alvin Kong | IM kernel | [2] | | |
| Richard Chen | MICA | [7] | APR | [5] |
| Golnoush Asaeikheybari | MIGraph | [11] | | |

Table 1: Work allocation for the MIL project.

[1] Each person contributes fairly to the project.

# 1 Introduction

Multiple Instance Learning (MIL) is a variant of supervised learning which receives a considerable amount of research attention due to its applicability in real world such as drug acitivity prediction, image classification, text documents classification, etc. In traditional supervised learning scenario, each example is represented by a feature vector where each feature vector has an associated class label. In MIL, we learn a classifier based on a training set of bags, where each bag contains several feature vectors (called instances) and has an associated label. Therefore, MIL provides a framework to handle the scenarios where class labels are naturally associated with sets of samples instead of individual samples. Note that not all of the instances in the bags are necessarily relevant. Some of the instances in a bag may not convey any information about that bag.
Amores [1] proposed an exhaustive taxonomy of different kinds of MIL methods according to how the information in the MI data is exploited. He summarized the existent algorithms into 3 categories: Instance-Space (IS) paradigm, Bag-Space (BS) paradigm and Embedded-Space (ES) paradigm as shown in Fig 1. In IS paradigm, the discriminative information lies at the instance-level so that a discriminative instance-level classifier is trained to seperate the instances in positive bags from those negative ones. In BS paradigm, the discriminative information lies at the bag-level so that the learning process discriminates between entire bags. In ES paradigm, each bag is mapped to a single feature vector that summarizes the information about the whole bag.

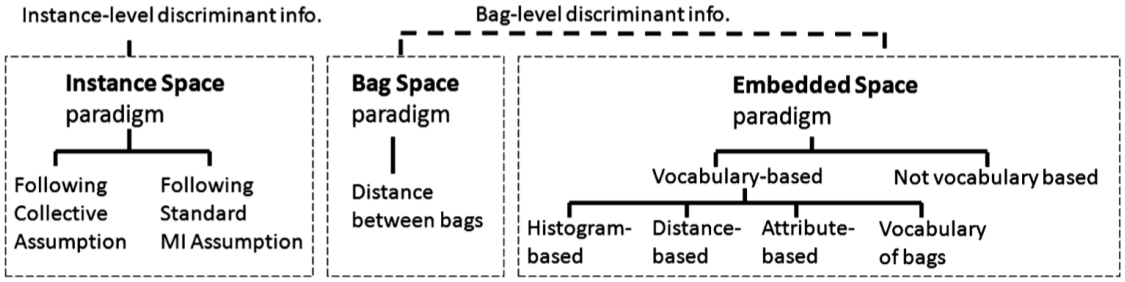



Figure 1: Taxonomy of MIL methods proposed in [1].

In this group project, the group members implemented 12 MIL algorithms, which are:

- Instance-Space Paradigm: MICA, APR, EMDD;
- Bag-Space Paradigm: Bayesian kNN, Citation kNN, IM Kernel, MIGraph;
- Embedded-Space Paradigm: BARTMIP, miVLAD, MILES, CCE, Bag Dissimilarity;

and compared the performance of them on the Musk1, Elephant, Fox, and Tiger datasets.

# 2 Background

MIL was introduced in the context of drug activity prediction by Dietterich et al., which intended to foretell the potency of candidate drug molecules by analyzing a collection of previously synthesized molecules whose potencies have already been tested.

The potency of a drug molecule is determined by the degree to which it binds to the target protein. However, as a molecule can adopt a wide range of shapes by rotating some of its internal bonds and the strength of binding is determined by the 3D structure of a drug, it is hard to decide whether a molecule can bind to a target protein by using the standard supervised learning paradigm. In the MIL problem, a training bag is a set of same molecules with different shapes and a label of binding or not binding is attached to the training bag. The goal is to learn the concept of binding and therefore predict whether a new bag (that is, a new molecule) is able to bind to the target protein.

Another real-world application that highlights the importance of MIL is the image classification task. For example, given a set of images, we are asked to identify the ones represent the beach. An image that is identified as "beach" must contain two elements, the sand and the sea, and no other visual contents such as mountains, trees are necessary for this task. Yet if it only displays the sand element, it could be a desert; if it only displays the sea, it could be just the sea. Neither could be treated as a beach. A visual demonstration is given in Fig 2. The general procedure for image classification is first to extract a collection of regions in the image, and for each region, it is described by a feature vector. Therefore, the image is represented by a bag $X = \{x_1, x_2, ..., x_N\}$, where $N$ is the number of regions extracted and $x_i$ is the feature vector describing the $i$th region of the image [1].

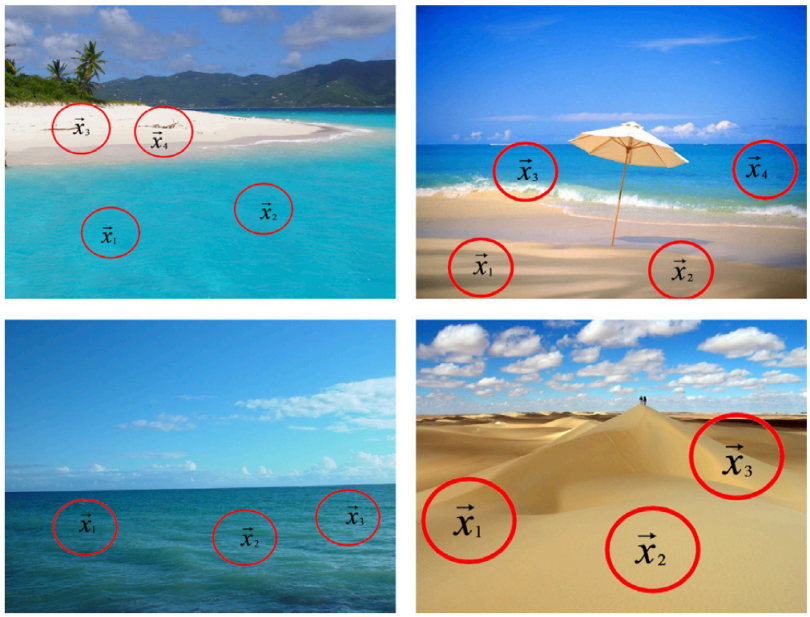



Figure 2: Images of beaches (top row) and non-beaches (bottom row) [1].

Drug activity and image classification are just two examples that show the significance of MIL. There are way more tasks where MIL plays an important role, including econometrics analysis, audio processing, etc.

# 3 Individual Report

# Nora Tang's Individual Report

December 10, 2021

# 1 Bag Dissimilarities

## 1.1 Overview

[2] transforms each bag into a dissimilarity vector $d(B_i,T) = [d(B_i,B_1), d(B_i,B_2), ...., d(B_i,B_M)]$, where M stands for the size of the training set, and $d$ is the dissimilarity measurement. In this paper, they proposed several dissimilarity measurements including Dmeanmean, Dmeanmin, Dminmin and so on. At first, the training set and the testing set are transformed into a matrix calculated by the dissimlarity measurements. Then, the data is fed into a supervised learner for classification.

## 1.2 Implementations

I implemented three distance methods *DMeanmean, DMinmin, and DMeanmin* using Python and Numpy, which are defined as follows:

$$dmeanmean(B_i,B_j) = \frac{1}{n_i n_j}\Sigma_{k=1} n_i \Sigma_{l=1} n_j d(x_{ik},x_{jl}) \tag{1}$$

$$dminmin(B_i,B_j) = \min_k \min_l d(x_{ik},x_{jl}) \tag{2}$$

$$dmeanmin(B_i,B_j) = \frac{1}{n_i}\Sigma_{k=1} n_i \min_l d(x_{ik},x_{jl}) \tag{3}$$

, where all of $d$ are squared euclidean distances.

After computing distances of all bags, the algorithm outputs a dissimilarity matrix as discussed above. Then the transformed representation of the training bags can be fed into supervised learners such as Logistic Regression and SVM with Linear Kernel(**sklearn**).

## 1.3 Result

The algorithm is tested with Linear SVM using 10-fold cross validation, and I used **svm** and **metrics** from **sklearn**. In order to be consistent with the performance evaluation with other algorithms, here I used the mean of selected metrics $\pm$ its standard deviation. The result is shown in Table 1.

Dmeanmin has better performance overall. This is probably because all the datasets the bag as a whole is a more discriminative source of information than a particular instance [2]. Thus, Dmeanmin is able to capture the relevant information from most instances to determine the bag label.

Table 1: Performance Evaluation for Bag Dissimilarities (SVM linear kenel).

| | | Dmeanmean | Dminmin | Dmeanmin |
|---|---|---|---|---|
| Musk1 | Accuracy | 81.2 ± 14.164 | 83.9 ± 8.244 | 90.3 ± 8.945 |
| | Recall | 83.7 ± 14.478 | 85.0 ± 18.93 | 88.6± 14.938 |
| | Precision | 82.8 ±19.294 | 86.2 ± 12.911 | 93.1± 13.95 |
| | AUC | 92. 9 ± 8.231 | 96.1 ± 3.711 | **98.6 ± 2.766** |
| Musk2 | Accuracy | 77.4 ± 12.24 | 89.3 ± 6.801 | 81.10 ± 18.185 |
| | Recall | 61.8 ±35.186 | 68.8 ± 31.379 | 76.9 ± 31.881 |
| | Precision | 69.69 ± 38.137 | 86.0 ± 31.048 | 78.4 ± 35.238 |
| | AUC | 92.10 ± 9.74 | 91.8 ± 10.48 | **92.9 ± 8.23** |
| Fox | Accuracy | 53.5 ± 11.192 | 58.0 ± 8.124 | 65.0 ± 8.66 |
| | Recall | 43.7 ± 44.672 | 52.5 ± 16.122 | 77.4 ± 26.923 |
| | Precision | 31.6 ± 27.596 | 59.0 ± 12.274 | 60.3 ± 14.484 |
| | AUC | 65.5 ± 8.719 | **65.9 ± 6.782** | 65.1 ± 11.012 |
| Elephant | Accuracy | 80.0 ± 8.062 | 80.0 ± 8.66 | 81.5 ± 6.727 |
| | Recall | 80.6 ± 16.03 | 81.8 ± 17.93 | 84.5 ± 15.525 |
| | Precision | 81.8 ± 14.912 | 81.90 ± 8.806 | 82.8 ± 13.691 |
| | AUC | 89.5 ± 8.201 | 91.8 ± 6.021 | **93.8 ± 5.206** |
| Tiger | Accuracy | 69.5 ± 10.828 | 69.0 ± 9.165 | 76.0 ± 14.799 |
| | Recall | 68.4 ± 19.796 | 71.8 ± 13.967 | 77.2 ± 17.164 |
| | Precision | 76.0 ± 18.138 | 69.1 ± 16.042 | 80.2 ± 19.664 |
| | AUC | 80.3 ± 8.562 | 76.5 ± 8.349 | **82.3 ± 10.129** |

# 2 BAMIC and BARTMIP

## 2.1 Overview

[3] first uses a K-mediods clustering algorithm, *BAMIC*, to cluster the training bags into $k$ groups. The distance used here is **Average Hausdorff** to overcome the sensitivity to outliers of Maximum Hausdorff and Minimum Hausdorff. Then each bag is represented as a vector of distance from each mediod $[d(TBag,M_1),d(TBag,M_2),...,d(TBag,M_k)]$, where $M_i$ stands for the $i$th mediod found by *BAMIC*, and $d$ stands for the distance method. Then the data could be fed into any supervised learners for classification.

## 2.2 Implementations

The definition of Average Hausdorff between two bags is:

$$AveH(A,B) = \frac{\Sigma_{a\in A}\min_{b\in B}||a-b|| + \Sigma_{b\in B}\min_{a\in A}||b-a||}{|A|+|B|} \tag{4}$$

, where $|\cdot|$ measures the cardinality of a set, and $||a-b||$ is calculated by Euclidean distance.

BAMIC will first cluster the training bags into $k$ groups, and outputs the groups and corresponding centers. Then each bag will be represented as a vector of distances from each center as discussed above. The transformed training bags will then be fed towards some supervised learners, such as SVM with Gaussian Kernel and K Nearest Neighbors, which are both included in **sklearn**.

## 2.3 Result

The BARTMIP algorithm is tested with K-nearest-neighbor ($k = 3$ as suggested by the paper) using 10-fold cross validation, and I used **KNeighborsClassifier** and **metrics** from **sklearn**. Here, $\mu$ stands for the percentage of the size of the training bag as the number of clustering centers. In order to be consistent with the performance evaluation with other algorithms, here I used the mean of selected metrics $\pm$ standard deviation. The result is shown in Table 2.

The table shows that the number of mediods does not significantly affect the performance.

# 3 Extensions

The extensions are tested with ten-fold cross validation on four datasets: Musk1, Fox, Elephant, Tiger. I excluded Musk2 because there are too many extensions to run and Musk2 is very time-consuming. The metrics I used here are Area Under ROC and Accuracy.

## 3.1 Manhattan Distance (Both algorithms)

From the observation, the data set is consisted of high-dimensional data points. In a high dimensional space, there exists *Curse of Dimensionality*[1], which suggests Manhattan distance is more suitable in calculating the distance between two high-dimensional data points. Hence, besides the Squared Euclidean Distance, I also implemented Manhattan Distance[1] for performance evaluation for both algorithms as shown in Table 3 and the second column of Table 4.

[1]The Manhattan Distance extension code is in **util.py** and **Distance.py**.

Table 2: Performance Evaluation for BARTMIP(AveH) (KNN with $k = 3$).

| | | $\mu = 0.2$ | $\mu = 0.4$ | $\mu = 0.6$ | $\mu = 0.8$ | $\mu = 1.0$ |
|---|---|---|---|---|---|---|
| Musk1 | Accuracy | 85.8 ± 7.232 | 88.0 ± 6.041 | 84.9 ± 15.557 | 84.9 ± 15.557 | 85.9 ± 9.889 |
| | Recall | 93.8 ± 9.605 | 93.8 ± 9.943 | 85.0 ± 30.231 | 98.3 ± 5.0 | 95.8 ± 8.539 |
| | Precision | 82.3 ± 11.774 | 84.3 ± 15.85 | 78.7 ± 30.158 | 77.2 ± 12.428 | 79.9 ± 14.205 |
| | AUC | 92.4 ± 6.882 | 96.0 ± 4.286 | 87.5 ± 19.249 | 91.6 ± 9.69 | 94.7 ± 6.716 |
| Fox | Accuracy | 54.0 ± 12.0 | 57.0 ± 7.81 | 52.5 ± 10.782 | 56.0 ± 5.831 | 59.5 ± 6.874 |
| | Recall | 56.0 ± 12.243 | 57.4 ± 16.535 | 54.0 ± 16.97 | 52.4 ± 13.911 | 61.8 ± 11.169 |
| | Precision | 54.4 ± 19.45 | 57.7 ± 12.442 | 52.7 ± 11.112 | 56.4 ± 13.765 | 61.8 ± 16.406 |
| | AUC | 55.5 ± 13.718 | 62.6 ± 9.624 | 55.2 ± 11.442 | 57.6 ± 6.601 | 63.4 ± 10.361 |
| Elephant | Accuracy | 74.0 ± 9.695 | 75.0 ± 7.416 | 80.5 ± 8.789 | 75.5 ± 11.927 | 80.5 ± 9.341 |
| | Recall | 79.2 ± 15.624 | 79.6 ± 13.695 | 81.8 ± 12.054 | 73.7 ± 18.59 | 82.4 ± 12.551 |
| | Precision | 72.2 ± 11.769 | 73.1 ± 9.307 | 80.5 ± 13.783 | 75.4 ± 10.811 | 79.4 ± 8.084 |
| | AUC | 81.7 ± 7.553 | 81.7 ± 9.582 | 84.1 ± 7.372 | 80.5 ± 8.433 | 86.7 ± 8.794 |
| Tiger | Accuracy | 65.0 ± 10.724 | 67.5 ± 12.698 | 65.5 ± 11.715 | 73.0 ± 8.124 | 68.5 ± 10.5 |
| | Recall | 63.5 ± 11.98 | 65.8 ± 17.386 | 60.6 ± 22.4 | 72.6 ± 13.354 | 70.0 ± 11.71 |
| | Precision | 66.1 ± 12.545 | 66.1 ± 15.673 | 67.1 ± 20.816 | 70.8 ± 13.474 | 70.0 ± 18.084 |
| | AUC | 70.1 ± 7.016 | 70.8 ± 14.165 | 70.7 ± 12.048 | 78.3 ± 9.382 | 75.7 ± 10.209 |

Table 3: Performance Evaluation for Bag Dissimilarity with Manhattan distance.

| | | Euclidean | | | Manhattan | | |
|---|---|---|---|---|---|---|---|
| | | Dmeanmean | Dminmin | Dmeanmin | Dmeanmean | Dminmin | Dmeanmin |
| Musk1 | Acc. | 81.7 ± 13.4 | 85.9 ± 8.55 | 91.4 ± 9.376 | 83.9 ± 14.378 | 84.9 ± 8.595 | 90.6 ± 11.41 |
| | AUC | 91.6 ± 8.232 | 96.9 ± 5.548 | 96.1 ± 8.259 | 95.6 ± 8.59 | 93.5 ± 7.292 | **97.0 ± 7.208** |
| Fox | Acc. | 54.0 ± 7.681 | 59.5 ± 13.124 | 63.0 ± 10.296 | 58.5 ± 10.966 | 60.0 ± 10.0 | 65.5 ± 9.341 |
| | AUC | 64.4 ± 12.281 | 64.4 ± 14.433 | 65.4 ± 10.196 | 72.7 ± 12.882 | 67.2 ± 12.238 | **73.6 ± 9.282** |
| Elephant | Acc. | 77.0 ± 13.266 | 81.0 ± 7.0 | 82.5 ± 7.826 | 88.0 ± 8.124 | 79.0 ± 11.358 | 85.0 ± 9.22 |
| | AUC | 89.8 ± 8.459 | 89.8 ± 8.152 | 88.1 ± 8.863 | **94.2 ± 7.284** | 87.1 ± 11.511 | 93.6 ± 7.279 |
| Tiger | Acc. | 63.0 ± 11.0 | 67.5 ± 12.5 | 77.5 ± 6.801 | 79.5 ± 8.201 | 79.0 ± 10.677 | 81.0 ± 8.307 |
| | AUC | 78.8 ± 8.44 | 75.1 ± 14.201 | 80.8 ± 8.14 | 87.9 ± 7.396 | 84.3 ± 9.086 | **90.1 ± 5.829** |

Table 4: Performance Evaluation for BARTMIP KNN with different distances.

| | | AveH with Euclidean | AveH with Manhattan | Dmeanmean | Dminmin | Dmeanmin |
|---|---|---|---|---|---|---|
| Musk1 | Acc. | 88.0 ± 6.041 | 89.1 ± 10.907 | 86.7 ± 14.741 | **90.2 ± 7.45** | 84.8 ± 12.313 |
| | AUC | 96.0 ± 4.286 | **96.9 ± 5.548** | 91.9 ± 5.901 | 93.6 ± 7.398 | 92.4 ± 5.549 |
| Fox | Acc. | 59.5 ± 6.874 | 59.0 ± 8.0 | 57.0 ± 9.539 | **61.0 ± 9.695** | 56.5 ± 5.5 |
| | AUC | 63.4 ± 10.361 | **64.1 ± 12.452** | 57.7 ± 10.186 | 62.6 ± 7.337 | 59.0 ± 8.377 |
| Elephant | Acc. | 80.5 ± 9.341 | 79.0 ± 7.681 | 78.0 ± 5.568 | **82.0 ± 8.426** | 79.5 ± 10.356 |
| | AUC | 86.7 ± 8.794 | **86.9 ± 6.171** | 82.9 ± 9.042 | 83.7 ± 9.31 | 84.4 ± 5.469 |
| Tiger | Acc. | 73.0 ± 8.124 | 77.5 ± 9.287 | 70.0 ± 12.042 | 75.0 ± 5.477 | **79.5 ± 8.5** |
| | AUC | 78.3 ± 9.382 | 83.6 ± 8.289 | 76.1 ± 8.531 | 81.8 ± 5.556 | **84.9 ± 6.335** |

Even though the mean of AUC under Manhattan is a bit higher than Euclidean, the overall performances are not statistically significant. It's probably because we are looking into how dissimilar the bags are. Additionally, Euclidean distance calculates the shortest path, which is easier to interpret than Manhattan. Overall, it would not hurt using Euclidean distance.

## 3.2 Distance measurement (BARTMIP)

As the first paper points out, rather than requiring a distance metric like Maximum Hausdorff, non-metric distances could also reveal information, and it proposes several distance methods. Therefore, I also implemented these distances in the BARTMIP algorithm. Additionally, I evaluated AveH with Manhattan distance mentioned in the previous section. The result is shown in Table 4.

In the other three distance measurement, Dminmin has better performance than AveH regarding accuracy across all datasets. This is probably because during clustering process, Dminmin is more biased compared to AveH since it only considers the shortest distance, which could be beneficial in the MIL clustering algorithm.

## 3.3 Instance Selection

My selected algorithms talked about the bag-space and embedded-space methods at bag level. Their assumption is that all the instances in the same bag contribute necessary information for the bag label. In other words, they do not specifically pick out the most representative instances from the bag. In both papers, they propose some ways to calculate distance to overcome the shortcomings of sensitivity to the outliers of maximum and minimum Hausdorff distances. From this perspective, I pondered what if we selected the most representative instances in the bag and then transform the representation of the bag. Thus, I came up with the idea of Instance Selection.

A bag is labeled positive indicates that it has at least one positive instance. For example, in MUSK dataset, an instance is positive if the shape of the molecule is correct. In Fox/Elephant/Tiger, an instance is positive if it contains correct descriptive region of the animal. If the instance is negative, then none of its features contain correct information.

Thus, we can conclude that if an instance is labeled positive then it must include the target concept.

Suppose $t$ is the target concept, and $T$ is the concept produced by the algorithm, then the probability that the instance $X_i = x_{i1}, x_{i2}, ..., x_{in}$ is positive, where $x_{ij}$ represents the $j$the feature and $n$ represents the number of features (assuming all features are conditionally independent):

$$Pr(X_i|T = t) = 1 - Pr(x_{i1}, ..., x_{in}|T \neq t) = 1 - \Pi_{j=1}^{n} Pr(x_{ij}|T \neq t) \tag{5}$$

Because we only need to look into some regions of the data to determine whether it's positive or not. Suppose only $k$ features are in the target concept and all the others are irrelevant. Then we only need to calculate the probability of those $k$ features in the given example that is not in the target concept. Therefore, equation 5 can be simplified as:

$$Pr(X_i|T = t) = 1 - \Pi_{l=1}^{k} Pr(x_{il}|T \neq t) \tag{6}$$

,where $x_i l$ is the feature discussed previously. My thought for the next step is that we can further use estimations derived from Bayes Formula to maximize the probability. Therefore, for each instance, we would obtain an estimate, and then we can select the $k$ instances with highest estimate ranking and then use the result to transform the representation of the bag.

Since I stuck in the previous step, I did not implement this extension. Hopefully my descriptions have thoroughly expressed my ideas.

## 3.4 Feature Selection and Dimensionality Reduction (Bag Dissimilarity)

As we can observe from the statistics from each dataset (Musk1, Fox, Elephant and Tiger), they all have high-dimensional features. At the step of fitting into different learners after the transformation, we have to take every feature into account to extract the information. Such an approach with high-dimensional data points will incorporate unnecessary information and may converge slowly. Hence, selecting a number of features or reducing the dimension of the feature space could be useful in picking concise information of the dataset. Therefore, I would like to test *K-best* using mutual information regression and *PCA* dimensionality reduction methods [2].

I imported Select-K-Best and PCA from sklearn and AUC is used to evaluate the performance.

### 3.4.1 Select K-Best

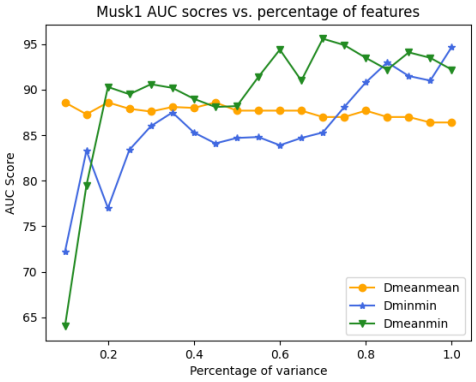
Here, the result of the cross validation is fixed. To see the performance of different number of features, I tested it from 10% to 100% with step size 5%, where the percentage means how many features to keep. The result of Bag Dissimilarity is shown in Fig 1.

From the observation, training on all features will not always achieve the best performance, which means the relevant information is captured by a subset of transformed features. Thus, it would be sensible to select a subset of transformed features before fitting into supervised learners.

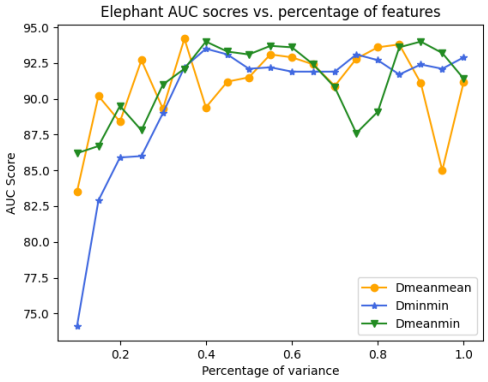
### 3.4.2 PCA

Here, the result of the cross validation is fixed. Additionally, before the bag dissimilarity matrix being fed to the linear SVM, it is first scaled to zero mean and one standard deviation. To see the performance of different number of components, I tested it from 95%

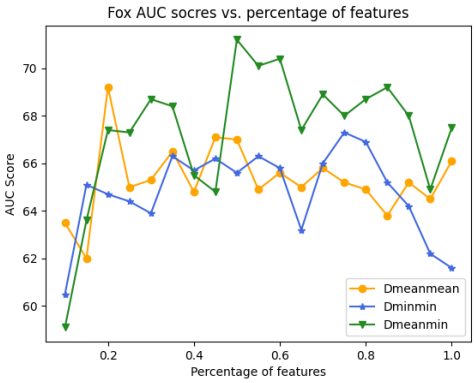
[2]The extension code is in **FeatureSelection.py**



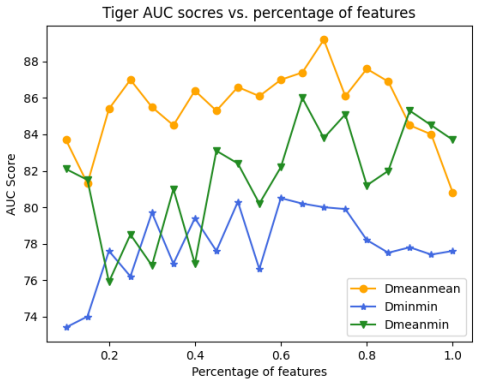

(a) MUSK1 dataset.




(b) Elephant dataset.

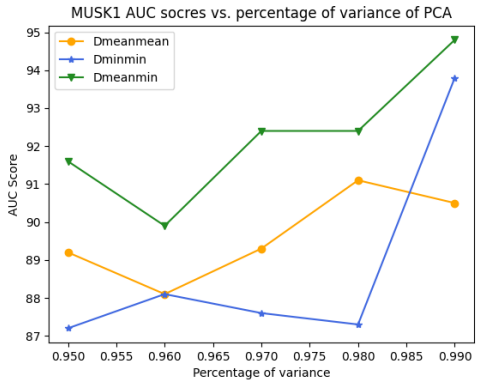



(c) Fox dataset.

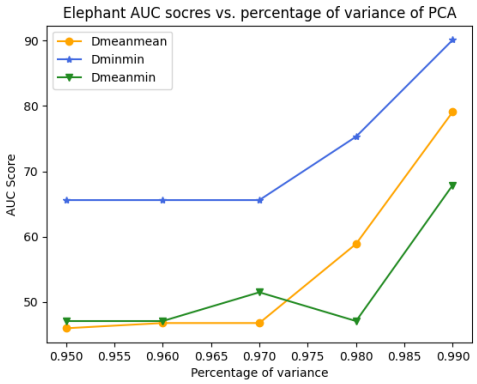



(d) Tiger dataset.

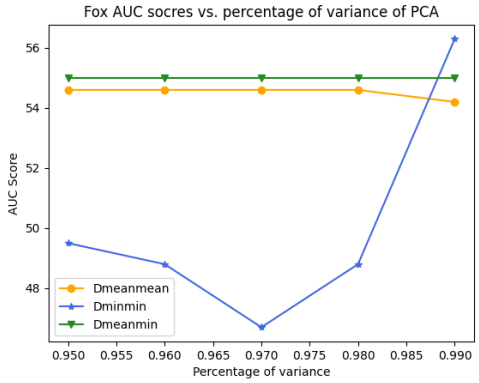
Figure 1: Bag Dissimilarity K-Best analysis for the four datasets.



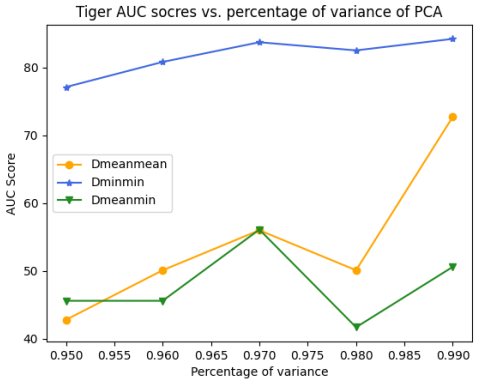

(a) MUSK1 dataset.




(b) Elephant dataset.




(c) Fox dataset.




(d) Tiger dataset.

Figure 2: Bag Dissimilarity PCA analysis for the four datasets.

to 99%, where the percentage sets up a threshold of how much variance to explain. It will then select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified by *n_components*. The result of Bag Dissimilarity is shown in Fig 2.

From the observation, *dminmin* achieves the best performance with 99% threshold across all datasets. This could be caused by the characteristics of Dminmin. Because Dminmin tends to find the shortest path from a bag to another as shown in Figure 3 while Dmeanmin and Dmeanmean tends to decrease the impacts from the outliers. Ideally, the distance is small for two bags with same labels, and large with different labels. With a high threshold, PCA captures the most discriminative information among positive and negative bags.

# 4 Conclusion

Overall, the runtime of Bag Dissimilarity is much faster than BARTMIP since it only calculates the distance between bags. And the performance of both algorithms is not statistically significant. Yet they both have bad performance on the Fox dataset. It's because the selected dissimilarity representation, K-medoids clustering and the embedded distance calculation are not suitable choices for Fox. The results of extensions are not statistically different as well. Future work should investigate the statistics of different datasets and then

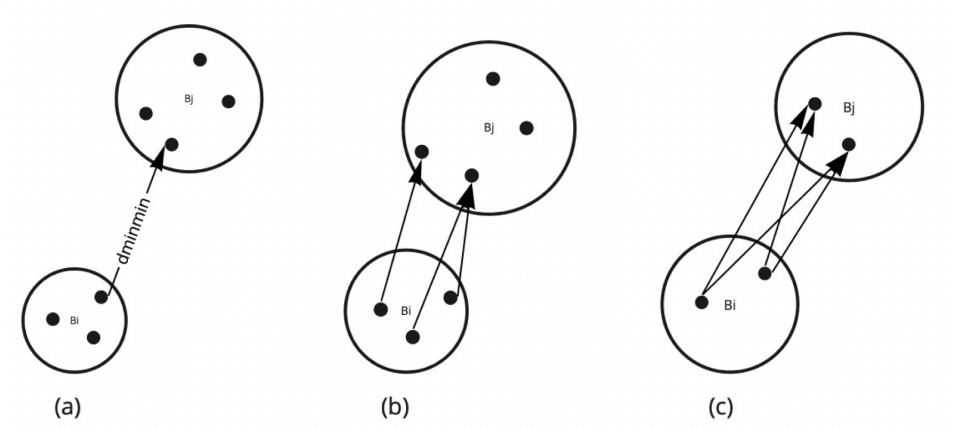



Figure 3: (a) Illustration of Dminmin. (b)Illustration of Dmeanmin. (c) Illustration of Dmeanmean.

design more suitable algorithms.

# Appendix: Instructions on running algorithms

**Bag Dissimilarity**
**-distance**: Specify which distance to use (dmeanmean, dminmin, dmeanmin).
**–use-euc**: Use Euclidean distance. If it's turned off, the algorithm will use Manhattan distance.
**–is-musk**: Specify the type of dataset. If running MUSK1 or MUSK2, this should be turned on; if running on Elephant, Fox or Tiger, this should be turned off.
**Example**:
python3 Bag_Dissimilarity.py elephant.csv -distance "dmeanmean" –use-euc

**Bag Dissimilarity Feature Selection**
**-distance**: Same as above.
**-FS**: Specify which feature selection extension to use (KBEST, PCA).
**-n**: Specify number of components to use in feature selection extension (In PCA, it means how much variance to preserve; in KBEST, it means the percentage of features to select.).
**–use-euc**: Same as above.
**–is-musk**: Same as above.
**Example**:
python3 Bag_Dissimilarity_FS.py elephant.csv -distance "dmeanmean" -FS "KBEST" -n 0.1 –use-euc

**BARTMIP**
**-distance**: Specify which distance to use (dmeanmean, dminmin, dmeanmin, aveh).
**–use-euc**: Same as above.
**–is-musk**: Same as above.
**-mu**: Determine the number of mediods of the clustering.
**–is-musk**: Same as above.
**Example**:
python3 BARTMIP.py elephant.csv -distance "aveh" –use-euc -mu 0.2

# References


[1] Charu Aggarwal, Alexander Hinneburg, and Daniel Keim. On the surprising behavior of distance metric in high-dimensional space. *First publ. in: Database theory, ICDT 200, 8th International Conference, London, UK, January 4 - 6, 2001 / Jan Van den Bussche ... (eds.). Berlin: Springer, 2001, pp. 420-434 (=Lecture notes in computer science ; 1973)*, 02 2002.

[2] Veronika Cheplygina, David M.J. Tax, and Marco Loog. Multiple instance learning with bag dissimilarities. *Pattern Recognition*, 48(1):264–275, Jan 2015.

[3] Min-Ling Zhang and Zhi-Hua Zhou. Multi-instance clustering with applications to multi-instance prediction. *Applied Intelligence*, 31(1):47–68, 2008.

# Liyuan Huang's Individual Report

December 10, 2021

# 1 Papers and Algorithms

## 1.1 Solving the Multiple-Instance Problem: A Lazy Learning Approach

The authors present two variants of the K-nearest negibor algorithm, called Bayesian-kNN and Citation-kNN. The distance formula used for both of the algorithms are minimal Hausdorff distance.

- **Minimal Hausdorff Distance**
  Given two sets of points, A = $\{a_1, ..., a_m\}$ and b = $\{b_1, ..., b_n\}$, the Hausdorff distance is defined as:
  $$H(A,B) = maxh(A,B), h(B,A) \tag{1}$$
  where h(A, B) = $max_{a\in A} min_{b\in B}||a-b||$. The distance metric that used in this paper is Minimal Hausdorff Distance, which the minimal one of the m individual point distances decides the value of the overall distance. Since:
  $$h_1(A,B) = min_{a\in A} min_{b\in B}||a-b|| = min_{b\in B} min_{a\in A}||b-a|| = h_1(B,A) \tag{2}$$
  Thus, the distance metric formula is:
  $$H(A,B) = h_1(A,B) = h_1(B,A) \tag{3}$$
  Additionally, the distance function used to calculate the distance is Euclidean Distance.

- **Bayesian-kNN**
  Since the majority vote approach does not always predict correct, Bayesian method provides a probabilistic approach that calculate explicit probabilities for hypotheses. For each hypothesis c that the class of b can take, the posterior probabilities of c is p(c—{ $c_1, c_2...c_k$}). By Bayes theorem, the maximally probable hypothesis is:
  $$argmax_c p(c|\{c_1, c_2...c_k\}) = argmax_c \frac{p(\{c_1, c_2...c_k\}|c)p(c)}{p(\{c_1, c_2...c_k\}} = argmax_c p(\{c_1, c_2...c_k\}|c)p(c) \tag{4}$$
  where $p(\{c_1, c_2...c_k\}$ is constant in calculation, so that it can be ignored. One of advantage of this algorithms is the not expensive computational cost, since the maximal number of combination that k trained bag is k+1.

- **Citation-kNN**
  Another adaptation of kNN algorithm is Citation-kNN. Suggesting by its name, the method is based on references and citers. The notion of citation take not only into account the neighbors of a bag b(according to the Hausdorff distance), which is the

**references**, but also the bags that count b as a neighbor, which is the **citers**.
Assume that there are $R_p$ positive bags and $R_n$ negative bags for the R-nearest references, and $C_p$ positive bags and $C_n$ negative bags for the C-nearest citers. Let p = $R_p$ + $C_p$, n = $R_n$ + $C_n$, if p > $n$, then the class of b is predicted as positive; otherwise, the class of b is predicted as negative. [1]

## 1.2 A fuzzy citation-kNN algorithm for multiple instance learning

- **Fuzzy-Citation-kNN**
For each instance $x_i$ of the training dataset, the $k_{init}$ nearest neighbors value are assigned by[2]:

$$\mu_c(x_i) = \begin{cases} 0.51 + (v_c/k_{init}) * 0.49 & \text{if c = } \omega \\ (v_c/k_{init}) * 0.49 & \text{otherwise} \end{cases} \tag{5}$$

where $v_c$ is the number of neighbors belonging to class c and $\omega$ is the original class label of $x_i$. To classify the class, all votes are added according to the following equation [2]:

$$\mu_c(b) = \frac{\sum_{i=1}^{K} \mu_c(X_i)(1/||b-x_i||^{2/(m-1)} + \sum_{j=1}^{n_c(b)} \mu_c(X_j)(1/||b-x_j||^{2/(m-1)})}{\sum_{i=1}^{K}(1/||b-x_i||^{2/(m-1)} + \sum_{j=1}^{n_c(b)}(1/||b-x_j||^{2/(m-1)})} \tag{6}$$

# 2 Implementations

The language I used to implement my algorithms are Python.

- **Preprocess datasets**
Since our group used two kinds of dataset, musk1, and animal dataset(fox, tiger, elephant datasets), I wrote two preprocess methods to store the list of bags that store instances that belong to the same bag in a list. In the process, **pandas** library is imported which reads the .csv and .svm files; **re** library is imported which split the dataframe value by ':'; and **numpy** is imported as well.

- **Minimal Hausdorff Distance**
Followed by the formula in section 1.1, **euclidean from scipy.spatial.distance** is imported to calculate the distances.

- **Leave one out cross validation**
As suggested by the paper, the validation method used is leave one out cross validation.

- **Prediction of Bayesian-kNN**
The number of k nearest neighbors are selected and sorted by the Minimal Hausdorff Distance. Then I calculated the p($\{c_1, c_2 ... c_k\}$| c) * p(c), where c $\in$ {positive, negative}. If p(positive) > p(negative), then the bag is considers as positive; otherwise, it is considered as negative.

- **Prediction of Citation-kNN**

  - **R-nearest References** For each test bag, the number of R nearest References are selected and sorted by calculating the distance between test bag with each of train bags. Then I have $R_p$ and $R_n$.

Table 1: Citation-kNN prediction results for different R and C

| Musk 1 | R=1, C=1 | R=2, C=2 | R=3, C=3 | R=4, C=4 |
|---|---|---|---|---|
| Accuracy | 0.859 | **0.880** | 0.870 | 0.870 |
| Precision | 0.854 | **0.89** | 0.818 | 0.830 |
| Recall | 0.872 | 0.872 | **0.957** | 0.936 |
| Elephant | R=1, C=1 | R=2, C=2 | R=3, C=3 | R=4, C=4 |
| Accuracy | 0.725 | 0.755 | 0.760 | **0.785** |
| Precision | 0.696 | **0.780** | 0.710 | 0.761 |
| Recall | 0.80 | 0.71 | **0.88** | 0.83 |
| Fox | R=1, C=1 | R=2, C=2 | R=3, C=3 | R=4, C=4 |
| Accuracy | 0.575 | 0.575 | 0.620 | **0.655** |
| Precision | 0.584 | 0.647 | 0.613 | **0.725** |
| Recall | **0.52** | 0.33 | 0.65 | 0.50 |
| Tiger | R=1, C=1 | R=2, C=2 | R=3, C=3 | R=4, C=4 |
| Accuracy | 0.705 | 0.670 | **0.755** | 0.725 |
| Precision | 0.753 | **0.854** | 0.807 | 0.869 |
| Recall | 0.61 | 0.41 | **0.67** | 0.53 |

– **C-nearest Citers** The finding of C-nearest Citers are more complex. I first build a distance matrix of train bags. Then I append the calculated distances between test bag with each of train bags to the distance matrix. Then sort the distance matrix. If the distance of train bag appears in the k nearest Citers, append the specific train bag to the list of citers. After looping through all train bags, I found the list of C-Nearest-Citers.

# 3 Experiment

## 3.1 Citation-kNN

From the table 1, we can see that the result of the Musk1 dataset performs better than the animal datasets. There are two possible reasons: the first one is that Musk1 dataset contains less features and less bags, the second one is that the feature value ranges of each instance of Musk1 dataset are larger than the feature value ranges of animal dataset since Musk1 dataset is not normalized, while animal datasets are normalized.
Additionally, compared the musk1 prediction results with the papers' prediction results, our results are slightly lower than the papers' results, while the accuracy differences are around 0.03. The differences may be caused by different leave one out cross validation method used, or the dataset used by the paper authors is normalized. With the small differences, we can still conclude that our results of Musk1 dataset follows the general trend of the paper's results.

Table 2: Bayesian-kNN prediction for different K

| Musk 1 | K = 1 | K = 2 | K = 3 | K = 4 | K = 5 |
|---|---|---|---|---|---|
| Accuracy | 0.859 | **0.88** | 0.826 | 0.815 | 0.761 |
| Precision | 0.84 | **0.909** | 0.897 | 0.895 | 0.879 |
| Recall | **0.894** | 0.851 | 0.745 | 0.723 | 0.617 |
| Elephant | K = 1 | K = 2 | K = 3 | K = 4 | K = 5 |
| Accuracy | **0.745** | 0.740 | 0.72 | 0.73 | 0.675 |
| Precision | 0.725 | 0.773 | 0.833 | 0.859 | **0.872** |
| Recall | **0.79** | 0.68 | 0.55 | 0.55 | 0.41 |
| Tiger | K = 1 | K = 2 | K = 3 | K = 4 | K = 5 |
| Accuracy | **0.705** | 0.67 | 0.635 | 0.615 | 0.58 |
| Precision | 0.753 | 0.854 | 0.886 | **0.925** | 0.944 |
| Recall | **0.61** | 0.41 | 0.31 | 0.25 | 0.17 |
| Fox | K = 1 | K = 2 | K = 3 | K = 4 | K = 5 |
| Accuracy | 0.575 | 0.575 | **0.58** | 0.53 | 0.515 |
| Precision | 0.584 | 0.647 | **0.735** | 0.688 | 0.667 |
| Recall | **0.52** | 0.33 | 0.25 | 0.11 | 0.06 |

### 3.2 Bayesian-kNN

From the table 2, we can see that the dataset Musk1 also has the best performances among the other three datasets. The performance of Bayesian-kNN is slightly lower than the performance of Citation-kNN as well, which is consistent with the paper's results.

## 4 Research Extension

### 4.1 Citation-Bayesian-KNN

This algorithm combines the algorithm of Citation-kNN and Bayesian-KNN. It finds the R nearest referencers, C nearest citers, then use Bayes theorem and the Bayesian-kNN's classification rule to derive the class of the test bag. The formula becomes:
$argmax_c p(c|\{c_1, c_2...c_R\}) * p(c|\{c_1, c_2...c_C\})$
$= argmax_c p(\{c_1, c_2...c_R\}|c)p(\{c_1, c_2...c_C\}|c)p(c)^2$
The prediction results can be found in Table 3. Compared to the results of Citation-kNN, the results of this algorithm for all of four datasets are similar with the results of Citation-kNN when R=1=C, and R=2=C; the results of this algorithms are slightly lower than the results of Citation-kNN when R=3=C, R=4=C. Another difference is that the Citation-kNN performs better with larger value of R and C, while Citation-Bayesian-kNN performs better with smaller value of R and C.
One possible explanation is that Citation-kNN predicts more accurately with larger number of citers and references, while Bayesian approach not work well with the larger number of neighbors since it will be affect more by the labels of the neighbors.
The new algorithm has the similar and slightly better results by comparing with the prediction results of Bayesian-kNN. The most of the prediction results of Fox and Tiger datasets

Table 3: Citation-Bayesian-kNN prediction for different Rs and Cs

| Musk 1 | R=1, C=1 | R=2, C=2 | R=3, C=3 | R=4, C=4 |
|---|---|---|---|---|
| Accuracy | 0.859 | **0.880** | 0.859 | 0.837 |
| Precision | 0.854 | **0.891** | 0.925 | 0.921 |
| Recall | 0.872 | 0.872 | **0.787** | 0.745 |
| Elephant | R=1, C=1 | R=2, C=2 | R=3, C=3 | R=4, C=4 |
| Accuracy | 0.725 | **0.755** | 0.74 | 0.73 |
| Precision | 0.696 | 0.78 | 0.824 | **0.857** |
| Recall | **0.80** | 0.71 | 0.61 | 0.54 |
| Fox | R=1, C=1 | R=2, C=2 | R=3, C=3 | R=4, C=4 |
| Accuracy | 0.575 | 0.575 | **0.58** | 0.53 |
| Precision | 0.589 | 0.647 | **0.735** | 0.688 |
| Recall | **0.53** | 0.33 | 0.25 | 0.11 |
| Tiger | R=1, C=1 | R=2, C=2 | R=3, C=3 | R=4, C=4 |
| Accuracy | **0.705** | 0.67 | 0.635 | 0.615 |
| Precision | 0.753 | 0.854 | 0.886 | **0.912** |
| Recall | **0.61** | 0.41 | 0.31 | 0.27 |

are the same of those two algorithms respectively. One possible explanation is that the R nearest referencers of the Citation-kNN are the same as the K neareast neighbors in Bayesian-kNN, which dominate the prediction results of Citation-Bayesian-kNN in Fox and Tiger datasets.

## 4.2 Citation kNN with feature selection

As suggested by the authors, feature selection is a promising way to improve the accuracy and decreases the running time. Thus, I implemented the Citation-kNN with feature selection by using library **SelectPercentile and mutual info classif from sklearn.feature selection**. Due to running time consideration, I picked Musk1 data set(R=1=C=1) to test the impact of accuracy with different percentile. Here are the results.

| Musk1 data set | mutual info classif | time used |
|---|---|---|
| Accuracy (percentile = 40) | 0.837 | 8 min 34 s |
| Accuracy (percentile = 60) | 0.859 | 10 min 4 s |
| Accuracy (percentile = 80) | 0.851 | 13 min 6 s |

Then I picked the percentile of 60 and had the prediction results in table 4. (Since the tiger dataset has one bag that contains only one instance, it cannot be tested with mutual info classif.)
Given table 4, this algorithm has lower performances on Musk1 and Elephant dataset, while it has similar performances on Fox dataset. However, the advantage of feature selection is the saving of time – it saved about 1/3 of time to get the result of citation-kNN, with more than 3 hours running time of Citation-kNN. Consider the saving time, the lower but not much performances of citation-kNN with feature selection is acceptable.

Table 4: Citation-kNN with feature selection prediction for different Rs and Cs

| Musk 1 | R=1, C=1 | R=2, C=2 | R=3, C=3 | R=4, C=4 |
|---|---|---|---|---|
| Accuracy | 0.75 | 0.761 | 0.761 | **0.772** |
| Precision | 0.707 | **0.791** | 0.719 | 0.75 |
| Recall | **0.873** | 0.723 | 0.872 | 0.83 |
| Elephant | R=1, C=1 | R=2, C=2 | R=3, C=3 | R=4, C=4 |
| Accuracy | 0.59 | **0.665** | 0.66 | **0.655** |
| Precision | 0.565 | 0.646 | 0.607 | **0.80** |
| Recall | 0.78 | 0.73 | **0.91** | 0.62 |
| Fox | R=1, C=1 | R=2, C=2 | R=3, C=3 | R=4, C=4 |
| Accuracy | **0.575** | 0.52 | 0.53 | 0.51 |
| Precision | **0.584** | 0.522 | 0.531 | 0.51 |
| Recall | **0.52** | 0.26 | 0.51 | 0.36 |

# 5 References

[1] J. Wang, J.-D. Zucker, Solving the multiple-instance problem: A lazy learning approach, in: Proc. of International Conference on Machine Learning, 2000, pp. 1119–1125.
[2] D. Ghosh, S. Bandyopadhyay, A fuzzy citation-kNN algorithm for multiple instance learning, 2015 IEEE International Conference on Fuzzy Systems (FUZZ-IEEE), 2015, pp. 1-8, doi: 10.1109/FUZZ-IEEE.2015.7338024.

# Sherry Zhao's Individual Report

December 10, 2021

I implemented miVLAD and EM-DD algorithms for multiple instance learning. The algorithms are implemented using python. Numpy is imported for dealing with arrays, sklearn is imported for split training and testing data, calculating the evaluation metrics, and classification with svm. The datasets are in .csv files.

# 1 miVLAD

## 1.1 Implementation

miVLAD is a embedded-based, vocabulary-based multiple instance learning method. According to [2], in miVLAD, the algorithm use K-means clustering method to find K centroids, all the instance $b_{ij}$ are assigned to the nearest centroids in vocabulary, and the centroids are stored in vocabulary $C = \{c_1, ..., c_k, ..., c_K\}$, where i indicates the $i^{th}$ bag in the dataset, j indicates the $j^{th}$ instance in a bag, and k indicates the $k^{th}$ attribute of the centroid.
Then a mapping function M(B, C) = $\vec{\mathbf{v}}$ is used to map the instances to a K-dimensional feature vector, where B is a bag and V is vocabulary. $\vec{\mathbf{v}}_i = \{v_{i1}, ..., v_{ik}, ...v_{in}\}$ is the new feature vector obtained from the mapping function, and each bag is represented in a $\vec{\mathbf{v}}_i$. To map each bag to a feature vector, the function is:

$$v_{ikl} = \sum_{b_{ij} \in \{b_{ij} | NN(b_{ij}) = c_k\}} b_{ijl} - c_{kl}$$

$b_j - c_k$ indicates the likelihood of an instance belonging to the $c_k$ class. $v_{ikl}$ represents the $l^{th}$ attribute mapped to the $k^{th}$ centroid in v for bag i. $b_{ijl}$ represents the $l^{th}$ attribute in $j^{th}$ instance of bag i, and $c_{kl}$ represents the $l^{th}$ attribute in the kth centroid.
Because some of the attributes can be so large that the significance of other attributes is reduced, which is called burstiness. In order to get an accurate prediction, miVLAD needs to reduce burstiness in data. Intra normalization is used to reduce the burstiness, in the miVLAD algorithm in this project, the feature vector is normalized by sign square root (SSR) followed by $l_2$-nomalization:

$$SSR: \ \mathbf{v}_i \leftarrow sign(\mathbf{v}_i)\sqrt{\mathbf{v}_i}$$

$$l_2 - nomalization: \mathbf{v}_i \leftarrow \frac{\mathbf{v}_i}{||\mathbf{v}_i||_2}$$

After obtaining the normalized feature vector, we use a supervised classifier SVM to train a model. The training set is T = where v is the feature vector we get from the mapping function, $y_i$ is the label of training bag $B_i$. The training function can be represented as:

$$f(X) = g(M(X,V))$$

where f is miVLAD algorithm and g is svm classification.
When predicting a bag, we map the bags to the vocabulary we learned in the training process to find feature vectors, normalize the feature vectors with SSR and $l_2$-nomalization, then put them into the SVM to predict the label of each bag.

## 1.2 Result

| | musk1 | musk2 | tiger | elephant | fox |
|---|---|---|---|---|---|
| Accuracy | 0.821±0.078 | 0.776±0.085 | 0.822±0.044 | 0.828±0.065 | 0.613±0.069 |
| Precision | 0.795±0.085 | 0.735±0.149 | 0.800±0.062 | 0.853±0.049 | 0.598±0.071 |
| Recall | 0.897±0.100 | 0.820±0.133 | 0.860±0.061 | 0.832±0.110 | 0.721±0.091 |
| AUC | 0.825±0.075 | 0.797±0.085 | 0.824±0.046 | 0.836±0.061 | 0.617±0.068 |

From the result table, musk1, musk2, tiger, and elephant have similar accuracy, precision, recall, and auc than other dataset. The performance of miVLAD on fox dataset is lower than others. Two extensions are implemented for miVLAD.

## 1.3 Extension — Soft Assignment

In miVLAD, the vocabulary is essential for classification. The performance of clustering method influnce the performance of the entire algorithm because the classification is based on the vocabulary learned from clustering. This extension is trying to improve the performance of clustering. K-means is a hard assignment method. One instance is assigned to one centroid, and it's sensitive to noise in data. For extension of miVLAD, a soft assignment method can be used to cluster the instances, which allows each data to belong to more than one class. For some data that are overlapped between classes, fuzzy c-means clustering should perform bettern than k-means clustering. Fuzzy c-means clustering is implemented as the soft assignment method. The goal of fuzzy c-means clustering is to minimize:

$$J(\Gamma, V) = \sum_i \sum_j (\gamma_{ij}^m ||x_i - v_j||^2)$$

where v is the centroids, $\gamma$ is the membership values, m is the fuzziness index, and $||x_i - v_j||^2$ calculates the Euclidean distance. In fuzzy c-means clustering, k centoids are initiated and the

following functions are repeated until the functions converge:

$$v_j = \frac{\sum_i (\gamma_{ik}^m \cdot x_k)}{\sum_i \gamma_{ik}^m}$$

$$\gamma_{ij} = \left[ \sum_k \left( \frac{d_{ki}^2}{d_{kj}^2} \right)^{\frac{1}{m-1}} \right]^{-1}$$

where d is the Euclidean distance. Fuzzy c-means algorithm replaced k-means clustering in this extention to build the vocabulary, and here are the results:

| | musk1 | musk2 | tiger | elephant | fox |
|---|---|---|---|---|---|
| Accuracy | 0.817±0.061 | 0.787±0.068 | 0.843±0.057 | 0.850±0.048 | 0.604±0.071 |
| Precision | 0.765±0.069 | 0.820±0.181 | 0.833±0.076 | 0.863±0.075 | 0.598±0.087 |
| Recall | 0.922±0.078 | 0.623±0.182 | 0.888±0.054 | 0.841±0.081 | 0.698±0.114 |
| AUC | 0.824±0.069 | 0.764±0.071 | 0.843±0.056 | 0.851±0.047 | 0.609±0.063 |

The performance of miVLAD with soft assignemnt is similar to miVLAD with hard assignment. The change of accuracy of musk1 and fox are less then 1%, and though the increase of accuracy of musk2, tiger, and elephant is small. Because the change is too small, it is inconclusive that if the soft assignemnt can improve the performance a little or it cause by other reasons like getting better training set during cv split.
From the two performance table for miVLAD, it shows that in most of the experiment, the recall is higher than precision, which meeans that it is easier for miVLAD to give false positive than false negative.

# 2 EM-DD

## 2.1 Implementation

Expectation-Maximization Diverse Density (EM-DD) is an instance-based multiple instance learning method. EM-DD is based on the SMI assumption, which states that there is at least one positive instance in a positive bag, and there is no positive instance in a negative bag. A point is chosen to be the target concept, and the target concept should be a positive point. According to [1], DD is the measure of the distance of positive bags and negative bags to the target concept point. DD makes the assumption that the bags are conditionally independent. DD is defined by:

$$DD(t) = \frac{P(\mathbf{B}|y=t)P(y=t)}{P(\mathbf{B})}$$

where **B** is the bags, and y is the labels. To find the true concept, DD needs to be maximized. Because the prior probability $P(y=t)$ is uniform and constant and the probability of evidence $P(\mathbf{B})$ is also constant, maximizing likelihood can also maximize DD, applying the Baysian rule:

$$arg\max_x \prod_i P(x=t|B_i^+) \prod_i P(x=t|B_i^-)$$

where $B_i^+$ indicates a positive bag, and $B_i^-$ indicates a negative bag. The instances in positive bags should be close to the target concept and the instances in negative bags should be far away from the target concept. The most-likely-model is used to calculate the $\prod_i P(x=t|B_i^+) \prod_i P(x=t|B_i^-)$ term in above equation.The model is calculated by:

$$P(x=t|B_i^+) = arg\max_j P(x=t|B_{ij}^+)$$

$$P(x=t|B_i^-) = 1 - arg\max_j P(x=t|B_{ij}^-)$$

$P(x=t|B_i^+)$ is the probability of the instance being positive, and $P(x=t|B_i^-)$ is the probability of an instance being negative. Because there are two classes among the bags and assuming the probabilities of positive and negative bags are the same before we look at the attributes, the threshold is set to 0.5. If the highest probability among the instances in the bag is greater than or equal to 0.5, the bag is classified as positive, and if none of the instances in the bag has a probability greater than or equal to 0.5, the bag is classified as negative. Euclidean distance is used to calculate the distance between an instance and hypothesis, and the probability of the bag belongs to positive bag or negative bag can be represented by a Gussian-like distribution:

$$P(x=t|B_{ij}) = \exp(-\sum_k s_k^2 (B_{ijk} - x_k)^2)$$

where x is the target point vector, and s is the scaling vector for the target point vector. $B_{ijk}$ represents the $k^{th}$ attribute of the $j^{th}$ instance in the ith bag. Combining x and s, the target concept = $[[x_1,...,x_k],[s_1,...,s_k]]$, which is the vector that EM-DD algorithm looks for.

In the expectation step, the distances between each instance and the hypothesis are calculated, and the instance representing the bag the most is selected as a presentative instance, which means the instance closest to the hypothesis is selected in positive bags, and the instance most far away from the hypothesis is selected in negative bags.

In the maximization step, an optimal algorithm is needed to find the maximum DD. According to [3], finding the maximum DD equals finding the minimum of negative log DD (NLDD). Gradient descent is used to minimize the NLDD. EM-DD is very time consuming because gradient descent has to take one step at a time, so not all the instances in the training bags are used as a start point to find global minimum. Because gradient descent is likely to be stuck in local minimums, multiple start points are selected from the bags in the training data to help find the global minimum.

Here is the results of EM-DD algorithm using instances in difference number of largest positive bags as start points to classify Musk1:

| | all | 1 | 2 | 3 | 5 |
|---|---|---|---|---|---|
| Accuracy | 0.721±0.125 | 0.532±0.136 | 0.674±0.137 | 0.686±0.102 | 0.674±0.100 |
| Precision | 0.704±0.138 | 0.254±0.314 | 0.610±0.238 | 0.657±0.212 | 0.680±0.105 |
| Recall | 0.834±0.165 | 0.400±0.490 | 0.783±0.310 | 0.712±0.336 | 0.830±0.196 |
| AUC | 0.719±0.117 | 0.548±0.081 | 0.669±0.116 | 0.671±0.082 | 0.650±0.085 |
| Runtime | 763.13±0.281 | 114.448±107.816 | 170.878±103.739 | 259.90±133.93 | 438.890±240.396 |

From the experiment, the accuracy of EM-DD is very depend on the starting point of hypothesis. If a starting point near optimal is chosen, we can get high accuracy in prediction. If a starting point far away from optimal is chosen, it is less likely for the gradient cannot bring the hypothesis to the optimal. However, using all the positive points as starting points is too time comsuming for the algorithm. To balance the accuracy and time comsuming problem, instances in 3 largest bags in the training set is used as starting point of hypothesis.

## 2.2 Result

| | musk1 | tiger | elephant | fox |
|---|---|---|---|---|
| Accuracy | 0.686±0.102 | 0.672±0.121 | 0.675±0.056 | 0.545±0.078 |
| Precision | 0.657±0.212 | 0.493±0.327 | 0.600±0.218 | 0.507±0.206 |
| Recall | 0.712±0.336 | 0.559±0.371 | 0.672±0.240 | 0.585±0.219 |
| AUC | 0.671±0.082 | 0.669±0.119 | 0.664±0.072 | 0.547± 0.078 |

The overall performance of EM-DD is not as good as miVLAD. It is a weak classifier and its accuracy of most dataset is between 60% - 70%. According to experiments in [1], the accuracy of EM-DD should be higher. The reason caused the poor performance might be difference in parameters. If good parameters are used, like a more suitble learning rate or maximum iteration in gradient descent, the performance will probably be better. Because EM-DD is very time comsuming, trying for parameters will takes a large mount of time, I have only tried modifying the number of staring point for hypothesis as above.

## 2.3 Extension — Feature Selection

The feature selection method compute the ANOVA F-value by:

$$F = \frac{variation\,between\,sample\,means}{variation\,within\,the\,samples}$$

and choose the features with high f-values. Sklearn is used to calculate f-value and feature selection.

| | musk1 | tiger | elephant | fox |
|---|---|---|---|---|
| Accuracy | 0.679±0.126 | 0.660±0.118 | 0.653±0.068 | 0.543±0.070 |
| Precision | 0.616±0.117 | 0.615±0.235 | 0.607±0.218 | 0.532±0.075 |
| Recall | 0.820±0.242 | 0.637±0.251 | 0.517±0.284 | 0.582±0.106 |
| AUC | 0.669± 0.122 | 0.658±0.098 | 0.640±0.094 | 0.545± 0.069 |

Runtime of Musk1 is 219.64±98.873 seconds, which is slightly shorter than EM-DD algorithm without feature selection. The change of accuracy is not significant. For tiger, elephant, and fox, the difference between precision and recall is reduced, which means the probability of false positive and false negtive are reduced.

I haven't implemented other extension for EM-DD due to the limitation of time, but I have some ideas to improve the EM-DD algorithm. The biggest problem of EM-DD from my view is its runtime, it needs hours to learn a model, which is not efficient. I can think of two solutions for it. The first is to reduce the size of data, for example using PCA to reduce the dimension or use correlation to reduce the number of attributes to make the algorithm run faster. For the second method, because gradient descent uses first derivative test, the path to optimum can be zigzag and long, if a second derivative optimization algorithm is used, maybe the function can converge to the optimum faster, resulting in reducing the runtime.

# References


[1] L. Dong. A comparison of multi-instance learning algorithms. 2006.

[2] X. S. Wei, J. Wu, Zhou, and Z. H. Scalable algorithms for multi-instance learning. *IEEE transactions on neural networks and learning systems*, 24(4):975–987, 2016.

[3] Q. Zhang and S. A. Goldman. An improved multiple-instance learning technique. *Advances in neural information processing systems (pp. 1073-1080)*, 2002.

# 4 Performance Evaluation

## 4.1 Experiments Setup

10-fold cross validation is used to test the performance of algorithms. The data are split into 9 training sets and 1 testing set. The algorithms learn with the training sets and their performance are evaluated with testing data set, and this is repeated 10 times with random splits each time. The accuracy, precision, recall, and area under curve (AUC) scores are recorded for each algorithm for the purpose of performance evaluation.
Each algorithm is tested with Musk 1, Fox, Tiger, and Elephant dataset. Statistics of four datasets is shown in Table 2. Dataset Musk 2 is also used for the evaluation, but since Musk 2 contains a large amount of instances, it takes much more time for the algorithms to learn it. Some algorithms, such as EMDD, are very time consuming, it is not computationally feasible for the algorithms to finish running Musk2. As a result, Musk2 is only tested for some algorithms and the result is included in individual reports.

| | Features (Non-Zero) | Positive Bags | Negative Bags | Positive Instances | Negative Instances | Avg | Min | Max |
|---|---|---|---|---|---|---|---|---|
| Musk1 | 166 | 47 | 45 | 207 | 269 | 5 | 2 | 40 |
| Elephant | 230 (143) | 100 | 100 | 762 | 629 | 7 | 2 | 13 |
| Fox | 230 (143) | 100 | 100 | 647 | 673 | 6 | 1 | 13 |
| Tiger | 230 (143) | 100 | 100 | 544 | 676 | 7 | 2 | 13 |

Table 2: Statistics about four datasets.

## 4.2 Instance-Space

3 instance-space methods were implemented: MICA, APR, and EMDD. Their performance evaluation regarding accuracy and AUC is shown in Figure 3. We can see APR has relatively bad performance on image classification tasks (Elephant, Fox and Tiger). Given that APR was made only for the Musk data set, and that it is more primitive in its design by using simple feature bounds for classifying bags, this is not all that surprising. It follows that EMDD is the second best performing algorithm overall, as it uses more complex concepts such as k-nearest neighbors to re-distribute the instances to train and classify the instances. MICA gives the best performance across all the datasets. This also makes sense as rather than follow the Standard MI assumption that there is only one instance in a bag that makes it positive, MICA uses the weighted collective assumption in that all of the instances contribute in some way to the bag label. For images, it is useful to identifying a combination of segments being present in order to make a prediction, such as identifying water and sand to infer that the image shows a beach. This explains why MICA is able to vastly outperform the other IS algorithms with the images data sets.
Looking at Musk1, the IS methods do not perform well compared to BS and ES methods regarding AUC metrics. Given the fact that they are meant to mostly focus on the instances rather than the entire bag, this is not all that surprising. In addition, as the number of instances increase in each bag, the training time also increases. It was found that APR was able to train quickly due to its relatively simplistic nature, but the EMDD algorithm took longer to train, especially for the data sets that had more instances in them. The same thing happened for MICA, except it was much longer than both EMDD and APR, as it needs to solve two linear programs during training.

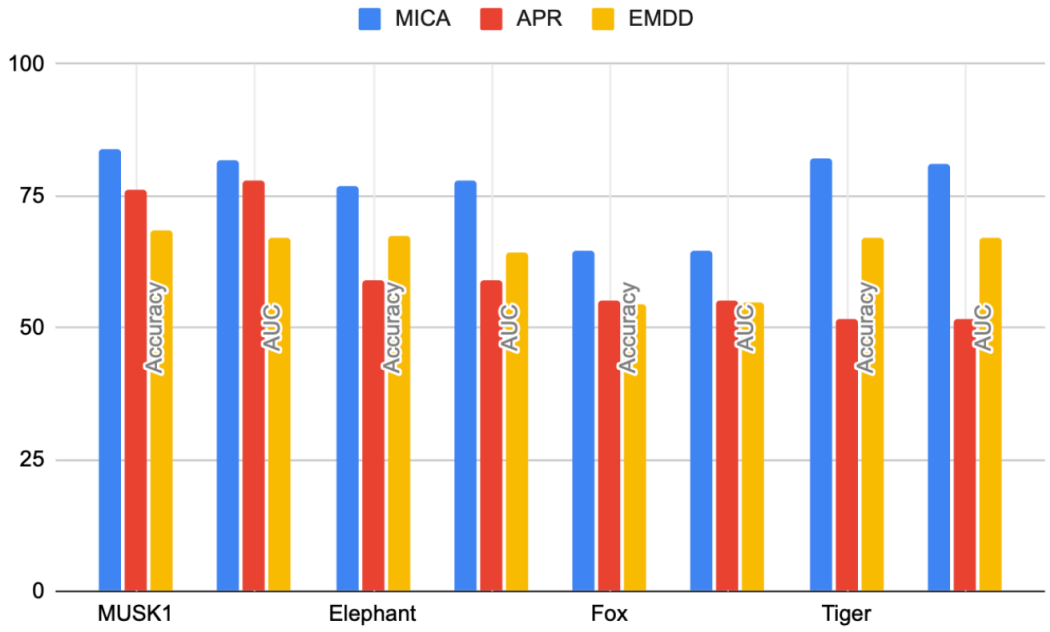



Figure 3: Accuracy and AUC comparison on IS methods.

## 4.3 Bag-Space

4 bag-space methods were implemented: MIGraph, Citation-kNN, Bayesian-kNN, and IM Kernel. Their performance evaluation regarding accuracy is shown in Figure 4. We can see that performance drops significantly on the Fox dataset. It is because BS methods depend on the defined distance. If the distance is not appropriately designed for the dataset, the overall performance will not be desirable.
Compared to other methods' prediction results on dataset Fox, Citation-kNN has the best performance. The possible explanation is that Citation-kNN not only takes account the R nearest Referencers but also takes account the C neareast Citers. This feature ensures that Citation-kNN will not be affected significantly as other methods if the instances value of bags are misleading. This is also the reason why it performs better than Bayesian-kNN. Additionally, Bayesian-kNN performs slightly worse than all of the other three methods. This is because Bayesian probabilistic approach is not suitable for kNN, since kNN only considers the neareast k neighbors. The result, which is the probabilities of being predicted as positive or negative, will be solely determined by the nearest neighbors, rather than the overall, or dominated neighbors. Overall, the performance of all the implemented BS methods is similar.

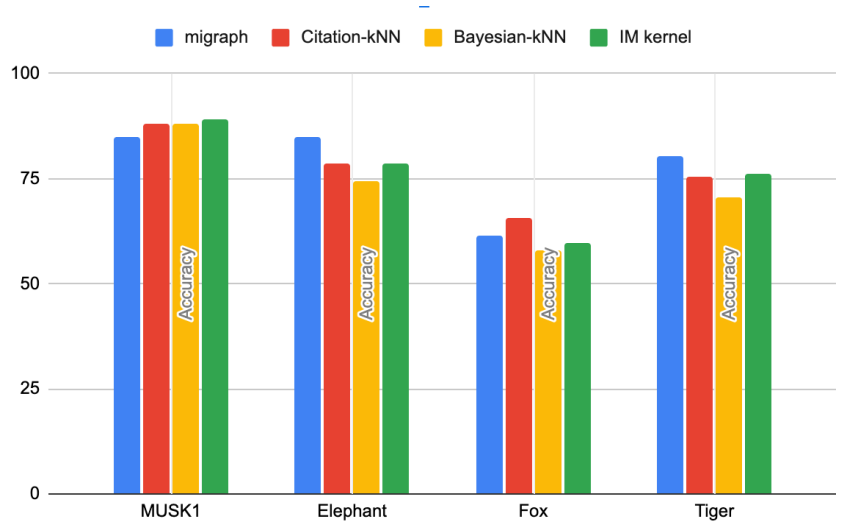



Figure 4: Accuracy and AUC comparison on BS methods.

## 4.4 Embedded-Space

5 embedded-space methods were implemented: Bag Dissimilarity, BARTMIP, MILES, CCE and miVLAD. Their performance evaluation is shown in Figure 5. We can see that CCE has bad performance on MUSK1 and Elephant compared to the others. This is because CCE relies on K-means to cluster the instances into $d$ clusters and use the clusters to build a new binary feature vector for each bag. As the dimensionality of the datasets is high, the resulting centroids vary significantly which induces unstable feature vectors. When KPCA was used for dimensionality reduction, the overall performance was improved.
Yet we can see again the performance drops dramatically on Fox dataset. Since ES methods are based on global, bag-level information, and the bag is represented by a summary of relevant information of the whole bag [1], the reason for the drop could be that the mapping functions are not appropriately designed for Fox dataset. Overall, ES methods other than CCE perform better than IS methods regarding AUC of MUSK1. This is because IS methods only look into local information while ES methods investigate global one. It is possible that a single molecule can have multiple shapes and the classification depends upon the exact shape of the molecule. The discriminative information could be hidden implicitly at a global level, and cannot be obtained at a local level.

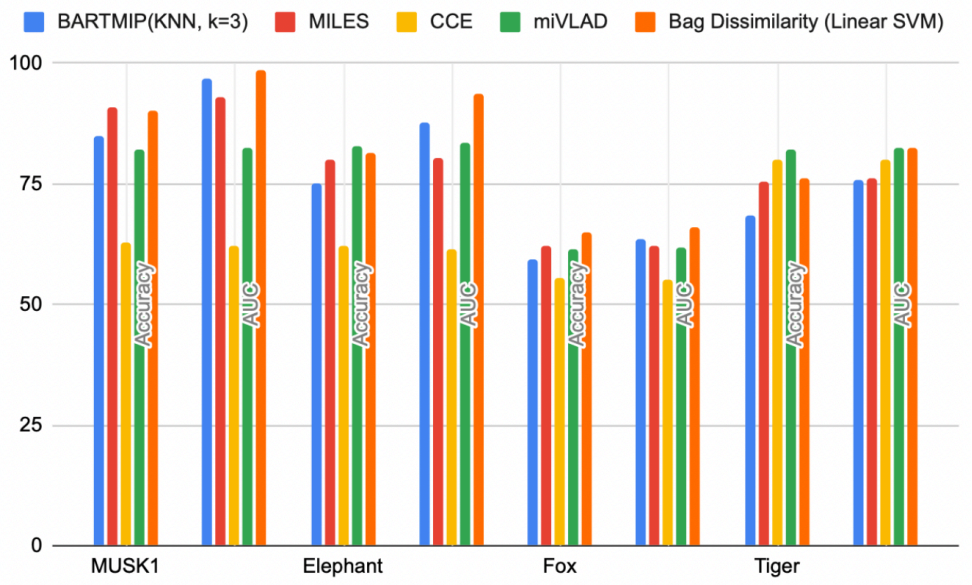



Figure 5: Accuracy and AUC comparison on ES methods.

# 5 Discussion

The overall accuracy and AUC scores for each method(including ES without CCE) is shown in Table ??. According to the table, BS has the highest accuracy and AUC among all three methods, but if we exclude CCE algorithm from the ES method, the overall score for ES method will be higher.

| | IS | BS | ES | ES(exclude CCE) |
|---|---|---|---|---|
| Overall Acc. | 67.24 | 75.89 | 74.01 | 76.09 |
| Overall AUC | 66.85 | 79.8 | 77.26 | 80.4 |

Table 3: Comparison of the Overall Result of 3 Different Methods

# 6 Conclusion

In this group project, our group implement 12 algorithms related to Multiple Instance Learning in Instance-Space, Bag-Space, and Embedded-Space categories after reading through related papers. The experiments are done on four datasets including MUSK1, and animal images. The results show that our implementation results basically followed the trend as the papers suggest. Overall, BS and ES methods outperform IS methods, suggesting that in MIL problems, it is more appropriate to consider global, bag-level information. Yet during the design phase of BS methods, the distance used to calculate similarity or dissimilarity between the bags should be carefully chosen when facing different kinds of datasets. It is also important for ES methods to notice the importance of the mapping regarding various datasets.
Additionally, we also completed several research extensions around our own algorithms. The detailed implementation and results of research extensions can be found in the group member's individual reports.

# References


[1] Jaume Amores. Multiple instance classification: Review, taxonomy and comparative study. Artificial Intelligence, 201:81–105, 2013.

[2] S. Boughorbel, J.P. Tarel, and N. Boujemaa. The intermediate matching kernel for image local features. In Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., volume 2, pages 889–894 vol. 2, 2005.

[3] Yixin Chen, Jinbo 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.

[4] Veronika Cheplygina, David M.J. Tax, and Marco Loog. Multiple instance learning with bag dissimilarities. Pattern Recognition, 48(1):264–275, Jan 2015.

[5] Thomas G. Dietterich, Richard H. Lathrop, and Tomás Lozano-Pérez. Solving the multiple instance problem with axis-parallel rectangles. Artificial Intelligence, 89(1-2):31–71, 1997.

[6] L. Dong. A comparison of multi-instance learning algorithms. 2006.

[7] O. L. Mangasarian and E. W. Wild. Multiple instance classification via successive linear programming. Journal of Optimization Theory and Applications, 137(3):555–568, 2007.

[8] Jun Wang and Jean-Daniel Zucker. Solving the multiple-instance problem: A lazy learning approach. In Proceedings of the Seventeenth International Conference on Machine Learning, ICML '00, page 1119–1126, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[9] X. S. Wei, J. Wu, Zhou, and Z. H. Scalable algorithms for multi-instance learning. IEEE transactions on neural networks and learning systems, 24(4):975–987, 2016.

[10] Min-Ling Zhang and Zhi-Hua Zhou. Multi-instance clustering with applications to multi-instance prediction. Applied Intelligence, 31(1):47–68, 2008.

[11] Zhi-Hua Zhou, Yu-Yin Sun, and Yu-Feng Li. Multi-instance learning by treating instances as non-i.i.d. samples. In Proceedings of the 26th Annual International Conference on Machine Learning, ICML '09, page 1249–1256, New York, NY, USA, 2009. Association for Computing Machinery.

[12] Zhi-Hua Zhou and Min-Ling Zhang. Solving multi-instance problems with classifier ensemble based on constructive clustering. Knowledge and Information Systems, 11(2):155–170, 2006.