# EECS 440: Final Project, Multiple Instance Learning

12/7/2019

David Kerrigan

Eric Miller

Kyle Pham

Anna Sedlackova

Gareth Valentin Li

Gary Yao

## 

## 

## Table of Contents

[**EECS 440: Final Project, Multiple Instance Learning**](#_ninyzbmo2oau) **1**

[Table of Contents](#_3m4zjk4pmdwl) 2

[**Part 1**](#_bf4ryk2pku4a) **5**

[Introduction](#_tl5h3yqw2qxb) 5

[Applications & Significance](#_o6okr3wtu3dl) 5

[**Part 2**](#_5wnjw4dqirzw) **7**

[Instance Space Algorithms](#_egf81z1ianlc) 7

[Axis Parallel Rectangles - David Kerrigan](#_8s4431mqn5ws) 7

[Data Format](#_j5evgwo1cbkb) 7

[Gaussian PDF for similarity calculation](#_mcdot8q3n02s) 8

[Classification:](#_wua1smg4icj9) 10

[Results](#_dxe5ajzie4h3) 11

[Some issues with implementation](#_9vv2qech6j2q) 13

[References](#_b2t1bw603rvf) 13

[Instance Space Algorithms - Gareth Valentin](#_80nx8enlyl6u) 15

[Diverse Density](#_47c7xnqanq8c) 15

[Expectation-Maximization version of Diverse Density](#_75qbq5ttkr2a) 17

[Results](#_641nbbqi6hpc) 17

[Discussion](#_t794aejijffi) 18

[References](#_13s3og7pycuj) 20

[Embedded Space Algorithms](#_h5bojn7jgod7) 21

[Embedded Space Distance Functions- Eric Miller](#_qjy3xaz66kji) 21

[Non-Vocabulary Based Methods](#_d09g8a6rgdin) 21

[Experiments](#_19fn7s6voror) 21

[Vocabulary Based Methods](#_kj78yfor530l) 22

[Histogram-Based Approaches](#_co5rsqkhuq3k) 23

[Hard Assignment](#_kwadqs59wpva) 23

[Soft Assignment](#_ozot5j74wyb) 23

[YARDS](#_p39smnn2l8hd) 24

[Implementation](#_swix3dlu6pya) 25

[Experiments](#_vwnxeqg12is1) 25

[Distance-Based Approaches](#_7t6lglkmq8b9) 27

[Implementation](#_r3qjapqu4mve) 27

[Experiments](#_ieo14jtj7bud) 28

[Attribute-Based](#_i6qphedfdz68) 28

[Implementation](#_a2yn7y2aty5k) 29

[Experiments](#_i2ianlrkernz) 29

[Methodology](#_6bi1kcobd872) 30

[Results and Discussion](#_6r9g9w3r6zfc) 30

[References](#_g4l8evxk5p9l) 32

[Embedded Space Clustering Algorithms- Anna Sedlackova](#_sja1a8m1xwlf) 33

[Introduction](#_itbkzcucs8p2) 33

[Use of clustering in the Vocabulary-Based Embedded Space (ES) Paradigm](#_xqvjc6yjkggs) 33

[Summary of Clustering Algorithms Investigated](#_f8z2exbtika7) 34

[k-means](#_33tlp3a6qy06) 34

[k-median](#_t6mukt2n5sl1) 35

[Mean shift](#_p8pc12aqewjn) 35

[Fuzzy c-means](#_1jbv4oa6t7jr) 35

[Methodology](#_p40qkaz36hru) 36

[Implementation](#_e41urhq4ll9p) 36

[Data Sets Used](#_dhuv252stmc9) 36

[Experimental Procedure](#_1n0tafuk5z8z) 36

[Number of cross-validation folds and iterations](#_48czkbu29b0p) 36

[Classifier parameters](#_wpki6sf5owtf) 37

[Number of clusters](#_16e40n1decdl) 37

[Determining other clustering parameters](#_spl912y1yd0g) 38

[Results](#_6e0gftropv7n) 39

[Discussion](#_9m35gcpzn2yq) 40

[Results interpretation & Relating to other studies](#_twvitslb06f) 40

[Comparing Clustering Algorithms](#_58psjzrhahm4) 40

[Comparing Distance Functions](#_sdlva18v32u8) 41

[Comparing Data Sets](#_19rtmrn7qx7d) 41

[Critical Difference Diagram](#_5ogk3jr92aag) 41

[Conclusion](#_f0x5ir95vsz) 41

[References](#_2vww2ngr1ccp) 42

[Bag Space Algorithms](#_ranjjd5w2z1d) 43

[Kyle Pham: Bag-Space paradigm (K-Nearest Neighbors algorithms)](#_t8a99zc252go) 43

[kNN Algorithm](#_59gor8wlzscm) 43

[Weighted-kNN Algorithm](#_69xu96yas78w) 45

[Citation-kNN Algorithm](#_g0bzmhfobego) 46

[Weighted-Citation-kNN](#_1ax2eu68ci40) 49

[Overall Dicussion](#_e4ecgx4bj8s) 51

[References](#_5rp566t9jur6) 51

[Gary Yao: Support Vector Machine for Multiple Instance Classification](#_uzpxfjpigngy) 53

[Bag Classification as a Support Vector Problem](#_9fgt9atoh1i6) 53

[Instance Learning: Reformulating the Objective Function](#_96sy7a7edjza) 54

[Bag-Space: Formulation as Kernel Tricks](#_1axjau9yb318) 55

[Embedded Space: Finding Vector Representations of Multiple Vectors](#_9zu2cz74h80l) 56

[Methods](#_bvczv416z6m4) 56

[mi-SVM & MI-SVM](#_u4tspvh8vtb8) 56

[SVM with Chamfer Distance and Hausdroff Distance](#_jiqjk3rfgyza) 57

[EMD+SVM](#_652stkbh4vvn) 57

[IMF SVM](#_550a6kt347fg) 57

[Results](#_gr5gi0qw2whk) 57

[References](#_g7wmq54ka9m7) 59

[**Part 3**](#_kcfl228527xo) **60**

[Results](#_q15v18seufrv) 60

[Comparative Study](#_ptdx0sl9i4so) 61

[Instance Based](#_wlt12bbd2jv) 61

[Bag Space](#_1ez4qdyta8l6) 62

[Embedded Space](#_pv5mml5toigl) 62

[Discussion](#_d5zq3mainxb) 63

[References](#_ptdx0sl9i4so) 64

# 

# Part 1

## Introduction

Multiple Instance Learning is a type of weakly supervised learning where labels are assigned to sets of samples or instances instead of samples themselves. Differing from Supervised Learning tasks, a Multiple Instance Learning Model considers multiple and varying numbers of instances in a “bag” and thus require models of additional flexibility. A bag is a set , where are the feature vectors or instances. Many tasks from Computer Vision to Biology with weakly labeled data and unstructured features naturally reduces to multiple instance learning tasks. As a result, a great number of studies have proposed algorithms to solve this classification task. Currently, there exist three main branches of Multiple Instance Learning, instance-space, bag-space, and embedded space. In this paper, we implemented and analyzed compared algorithms across the three main branches of Multiple Instance Learning and present our ideas of building a model that will solve the Multiple Instance Learning task.

We have decided to look at those specific data sets because we wanted to compare our findings to Amores et al. who used the MUSK-1 and MUSK-2 data sets which are drug activity prediction data sets. However, since MUSK-2 is 6598 instances, and took too long to run we have decided to look at the fox and elephant data sets instead which are content-based image retrieval and classification.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Database | Total Bags | Total Instances | Positive Bags | Negative Bags | Dimensions | Classes |
| MUSK1 | 92 | 460 | 47 | 45 | 160 | 2 |
| Fox | 200 | 1320 | 100 | 100 | 143 | 2 |
| Elephant | 200 | 1391 | 100 | 100 | 143 | 2 |

Table 1. Data sets information.

## Applications & Significance

MI Learning is used commonly used in two applications: drug activity prediction and image classification. These two problems require MI learning because the objects contain multiple feature vectors and a single class label, which would be difficult to solve with typical classifiers. Other applications mentioned in the paper include text classification, signal processing and economic prediction.

For drug activity prediction problems, the goal is to classify chemical molecules. Typically the system is trying to identify ‘good drugs’, which are judged on their ability to bind to a target site. This problem provides a significant challenge because molecules can typically take multiple shapes, of which only some or one may bind well. For the drug activity problem, bags typically hold feature vectors for all possible shapes a molecule can take.

The other popular application of MI learning is image classification, where the goal is to decide if an image belongs to a target class. For example, our *elephant* and *fox* files attempt to classify images that hold the respective animal as *True* and the rest as *False*. Image classification problems typically break the image into regions that are related to the target class. The regions are then converted into a feature vector describing the region. The bags in an image classification problem are a collection of feature vectors, with each vector describing a region in the image. The classifier then learns that inorder to classify an image as true, certain regions must be present.

# Part 2

## Instance Space Algorithms

Instance level classification algorithms rely on classifying each instance in the bag and aggregating the results into a single label. This is usually done with the SMI assumption: if any of the instances in the bag are positive, the bag is positive. If none of the instances in the bag are positive, the bag is negative. Instance level algorithms use different methods of aggregating instances into a single label and in normalization factor of the bag label [2]. For our project, we have implemented axis parallel-rectangles, diverse density, miSVM, and MISVM.

## Axis Parallel Rectangles - David Kerrigan

Axis parallel rectangles were the first proposed solution to the multiple instance classification problem. APR falls under the instance space group of algorithms for multiple instance learning. The paper, *Solving the Multiple Instance Problem with axis-parallel rectangles* by Diettrich, Lathrop and Perez [1], introduces two variants of APR learning. Both attempt to form a boundary in instance space that ideally contains no instances from negative bags and contains at least one instance from each positive bag. The assumption is that there exists a certain feature that results in the bag being positive. Classification then checks if the given bag has an instance in this training boundary. If any instance falls in the boundary, the bag is positive. Otherwise, the bag is negative [1].

### Data Format

The paper introduced APR for the purpose of solving musk drug classification. The datasets used were musk1 and musk2. Bags here are molecules. Each molecule has multiple conformations, each conformation is an instance. Molecules are either musks or non-musks. Data is continuous and are floating point values. For example, for a single molecule conformation take rays emanating from the center of the molecule and outwards in different directions. The attributes will be a ray, and where the ray intersects the surface of the molecule. A single conformation instance will be all the rays and their values [1]. For this project I used two data parsers made by Eric and Gary to produce data values for the animal image sets from a .svm file. In addition I used functions from our team’s statistics.py file for calculating precision and recall. I did not use the accuracy function as it didn’t work and instead wrote one; it was extremely simple.

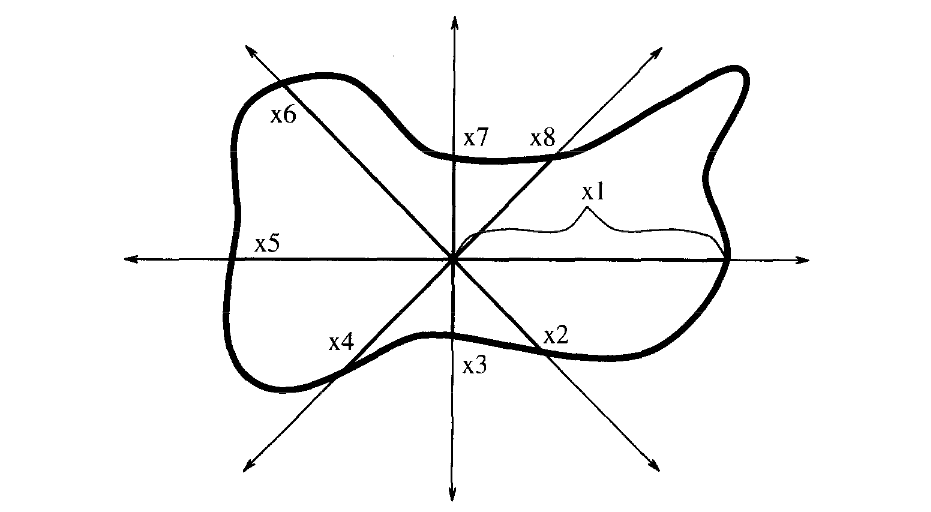


Figure 1: Molecule Conformation Data [1]

### Gaussian PDF for similarity calculation

The paper also introduces a method of finding commonality of attribute values using gaussian kernels. For the purpose of the APR algorithm we need to find how common a certain attribute value is within a certain molecule. For example, how common is it that x1=100 in molecule MUSK-288? We also seek to find how common this value is among all positive or negative molecules in our data set. To perform this probability calculation, for each attribute we place a gaussian function at each attribute value over the entire dataset. Then we normalize the function to construct a probability density function. To find the probability of a certain attribute value we input the value into the PDF for the corresponding attribute. Here is the exact gaussian function I used [1]:

The paper specified 𝜶 = 0.1. T is the domain of possible attribute values and i is the given attribute value [1].

I implemented two variations out of three of the APR algorithm. First, form a boundary in instance space that surrounds all instances belonging to positive bags. This is called the all-positive APR. This boundary will most likely contain some instances belonging to negative bounds. The figure shows the all-positive boundary as a solid rectangle. x1 and x2 are attributes. White points are members of a positive bag and black points are members of a negative bag. Shapes denote members of the same molecule, for example rectangles, circles, triangles, represent formations of the same molecule. However, the first APR algorithm does not take into account distinct molecules, whilst the second algorithm does [1].

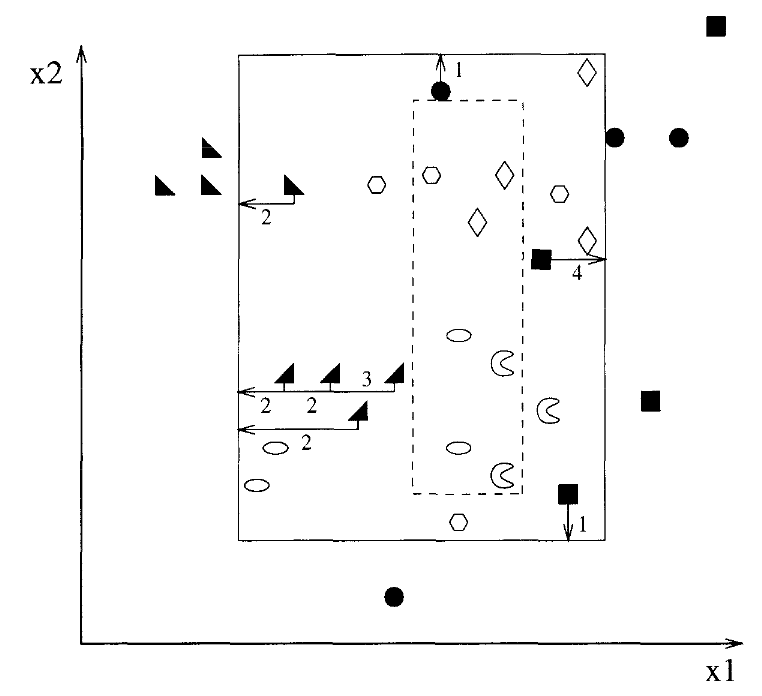


Figure 2: Instance Space in Two Dimensions [1]

The goal is to shrink the boundary as to remove all “negative instances” -- instances belonging to a negative bag whilst keeping as many “positive instances” -- instances belonging to a positive bag. To do this, for each attribute, label all values of “negative instances” with the number of “positive instances” that would be excluded were the APR to shrink to exclude this negative instance along this attribute. The APR can shrink either by decreasing the maximum bound or increasing the maximum bound. The numbers in the figure represent the cost of removing the negative instance. We then use a greedy algorithm to remove the least costly negative instance until we no longer include negative instances in our APR boundary [1].

The second option for shrinking the all-positive APR involves a more complicated cost calculation. For a positive instance mi,j -- instance j of molecule i -- depends on the other instances of mi as well as all other positive instances currently in the APR boundary. If among mi , there exist many similar instances to mi,j, the cost of exclusion will be low. Also, if among all the remaining positive instances in the APR boundary -- excluding mi,j -- all have similar feature values, then the cost of removing mi,j will be low. Here is the equation used [1]:

I used 𝛂 = 10.0 as instructed by the paper. Dd is the probability density function constructed from gaussian kernels. Vi means iteration over all values of molecule i along a single attribute [1].

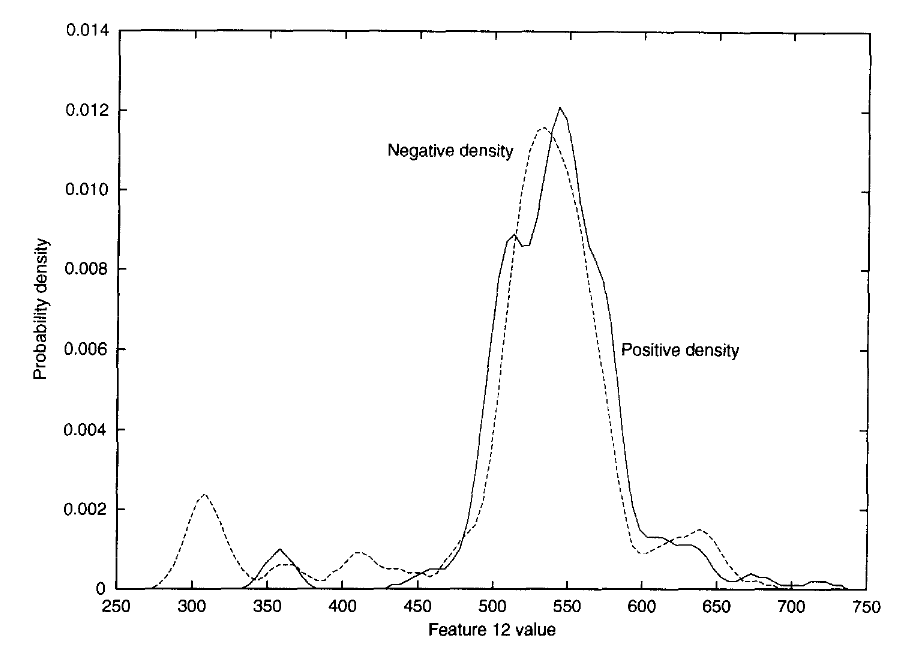


Figure 3: PDF of Attribute 12 [1]

Here is a given example PDF in the paper for a single musk feature. The graph is the sum of many different gaussian kernels placed at each occurrence of each attribute value. Given an inputted value for feature 12, this function will predict the probability of finding this value in the dataset; in other words, its commonality to other values in the data [1].

### Classification:

For instance space classification we use the equation [2]:

Here our aggregation operation () represents the or operator and Z = 1 [2].

Classifying a bag with APR is simple. Here’s the equation used [2]:

Where R is the shape defined by our APR boundary. Hence, the classification F(X) given bag X is [2]:

Which can be interpreted as simply [2]:

Note that this is a simplified version of the general instance space classification formula.

So if any of the instances in the bag are contained fully in the APR boundary, the bag is positive, else it is negative. In the instance space paradigm this is called the SMI assumption [2].

### Results

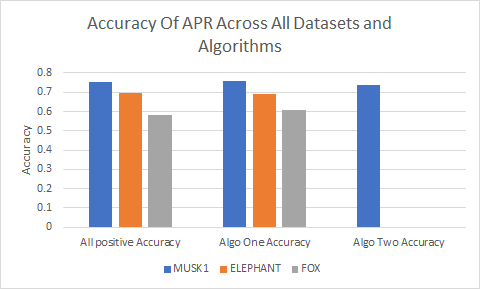


Figure 4: Accuracy Results for Each Dataset and Algorithm

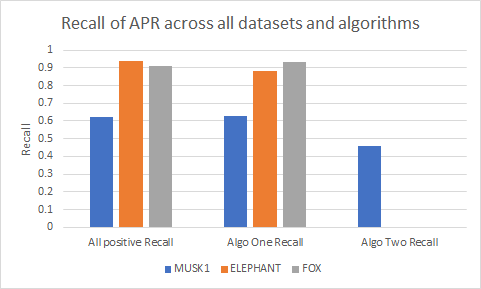


Figure 5: Recall Results for Each Dataset and Algorithm

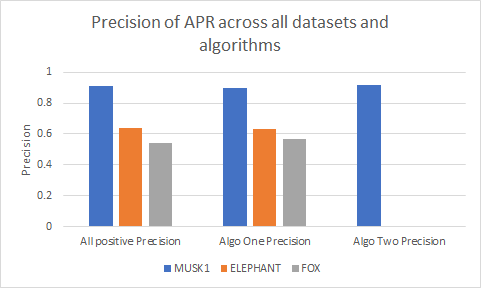


Figure 6: Precision Results for Each Dataset and Algorithm

Axis Parallel Rectangles seems to work reasonably well on the datasets, namely MUSK1. The datasets have a large number of features, which allows the APR boundary to be extremely specific. For example, with musk1, the APR has minimum and maximum boundaries over 163 dimensions. As a result, the all-positive APR is actually very accurate -- it contains few negative instances. The all-positive boundary performs comparably to algorithm one -- simple boundary reduction -- for all datasets across accuracy, precision, and recall. I assume this is because there is only a single negative instance needing to be pruned, and the cost of pruning it -- the number of positive instances removed from the boundary -- is comparable to leaving in place.

APR performs worse on the elephant dataset and worse still on the fox dataset. I think this is because of the quality of data. Almost half of the features are non-zero and many others have values extremely close to other attributes -- therefore the entire feature is essentially a copy of another feature. Also, the APR algorithms were designed with drug classification in mind [1], specifically the musk datasets, so it is not a surprise that it performs best on musk. One specific detail is the musk operates under the SMI assumption. If one conformation is positive for binding than the molecule is positive. However, other datasets may not follow this logical assumption for classifying bags. This may be one reason why fox and elephant performed worse for my instance level learning algorithms. Some of the algorithms that performed the best on the animal sets were embedded space algorithms not following the SMI assumption. However, my algorithms seemed to produce a higher recall on the animal datasets than on musk1.

The APR algorithm two -- which performs a more complicated cost calculation -- seems to perform comparably on the musk dataset to simple cost calculation. I attribute this to the fact that throughout the entire musk1 dataset, there is only one negative instance that falls within the all-positive boundary -- minimizing the effects of the different cost calculations.

### Some issues with implementation

The paper does not specify a few important details. Firstly, where does the boundary of the APR get drawn? The boundary is first drawn at the outermost positive molecules, and then set to outermost negative molecules as they are being removed. One can either assume a small margin to expand the boundary by, or leave the boundary intersecting instances. However, the second approach gives rise to the issue of classifying bags containing instances that fall on the APR boundary. Hopefully, this will not be much of an issue with folded datasets.

A second issue is the labelling of each negative instance’s cost for every iteration of shrinking the APR boundary. The paper specifies we must make these calculations every iteration for an optimal boundary, but for large datasets with many reductions I believe this will be inefficient. Why not label costs once and then reduce sequentially lowest costs until all negative instances have been excluded. The result will still be fairly accurate and run much more quickly. As the paper seems to specify the first procedure, this is what I used.

I was unable to run the APR algorithm two on larger datasets. Musk1 is fairly small and the iteration over all positive instances takes a short amount of time. In cost calculation for each instance, I had to iterate over each positive instance to find other members of the same molecule. With an expanding dataset the number of positive instances also expands, and reduction takes an unfeasible amount of time.

I attempted two methods of speeding up cost calculation. First are numpy array queries to return the set of instances of a relevant molecule. Second is to sort data into a nested numpy array and simply index this nested array to get the list of relevant instances. However, nested arrays took to long to render compared to rendering a simple array. I found that none of these measures resulted in any meaningful speedup and I was unable to run the data on sets like musk2.

### References

[1] T. Dietterich, R. Lathrop, and T. Perez. “Solving the Multiple Instance Problem with Axis Parallel Rectangles” *Artificial Intelligence*, vol. 89, 1997, pp. 31–71.

[2] J. Amores, “Multiple instance classification: Review, taxonomy and comparative study,” *Artificial Intelligence*, vol. 201, 2013, pp. 81–105.

## 

## Instance Space Algorithms - Gareth Valentin

### Diverse Density

Diverse Density (DD) and Expectation-Maximization version of Diverse Density (EMDD) are multiple instances learning algorithm that is instance space oriented. A bag is positive if at least one of the instances in the bag is positive. The diverse density algorithm pictures an n-dimensional space for n features where each point in the world space represents one hypothesis since multiple instances learning algorithms make the assumption that a bag is labeled negative if all the instances in the bag are negative. Assuming all the bags each as a collection of points in the world space, ideally, the intersection of points of all positive bags without points from the negative bags will be the target hypothesis. The figure from the paper provides a straightforward illustration of the algorithm schema.

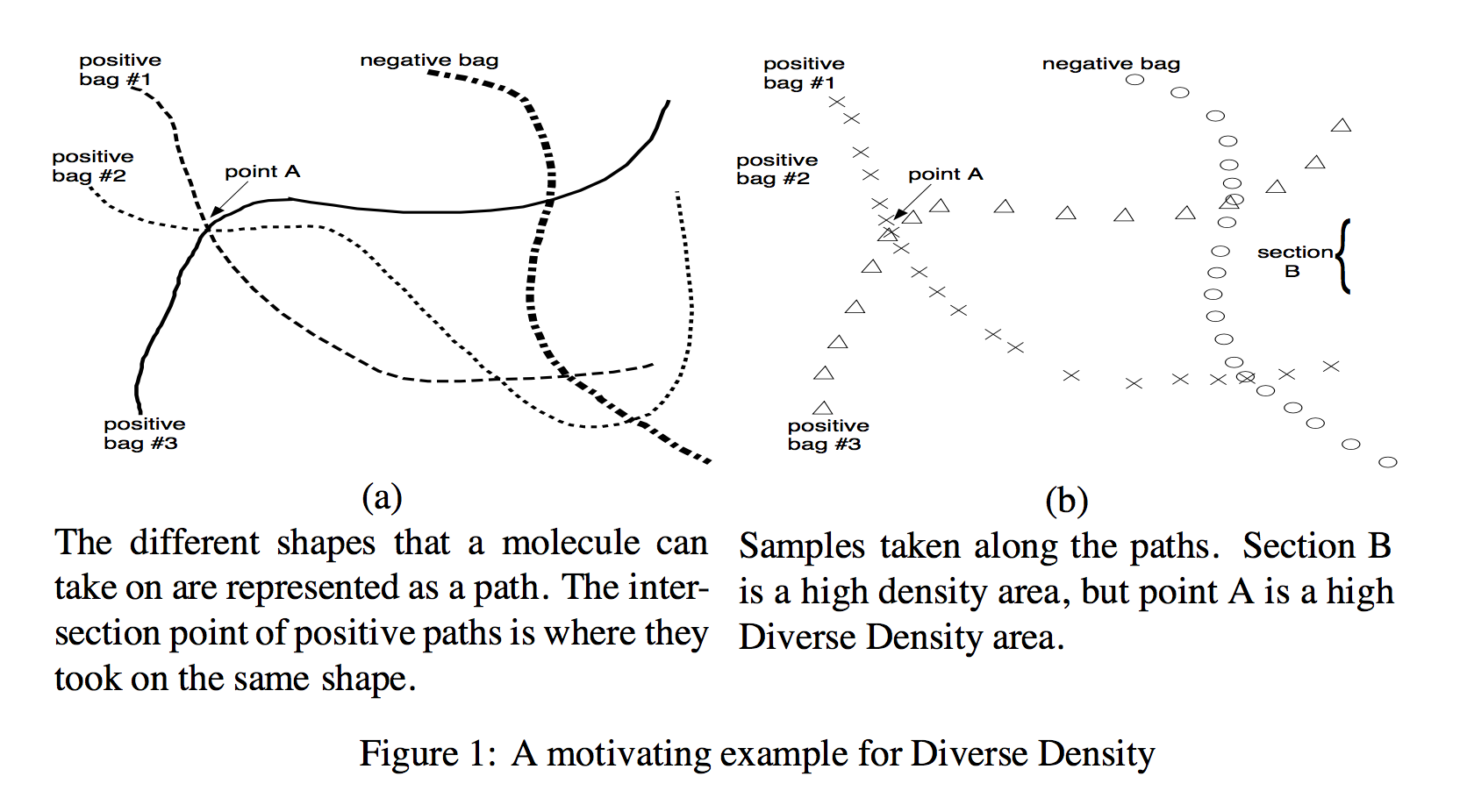
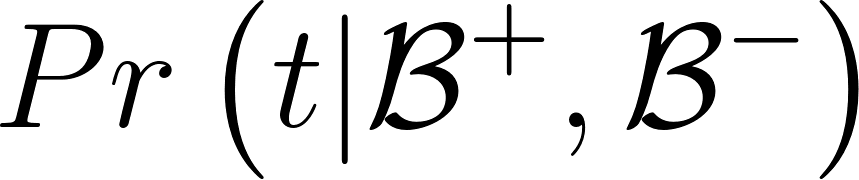


Figure 1: A motivating example of diverse density. The left side figure shows different shapes that a molecule can take are represented as a path and the intersection point of all positive bags is where they took on the same geometry. On the right, point A shows the high diverse density area.

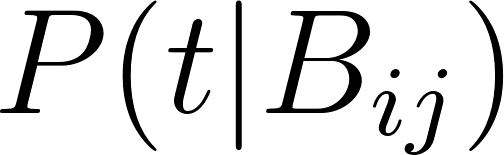
The graph used a two-dimensional space in which each dimension is correlated to one feature space to simplified the original problem. Each point on the graph would then be correlated to a hypothesis. A line in space corresponds to a bag with a list of instances. The diverse density function is looking to find the intersection point of all positive bags that is not part of any negative bags. This scenario has a special representation in the molecule identification example. In the problem, each molecule is an individual bag. Each bag has different geometries that are represented by different combinations of features. The test wants to allocate the target geometry that provides certain characteristics to the whole molecule. A molecule with such geometry will show a positive label, while molecules without such geometry will show a negative label. The number of geometry within positive bags does not matter. The problem wants to look for a geometry which is an instance that proves to be essential for the identification of the whole molecule which is a bag. After locating the target geometry, the algorithm can use it to classify other molecules through comparisons in molecule classification problems.

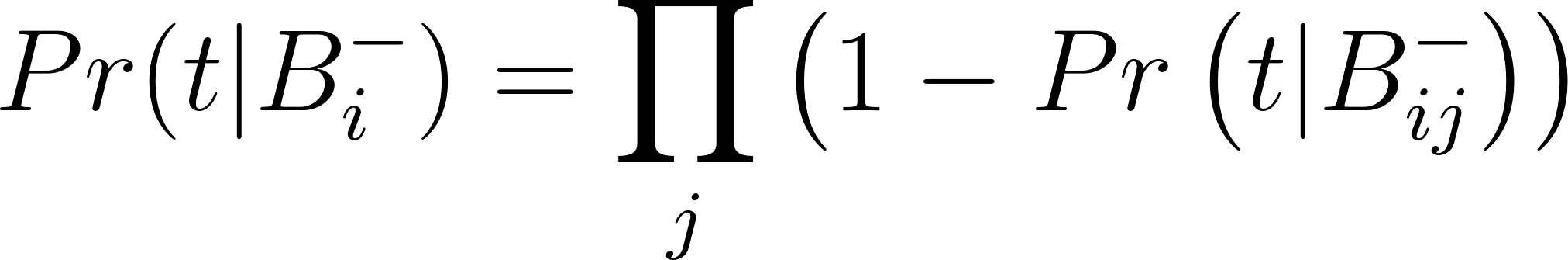
The implementation of diverse density algorithms including several iterations, for each iteration a random instance from a random positive bag is selected. The instance will serve as the hypothetical target instance, we then test the probability of the selected instance being the actual target instance. The goal is to maximize the probability of this selected instance being the target instance. To calculate the probability, a distance comparison was made between the selected instance and each instance in each bag. The distance is an average of the difference in each feature value within the instance. Then the probability will be calculated based on the distance. According to the Bayes’ rule, to maximize the probability of a bag with correct labels given a known target is to maximize the probability of the target given the probability of bag labels, which is

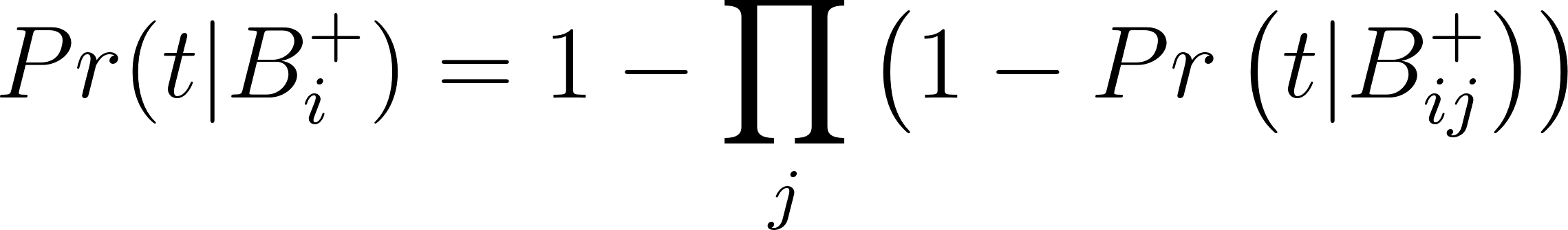
[](https://www.codecogs.com/eqnedit.php?latex=Pr%5Cleft(t%7C%7B%5Cmathcal%20%7BB%7D%7D%5E%7B%2B%7D%2C%7B%5Cmathcal%20%7BB%7D%7D%5E%7B-%7D%5Cright)%7D%0)

given t being the target instance and B being the bags. Therefore using the independence probability rules, the probability of the total bags is the product of the probability of each of the positive and negative bags, and the maximization function can be expressed as,

[](https://www.codecogs.com/eqnedit.php?latex=%7B%5Cdisplaystyle%20DD(t)%3DPr%5Cleft(t%7C%7B%5Cmathcal%20%7BB%7D%7D%5E%7B%2B%7D%2C%7B%5Cmathcal%20%7BB%7D%7D%5E%7B-%7D%5Cright)%3D%5Carg%20%5Cmax%20_%7Bt%7D%5Cprod%20_%7Bi%3D1%7D%5E%7Bm%7DPr%5Cleft(t%7CB_%7Bi%7D%5E%7B%2B%7D%5Cright)%5Cprod%20_%7Bi%3D1%7D%5E%7Bn%7DPr%5Cleft(t%7CB_%7Bi%7D%5E%7B-%7D%5Cright)%7D%0)

for each bag i. Following that, the probability of each bag can be a product of the probability of each instance j in each bag i, [](https://www.codecogs.com/eqnedit.php?latex=%7B%5Cdisplaystyle%20P(t%7CB_%7Bij%7D)%7D%0), and thus the positive bag and negative bag probability can be expressed as:

[](https://www.codecogs.com/eqnedit.php?latex=%7B%5Cdisplaystyle%20Pr(t%7CB_%7Bi%7D%5E%7B-%7D)%3D%5Cprod%20_%7Bj%7D%5Cleft(1-Pr%5Cleft(t%7CB_%7Bij%7D%5E%7B-%7D%5Cright)%5Cright)%7D%0)

[](https://www.codecogs.com/eqnedit.php?latex=%7B%5Cdisplaystyle%20Pr(t%7CB_%7Bi%7D%5E%7B%2B%7D)%3D1-%5Cprod%20_%7Bj%7D%5Cleft(1-Pr%5Cleft(t%7CB_%7Bij%7D%5E%7B%2B%7D%5Cright)%5Cright)%7D%0)

Combining all the above equation, we use exponential to calculate the entropy,

[](https://www.codecogs.com/eqnedit.php?latex=%7B%5Cdisplaystyle%20P(t%7CB_%7Bij%7D)%5Cpropto%20%5Cexp%20%5Cleft(-%5Csum%20_%7Bk%7Ds_%7Bk%7D%5E%7B2%7D%5Cleft(x_%7Bk%7D-(B_%7Bij%7D)_%7Bk%7D%5Cright)%5E%7B2%7D%5Cright)%7D%0)

According to the paper, Diverse Density has limited real-world application, and Molecule Classification is the probability of the most significant usage of diverse density. This is because of the special quality of that one geometry in the molecule bag can decide the test result of the molecule bag. And the absence of such a geometry instance will result in the test result of the bag being negative, which matches perfectly with the assumption in multiple instances problems.

The efficiency of diverse density heavily relies on the optimization function adopted in the algorithm. A fitting algorithm determines the target instance value and the scalar weight of each attribute to be considered in future iterations. A fitting optimization algorithm can both reduce the time and improve the accuracy of the result.

### Expectation-Maximization version of Diverse Density

Expectation-Maximization version of Diverse Density (EMDD) is a multiple instance learning algorithms that combines EM with the extended diverse density. The algorithm is an improvement in the time and efficiency of DD. The EMDD algorithm used a hidden variable that is calculated in the way of expected maximization. To elaborate on that, EMDD starts similarly to DD by picking random instances as a hypothesis. A series of repetitions was then performed on the selected date. The iteration process can be reflected from this pseudocode from the paper:

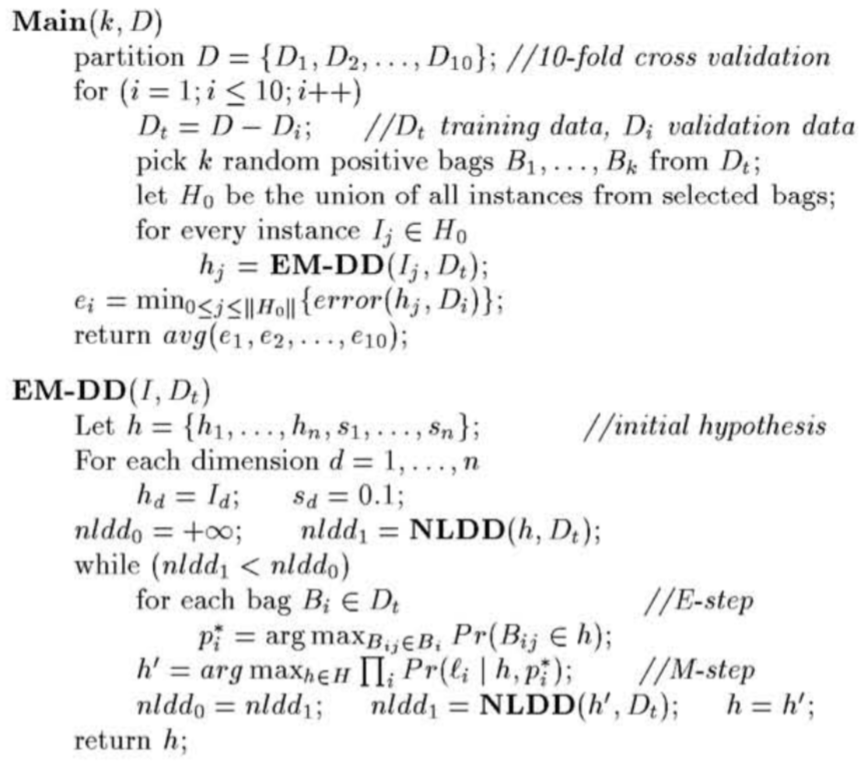


Figure 2: Pseudocode for EMDD where k indicates the number of different starting bags used.

The repetition includes using a generative model to calculate the instance with the highest probability in each bag based on its distance to the randomly selected hypothesis. After that, gradient descent was used to look for a new hypothesis that maximizes the DD instances, and this will be the representative instance for the whole bag. The algorithm eliminated the potential target concept through convergent after each iteration. The paper mentioned that the EMDD algorithm can be viewed as a special case of the K-means clustering algorithm, such that only one cluster is acknowledged. In this way, the algorithm removes the noise from the DD algorithm and turns the multiple instance problem as a single instance problem. The advantage of EMDD is its efficiency, because of the single instance characteristics, the algorithm is projected to run much faster than DD.

### Results

The following running results are for 10 iterations, with 5-fold cross-validation.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  | DD |  | EMDD |
|  | Musk 1 | Elephant | Fox | Musk 1 |
| Total bags | 92 | 200 | 200 | 92 |
| Actual Positive | 47 | 100 | 100 | 47 |
| Actual Negative | 45 | 100 | 100 | 45 |
| Predict positive | 61 | 116 | 143 | 64 |
| Predict negative | 31 | 84 | 57 | 28 |
| True positives | 45 | 92 | 95 | 38 |
| False positives | 16 | 24 | 48 | 26 |
| True negatives | 29 | 76 | 52 | 19 |
| False negatives | 2 | 8 | 5 | 9 |
| Accuracy | 0.8043 | 0.8400 | 0.7350 | 0.6196 |
| Precision | 0.7377 | 0.7931 | 0.6643 | 0.5938 |
| Recall | 0.9574 | 0.9200 | 0.9500 | 0.8085 |

Table1: Accuracy between Musk 1, Elephant, Fox dataset for DD and EMDD

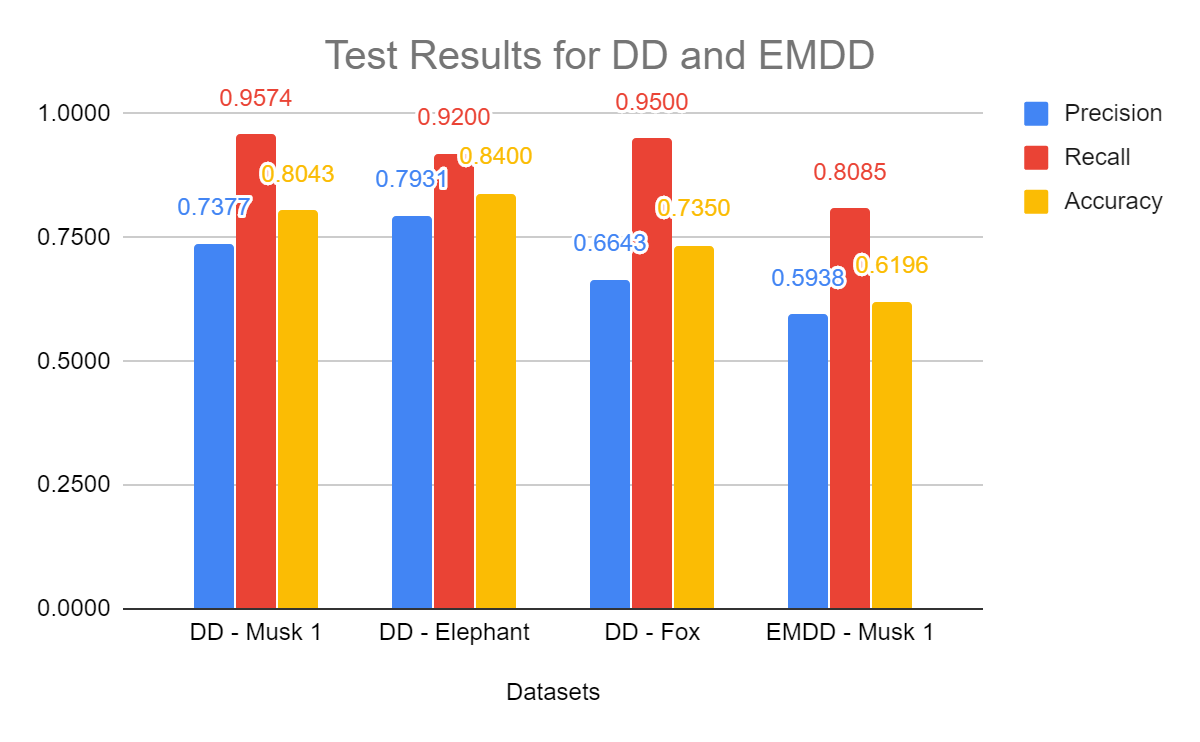


Figure 3: Comparison between different dataset for precision, recall and accuracy

### Discussion

In general, EMDD should have higher accuracy than DD. However, the EMDD accuracy in my test run is much lower than DD. This could be due to errors in coding. The elephant dataset has the best accuracy, following by Musk 1 and Fox. A test run was also performed on Musk 2, the dataset takes a long time to run. After the test, the code was also modified, therefore the result was not included. The DD is algorithm running on Musk1 has an accuracy of 80.43 percent which is lower than the 88 percent achieved from the paper. There could be various reasons for that. The optimization method is the most important factor in the efficiency of the algorithm. For this project, the optimization methods from Scipy were adopted to minimize the probability value of each bag within each iteration. Several optimization methods are tested including numerical values, minimize(method=’SLSQP’), minimize(method=’CG’), minimize(method=’L-BFGS-B’). The function of the gradient is absolutely essential in the usage of these optimization methods. The SLSQP minimizer without gradient takes hours to produce a result. The CG method, which is the conjugate gradient method works better than SLSQP. However, the produced scale for features is abnormally large therefore the algorithm is not adopted. The SLSQP with gradient descent using sigmoid, which was devised for logistic regression, is much faster, however, the accuracy was only 54 percent. The low accuracy could be a result of both the optimization algorithm and errors in the implementations. Since then, the algorithm has been improved and L-BFGS-B was adopted. L-BFGS-B which was mentioned in the R. Byrd paper [3] has a much visible improvement in the runtime of the algorithm during testing. However, it still takes more than 10 minutes to run the Musk1 dataset for an iteration of 10, 2-fold cross-validation attempts. And the elephant dataset takes even much longer to run. On top of that, the low number of iterations and cross-validation could be another reason that is contributing to the lower results of accuracy. Because the algorithm takes the highest accuracy from the overall runs. The accuracy could potentially be improved through increasing the number of iteration and folds of cross-validation.

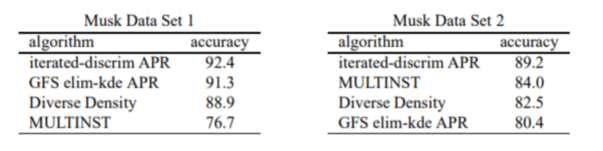


Figure 4: Comparison between different instance spaced algorithm in Maron [1]

In Maron’s paper, the author compared DD with other multiple instances of learning algorithms. From the table, we can observe that diverse density does not have a significant advantage in measuring the accuracy of either musk 1 or musk 2 dataset.

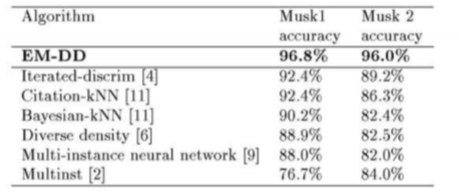


Figure 5. Result of EMDD run on Musk 1 and Musk 2 in Zhang [2]

In Zhang’s paper [2], the project reached a high 96 percent accuracy which is much higher than the 67 percent from my implementation. This shows that there might be errors in the implementation. In general, EMDD has a much better performance than other algorithms listed in Figure 3. However, the result was not quite the same in different testing. In J. Amores’s paper [4], a comparison between different instance spaced, bag space and embed spaced algorithms are compared altogether. And the chart shows that the accuracy of EMDD on running the same Musk 1 dataset is only in mid 80 percent which is not as high as a 96 percent indicated in the paper. This could be the result of different implementation methods and the optimization algorithm that is adopted. In conclusion, both DD and EMDD has a medium performance comparing with other instance spaced algorithms, and lower performance compared with SVM based algorithms, or bag instance and embedded instance algorithms.

There are many ways to improve DD and EMDD, the most important one is in searching for the initial hypothesis to be tested. Currently, a random hypothesis is selected as to be compared with each instance. In looking for a way that select random hypothesis in each iteration more strategically can improve both the accuracy and the efficiency of running. In addition to the improvement of the hypothesis, the weight methods of data can also be improved. Currently, in order to calculate the probability of each instance before calculating the probability of each bag, we used the average of the distance before sending them to the exponential function for calculating probability. For the EMDD paper, the author also mentioned that several different weighting methods could be used during this process such as searching for the maximum distance or the minimum distance instead of averaging all the scalar distance after the summation of all feature distance. One last thing that can be improved in future implementation is the standardized method for data. Currently, the data is standardized through normalization in order to achieve significant value for exponential calculation. The normalization process takes the maximum value and minimum value of each attribute as the boundary. More advanced way of standardization could make the difference between data more significant while minimize the effect from a large negative data being fed to exponential or data that is close to 0 which might cause 0 division error in some calculations of probability.

### References

[1] Maron, Oded, and Tomás Lozano-Pérez. "A framework for multiple-instance learning." In Advances in neural information processing systems, pp. 570-576. 1998.

[2] Zhang, Qi, and Sally A. Goldman. "EM-DD: An improved multiple-instance learning technique." In Advances in neural information processing systems, pp. 1073-1080. 2002.

[3] R. Byrd, L. Peihuang, and J. Nocedal, “A limited-memory algorithm for bound-constrained optimization,” Jan. 1996.

[4] J. Amores, “Multiple instance classification: Review, taxonomy and comparative study,” *Artificial Intelligence*, vol. 201, 2013, pp. 81–105.

## Embedded Space Algorithms

## Embedded Space Distance Functions- Eric Miller

In this section of the paper, I investigated the Embedded Space (ES) Paradigm of Multiple Instance Learning. ES algorithms seek to extract global information about the bag with a mapping function, *M(X),* which takes a bag, *X*, and returns a feature vector *v*, which attempts to summarize the entire bag’s characteristics. Once all bags are mapped to single vector, a standard classifier is used. Amores et al. [1] discusses two different sub paradigms, one that uses statistics about the bags (such as mean or min/max) to create the final vector (non-vocabulary based methods), and vocabulary-based methods, which attempts to create ‘classes’ of instances, typically with some form of clustering, and then uses this vocabulary to describe the content of a bag relative to the other bags.

### **Non-Vocabulary Based Methods**

The methods included in the ES paradigm that do not use vocabulary attempt to create a vector that represents all of the instances in the bag without considering differences between the instances. Thus, these methods tend to map a bag to a vector that represents some statistics about the instances in the bag. Dong et al.[2] proposes an approach which maps each bag to a vector holding the average of each instance, , where is a vector representing an instance in *X*. This method is known as Simple MI. Another method in this sub-paradigm is to create a min-max vector, using with and [3].

*Implementation and Experiments*

I implemented both of the aforementioned mapping functions, with the addition of one experimental approach. For this experiment, I combined the two methods above, so that the mapping function is as follows:

The resulting vector holds the min, max and average of the instances in *X.* All 3 of the mapping functions are used within the same algorithm, which maps each bag to a vector using *M(X)* and uses a standard classifier to classify the vectors. I used an implementation of a Support Vector Machine (SVM) from *Sci-Kit Learn* as the standard classifier, and used a Radial Basis Function (RBF) kernel also from *Sci-Kit Learn*. For all 3 methods, the C parameter was set to 10, though typically any C value above 1 resulted in a fairly significant increase in accuracy(3%).

#### Experiments

The first experiment I ran compared the accuracies of each of the 3 mapping functions on all 3 datasets. This result is shown to the right in figure (??). I expected the combined mapping function to be the most accurate, as it would provide additional information to the classifier about the distribution of the data. I also expected this approach to be more robust, as *musk1* is drug activity prediction, which generally requires the presence of a single configuration (instance) that binds well, whereas the image classification files require several regions or instances to exist. I thought the combined approach would provide the classifier with the information needed to perform well in both of these scenarios, though it seems this approach may have made the vectors too noisy.

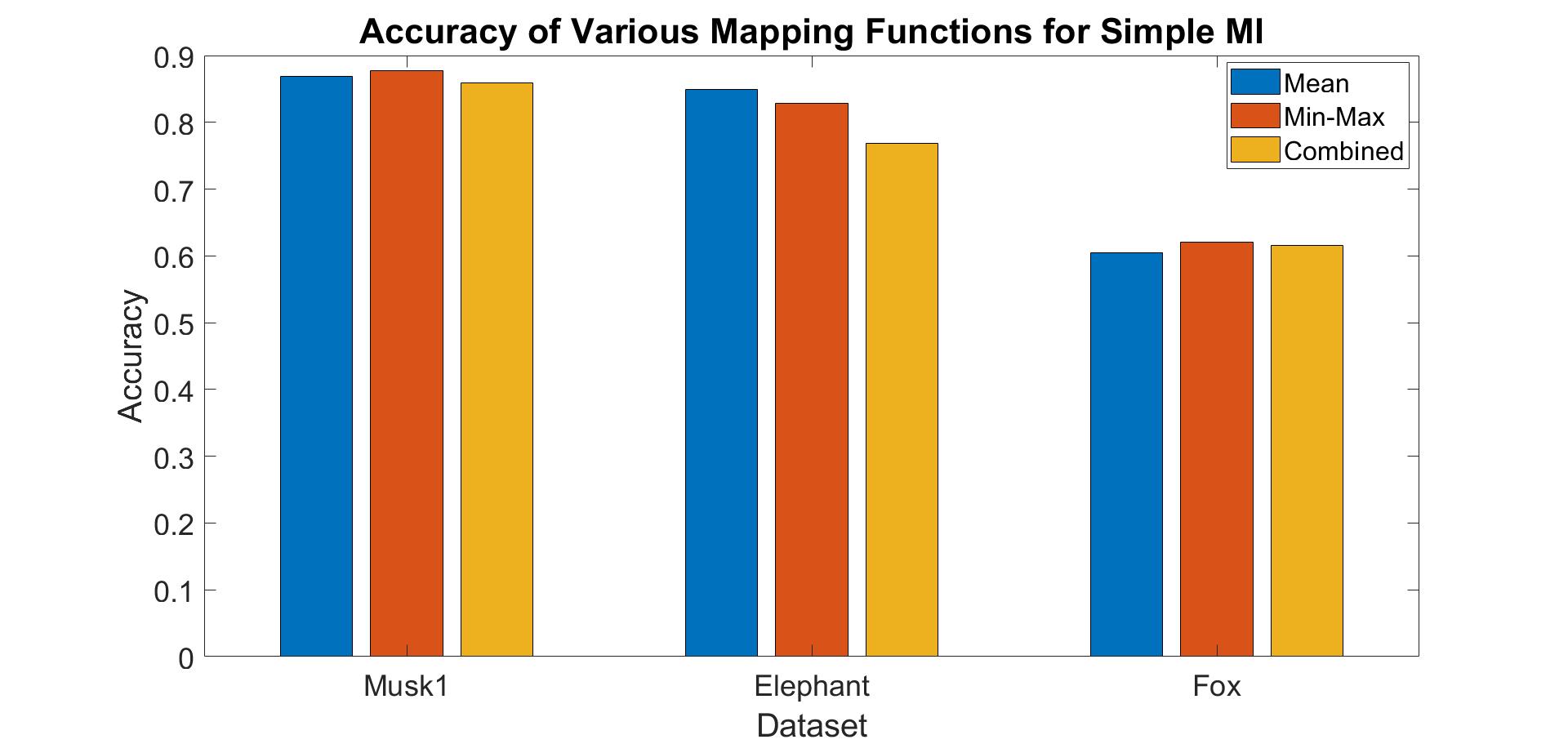


Figure 1: Simple MI Maping Functions

The other parameter I experimented with (aside from the C parameter tuned for each data set) was the kernel function used in the SVM for this approach. The linear kernel and RBF kernel performed similarly on all three datasets, and both outperformed the polynomial kernel. Since the vectors are made from aggregating statistics about the instances in the bag, a simple decision boundary is likely sufficient, since most of the low level information within the bag is lost during mapping.

### **Vocabulary Based Methods**

Most of the methods in this paradigm use what Amores et al. [1] refers to as a ‘vocabulary’, which attempts to classify the instances within the bags. To create the vocabulary, all of the instances within all of the training bags are clustered, typically using K-means. In vocabulary-based methods, the mapping function, *M(X,V)*, maps *X* to a vector *v* using vocabulary *V*. The resulting vector, *v*, will represent how well the instances in *X* match the classes of instances in *V*. All of the Vocabulary-based methods have the following components, as highlighted in Amores et al. [1] :

1. Vocabulary, V which stores classes of instances, each of which is typically described by a prototype vector, *P*, holding the averages of the instances contained in the class (i.e the center of the cluster).
2. Mapping function, *M(X,V) → v,* which creates *v* by considering how well the instances in X match each class in V
3. Standard Classifier, which classifies the vectors resulting from *M(X,V).*

There are many approaches to MIL in this sub-paradigm, differing in how the vocabulary and mapping functions are defined. In this analysis, I mainly investigated the different mapping functions that can be used, though some considerations of the vocabulary are discussed. Amores et al. [1] identifies 3 families of Vocabulary Based Methods: *Histogram-based*, *Distance-Based* and *Attribute-Based.* I implemented at least one algorithm in each family, and will discuss each general approach below. Amores et al. suggests experimenting with 6 different vocabulary sizes, starting with half the number of instances and successively dividing in half to get the rest, and also suggests concatenating various sizes of vocabulary; both of these approaches are discussed in the experiments section [1].

#### **Histogram-Based Approaches**

##### Hard Assignment

This class of ES algorithms use a mapping function *M(X,V)*, that maps a bag X to a vector, where is the number of instances in X that are part of the i-th class of the vocabulary. The final vector, *v*, is called a ‘histogram’ because each instance is typically in a single class, which makes the resulting vector resemble a histogram, with higher values suggesting that instances in X match that cluster well [1]. Due to its frequent use in computer vision, this approach is often called “Histogram Bag of Words” (HBOW) [1]. Most histogram-based methods cluster the instances, typically though K-means, though I tested one implementation without any clustering. Inorder to generate the histogram, Gemert et al. describes to the mapping function for hard assignment HBOW is defined as *M(X,V) → ,* where:

, *(1,2) [4]*

Note that Z is used to normalize *v*, so it is set to |*X*|. represents the center of cluster *i,* and *K* is the number of clusters in the vocabulary*.* The equation above uses hard-assignment, as each instance belongs to a single cluster. HBOW with hard assignment typically uses Euclidean Distance [4], though I also experimented with a similarity function suggested for Distanced based Bag of Words (DBOW) in [1]. The similarity function can be represented as:

*(3)*

When using the similarity function, is changed to maximize similarity as follows:

*(4)*

##### Soft Assignment

Gemert et al. propose an HBOW approach with soft assignment, which changes the mapping function so that the histogram can be more expressive of the data [4]. Rather than the mapping function in Eq. 1,2, [4] proposes 3 different mapping functions, based on kernel density estimation, which attempt to estimate the probability density function with the following kernel:

*(5)*

The first novel mapping function mentioned in [4] is referred to as the *kernel codebook*:

*(6)*

The *Kernel codebook* is similar to the similarity function used in the YARDS algorithm, discussed in the next section. *Kernel codebook* uses soft assignment to model the probability density function in a continuous manner rather than the discrete representation of the HBOW mapping function. This mapping function allows instances to belong to all clusters, with the ‘membership’ proportional to However, without normalizing, the *kernel codebook* can easily become skewed, as instances may contribute vastly different amounts of probability mass to the histogram. To solve this, Germert et al. proposes *codeword uncertainty*, which maps to *v* as follows:

*(7)*

This mapping function attempts to capture the degree that instances belong to all clusters, similarly to the *kernel codebook* [4].However, this approach ensures that each instance contributed equally to the histogram, and does this by normalizing the kernel function such that an instance that is close to multiple cluster centers does not skew the data. Each instance contributes a probability mass of 1, but can make fractional contributions to multiple classes in the histogram. This is much different than the mapping function in Eq. (2), above, as it allows membership in multiple classes by avoiding a hard 1 or 0 assignment. However, both approaches are also similar in the way that they allow each instance to contribute a probability mass of 1 to the histogram, making no more instance in the bag more important than the other [4].

Since the instances all have the same weight in the histogram, the resulting histogram is unable to represent which instances are most relevant. A consequence of this is that an instance that is very far from all clusters would have a similar impact on the histogram as an instance that is close to all clusters, since they would both contribute the same amount, despite one being irrelevant given the vocabulary. To account for this problem, Gemert et al. propose *codeword plausibility*, which maps to *v* a follows:

, *(8,9) [4]*

This approach attempts to capture the relevance of an instance, as it will assign probability mass based on the kernel function, so more relevant instances will have a larger impact on the histogram. This approach uses hard assignment, so that an instance can only belong to the cluster it is closest to. The three approaches outlined in [4] make different assumptions about the data and can be effective in the right scenario, such as object classification, which is the application [4] developed this approach for.

##### YARDS

Another histogram based approach that I implemented was ‘Yet Another Radial-based Similarity Measure’ (YARDS) [5]. I chose to implement this approach because Foulds et al. chose not to cluster the data, and use the raw instances as prototypes, which makes YARDS the only algorithm I implemented that does not cluster [5]. Amores et al. [1] places this algorithm into the HBOW family because it uses the same mapping function described above in Eq. (1). This approach is very close to the *kernel codeboo*k in Eq. (5), as the YARDS approach changes ) in the mapping function, Eq.(1, 2), to be:

, *(1,10) [5]*

##### Implementation

My implementation of all HBOW algorithms use *Sci-Kit Learn’s* SVM implementation, with a Chi-Squared Kernel function. This kernel is commonly used in many computer vision problems on histograms of words, and is discussed in Amores et al. [1]. Many different vocabulary sizes were tested, though my results favored large vocabulary sizes. The C parameter was chosen to be 10 (for Musk1), though it did not have a large effect on the accuracy. The sigma values for the kernel functions and similarities were set to one, though this was tested in an experiment. As for the distance function of hard-assignment HBOW, Euclidean distance slightly outperformed similarity. The experiments were run with 3 iterations of 10 fold cross-validation due to the amount of parameters considered.

##### Experiments

I ran several experiments on my implementations of HBOW. I first investigated the size of the vocabulary. Amores et al.[1] recommends using 6 different vocabulary sizes, starting with a power of 2 close to half of the total number of instances and successively dividing by 2 to get 6 vocabulary sizes. It is also recommended to make 10 vocabularies for each of the 6 sizes and concatenate the resulting vectors to get the final vector *v*. However, due to the computational expense such an approach would incur, I experimented with each of the 6 proposed vocabulary sizes on *Musk1* separately for all of the previously mentioned mapping functions.

As shown in Figure 2, accuracy generally increased with number of clusters, with an exception being *plausibility*. I originally expected accuracy to peak at a middle value, especially for the hard-assignment mapping functions. In hard-assign HBOW approaches, each instance can only belong to one cluster, which means that for a bag with *n* instances on a data set with K clusters, at most *n* of the *K* values in *v* will be non-zero. For the data sets we used, the number of instances in a bag were small relative to the number of instances in the training sets, which means that most of the values, , in the histogram are zero. However, as the number of clusters increases, the number of instances in each cluster decreases, and thus the clusters better represent the instances that belong to it.

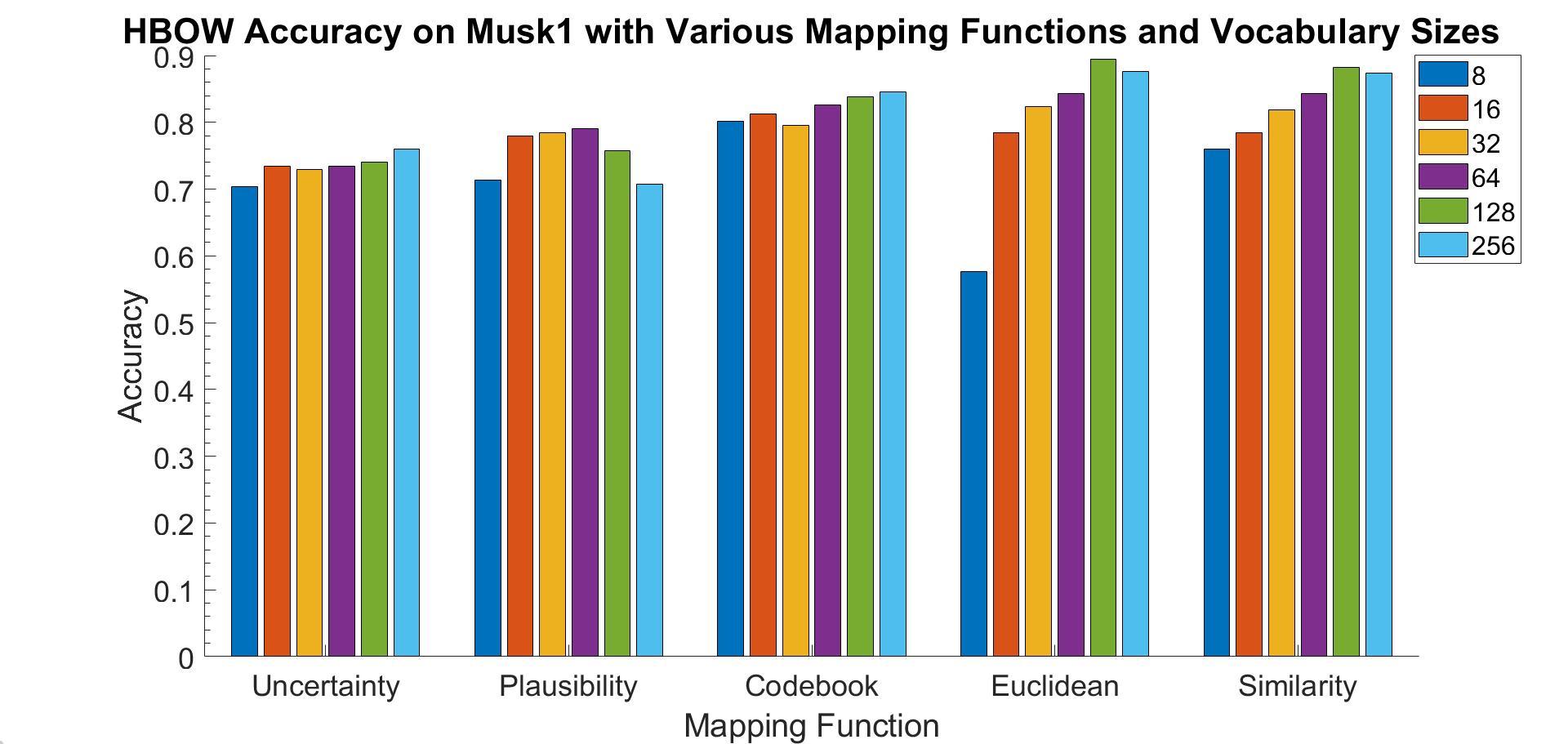


Figure 2: HBOW Vocabulary sizes

As the number of clusters decreases, it becomes more likely that both positive and negative instances (i.e the bag they are in) belong to a single cluster. Ideally, there will still be at least one cluster that contain mostly instances that are needed for a positive classification (i.e the ‘good drug’ instances in *Musk1),* though this becomes less likely as the number of clusters decreases. I also considered the possibility of overfitting with large numbers of clusters, as only about 2 instances belong to each cluster for the largest vocabulary size. However, this experiment was run using cross-validation, so over-fitting problems would have been reflected in the resulting accuracies. Also note that the hard-assignment approaches had higher accuracy on *musk1,* which makes sense as the soft-assignment mapping functions were developed for image recognition.

Another experiment I ran was with the *YARDS* algorithm. As mentioned above, [5] does not cluster the data and uses the raw instances as the ‘cluster centers’. I experimented with using only positive instances as the cluster centers, with the resulting accuracies for each dataset shown below in Figure 3. The *Musk1* problem typically requires a single instance (i.e one configuration that binds well) to classify as positive, and MIL image classification has a similar requirement, though the number of ‘regions’ required is larger than one. Thus, it makes sense that there was only a small performance effect for *Musk1*, which may have occurred because valuable information is lost when removing the negative instances, even if the SMI assumption is valid. *Elephant* and *Fox* see a small benefit and no benefit, respectively, which could be due to more regions being required for a positive classification.

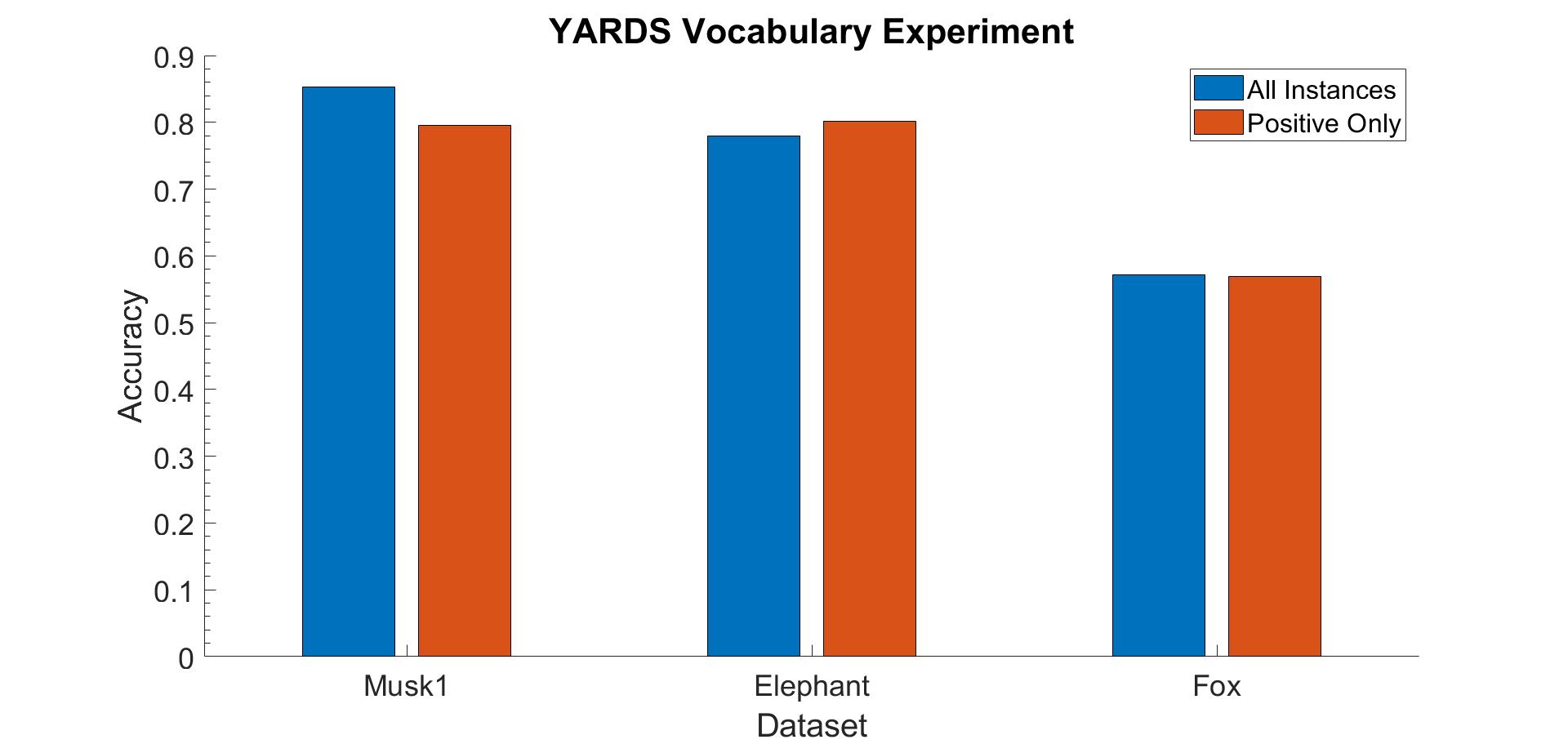


Figure 3: Yards Vocabulary Experiment

The final experiment I ran was comparing the final accuracy on the 3 data sets with various distance functions. I expected the soft-assignment to perform better on the *Elephant* and *Fox* datasets, as [4] originally proposed those mapping functions for the image classification problem. The results of this experiment are shown below in Figure 4, with the mapping functions sorted by performance on the *Elephant* and *Fox* datasets. The *plausibility* metric performed the best on the image data, though the improvement in accuracy was only about 2.5 percent. On the *Musk1* dataset, the Euclidean and Similarity outperform the measures proposed in [4], which is likely due to the fact that a single configuration is needed for positive classification, so a hard-assignment is sufficient. In the image files, multiple regions must exist for positive classification, so a soft-assignment approach would seem better suited, as instances can belong to multiple regions with soft-assignment. Additionally, Figure 4 shows that YARDS does not provide any performance benefits on the image datasets compared to Euclidean and similarity, and underperforms these metrics on *Musk1*.

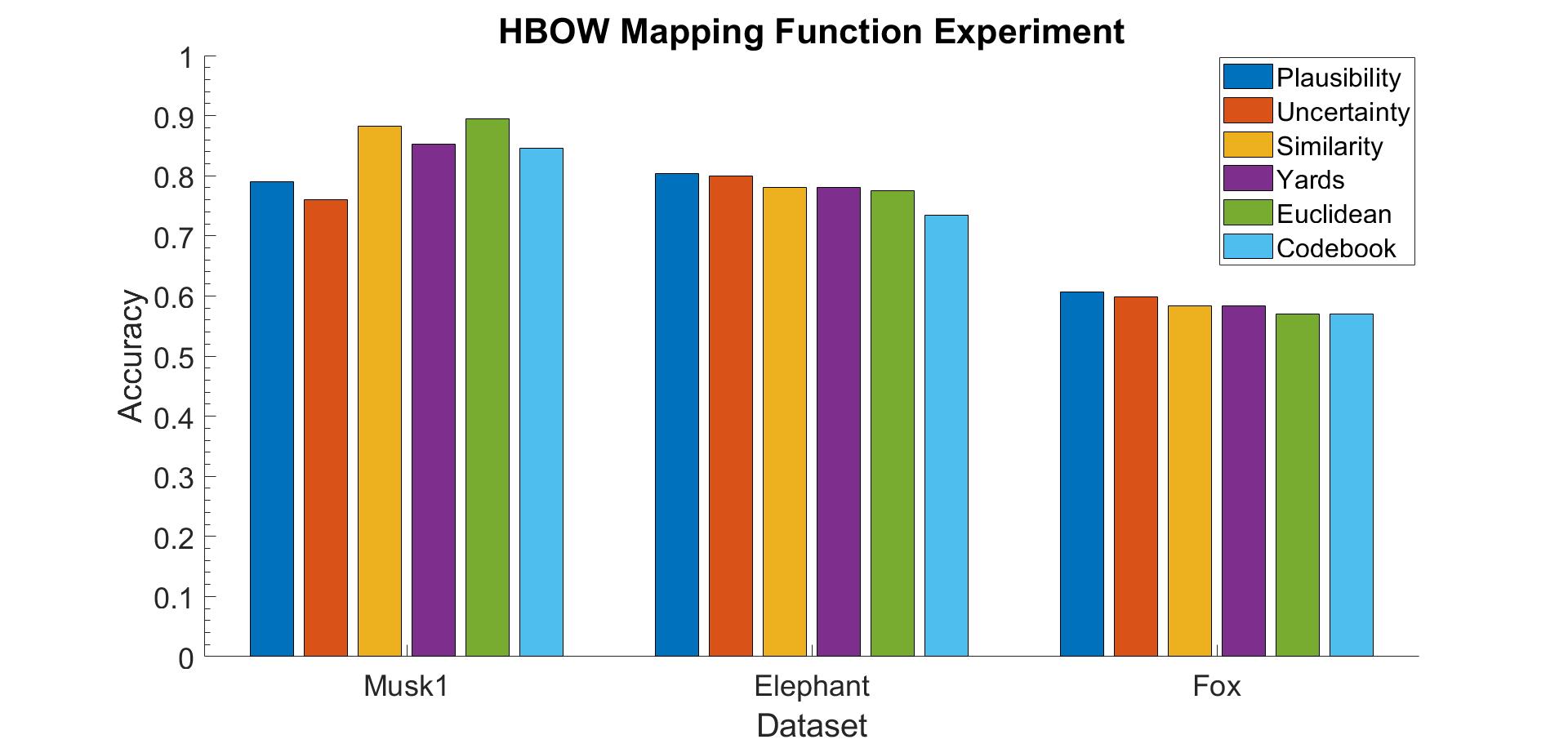
****

Figure 4: Histogram Bag of Words (HBOW) Mapping Function

#### **Distance-Based Approaches**

This class of ES algorithms use a mapping function that maps a bag X to a vector, where represents the distance or similarity to the i-th class of the vocabulary. There are two similar approaches suggested in Amores et al. [1], which mentions many papers implementing similar algorithms in this sub-paradigm. Due to its frequent use in computer vision applications, this approach is typically called Distance Based Bag of Words (DBOW). One of the approaches uses a mapping function *M(X,V) → v* where:

*(11)*

In this approach, *d* is the distance from an instance, *x*, to the j-th cluster in vocab *V*. The vector *v* becomes populated with the closest distance of instances in *X* to each cluster in *V.* The papers discussed in [1] use different distance metrics including the Euclidean distance, Mahalanobis distance, weighted Euclidean Distance. Some papers mentioned in [1] also use a similarity based approach, which uses a mapping function *M(X,V) →* v where:

(*12)*

Where *s* is the similarity equation from Eq. (3).

Many of the papers mentioned in [1] implement DBOW in slightly different ways. For example, Auer et al. [6] uses DBOW with Adaboost, where each weak learner is a single “ball”, which is represented by the center of a positive bag *x* and a radius *r*. A bag is classified as positive if it’s center is within the chosen radius *r* of the ball with center *x*. Every iteration of Adaboost iterates through all positive bags, and for each bag finds the optimal radius which maximizes the *distribution accuracy,* or the weights of the correctly classified training bags. The ball with highest distribution is chosen by Adaboost for the weak learner [6].

Additionally, many of the papers mentioned in [1] paper do not cluster the data, but rather use the positive instances of the training data as cluster centers. I ran many experiments with various clustering approaches, which are discussed in the experiments section. I also experimented with Adaboost and SVM as standard classifiers, though my implementation did not embed the mapping function into Adaboost as [6] did.

##### Implementation

For my implementation, I decided to implement the approach outlined in [1]. This approach mapped a bag, *X*, to a vector prior to using the standard classifier. My implementation of all DBOW algorithms uses *Sci-Kit Learn* implementations ofSVM and Adaboost. Once the C parameter was tuned, SVM outperformed Adaboost, as this tuning raised the cross validation accuracy by about 3 percent. For SVM, I ran experiments with both the RBF and Polynomial Kernels, but ultimately decided on RBF due to its superior performance. I also experimented with Euclidean distance and similarity, but ultimately picked similarity due to its superior performance, which is consistent with [1].

##### Experiments

Amores et al. [1] recommends using 6 different vocabulary sizes, starting with a power of 2 close to half of the total number of instances and successively dividing by 2 to get 6 sizes. For *Musk1* I tested vocabulary with sizes: [8,16, 32, 64, 128,256], and saw that accuracy generally increased with K, and found optimal values of K=128 for similarity and K=256 for Euclidean distance on *Musk1* (*Fox and Elephant* yielded similar results).

I additionally experimented with mapping a bag *X*, to vector, where is a vector resulting from for a set of vocabularies with different sizes. Amores et al. [1] recommends making 10 vocabularies of each 6 sizes, mapping *X* to for each vocabulary and concatenating all resulting *v*. Though the approach using 10 different vocabularies for each size was too computationally to test extensively, I one experiment I create a vector for each size and concatenated them for the final vector. I repeated this procedure for experiment but created 3 vectors from each vocabulary size. These experiments are labeled with “K=all” and “K=all\*3” in Figure 5. However, neither approach outperformed a single vocabulary size of 256 (for *Musk1*) and took much longer to run. Overall, SVM with K=128 was selected due to its relatively low computational complexity (compared with higher amounts of clusters) and its high accuracy.

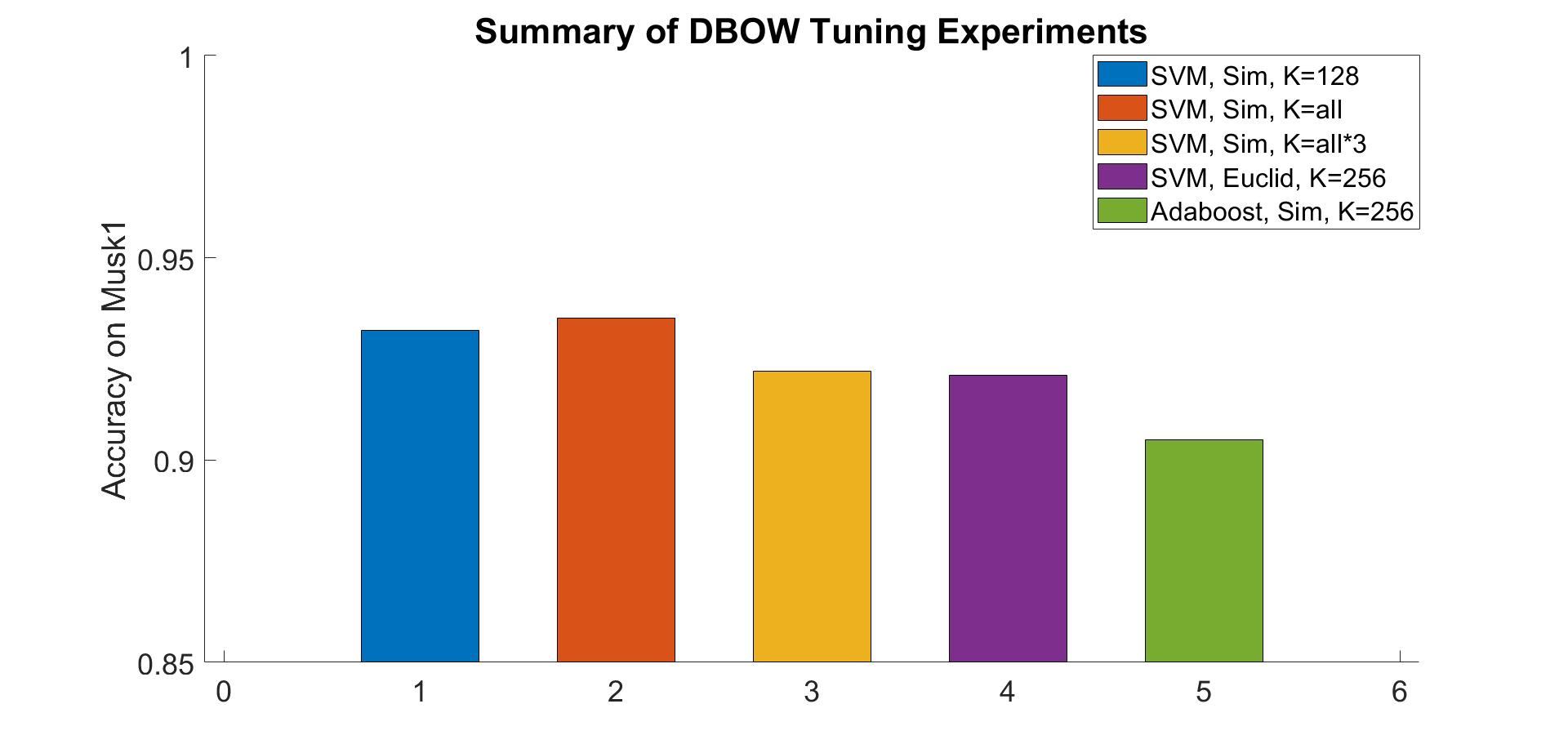


Figure 5: Summary of DBOW Experiments

#### **Attribute-Based**

Amores et al. [1] describes the Intermediate Matching Kernel (IMK) [7] algorithm as the only “Attribute-Based” approach. Rather than mapping a bag *X* to a vector *v*, the IMK approach maps a bag to concatenated subvectors. The IMK approach clusters the instances prior to mapping. Boughorbel et al. describes their mapping function, which finds the nearest neighbor in *X* for every cluster, such that is the instance closest to the j-th cluster (also referred to as prototype), [7]. I use the same *M(X,V) → v*, where:

*(13)*

Note that the Amores et al. [1] points out that the hard assign IMK approach could modified such that each sub-vector, is a weighted average of instances, where the weights are determined by how well an instance fits Boughorbel et al. uses fuzzy c-means clustering as its clustering algorithm, though this paper uses the *argmin* (hard-assignment) rather than the weighted approach proposed by Amores et al. [4]

The second part of the IMK approach is defining a kernel, which is where the Intermediate Matching Kernel approach gets its name. The IMK can be expressed as follows:

*(14)*

For each cluster, , the IMK finds the instances in X and Y that are the closest to , and finds their similarity using the above equation. This process repeats for each cluster, and the resulting similarity measure is summed. One important difference between the way the Amores et al. [1] describes this algorithm and the way that [7] describes this algorithm is that [1] uses the mapping function, *M(X,V),* prior to running the standard classifier, whereas [7] defines their kernel with the mapping inside, such that the kernel receives two bags, *X,Y* and the vocabulary, V, and finds the nearest neighbor of all clusters in the vocabulary for both bags. When implementing the IMK, I used the Amores et al. [1] approach, as it was computationally less expensive, as it only calculates the nearest neighbors once.

##### Implementation

When implementing the IMK approach, I used the *Sci-Kit Learn* implementation ofSVM. I implemented the mapping function and implemented the IMK in a way that was compatible with the implementation of SVM. The implementation of the IMK returns a matrix where is the number of training bags. This operation is done for every combination of bags, which makes this operation computationally expensive. [7] states that the runtime of the IMK as they wrote it is where p is the number of classes, and m and n are the sizes of the bags X, Y. However, this operation must be done for every combination of X,Y, so the overall runtime of calculating the IMK matrix is where B is the number of the bags. Due to the high runtime, the number of experiments run with IMK was limited. I was able to tune the C parameter of the SVM which ended up being set to 10. I also ran a couple simulations to tune the sigma used in the IMK, which ended up being set to .5.

##### Experiments

The key feature of the IMK is the custom kernel function specific for this problem. In order to test the effectiveness of this kernel, I used ran my implementation of IMK, but instead of using the IMK, I used an RBF kernel and a polynomial kernel. The results of this experiment are shown below in Figure 6. For this experiment, I used *Sci Kit Learn’s* implementation of the aforementioned kernels. This experiment kept all of the default parameters of the SVM, which means that I did not tune C for this experiment (default is 1). Due to the long runtime, only the *Musk1* data set was used with 10 iterations of 10 fold cross validation. The IMK kernel is likely superior for this approach, specifically when the vector contains concatenated instances, as IMK is similar to the RBF kernel, except that the IMK finds the similarity for each subvector separately. The IMK with inputs *X,Y* can be thought of as the sum of the RBF for each subvector in *X,Y*, and thus performs better than the RBF and polynomial kernels, which do not consider subvectors separately. Figure 6 also highlights that IMK accuracy was consistent across multiple vocabulary sizes.

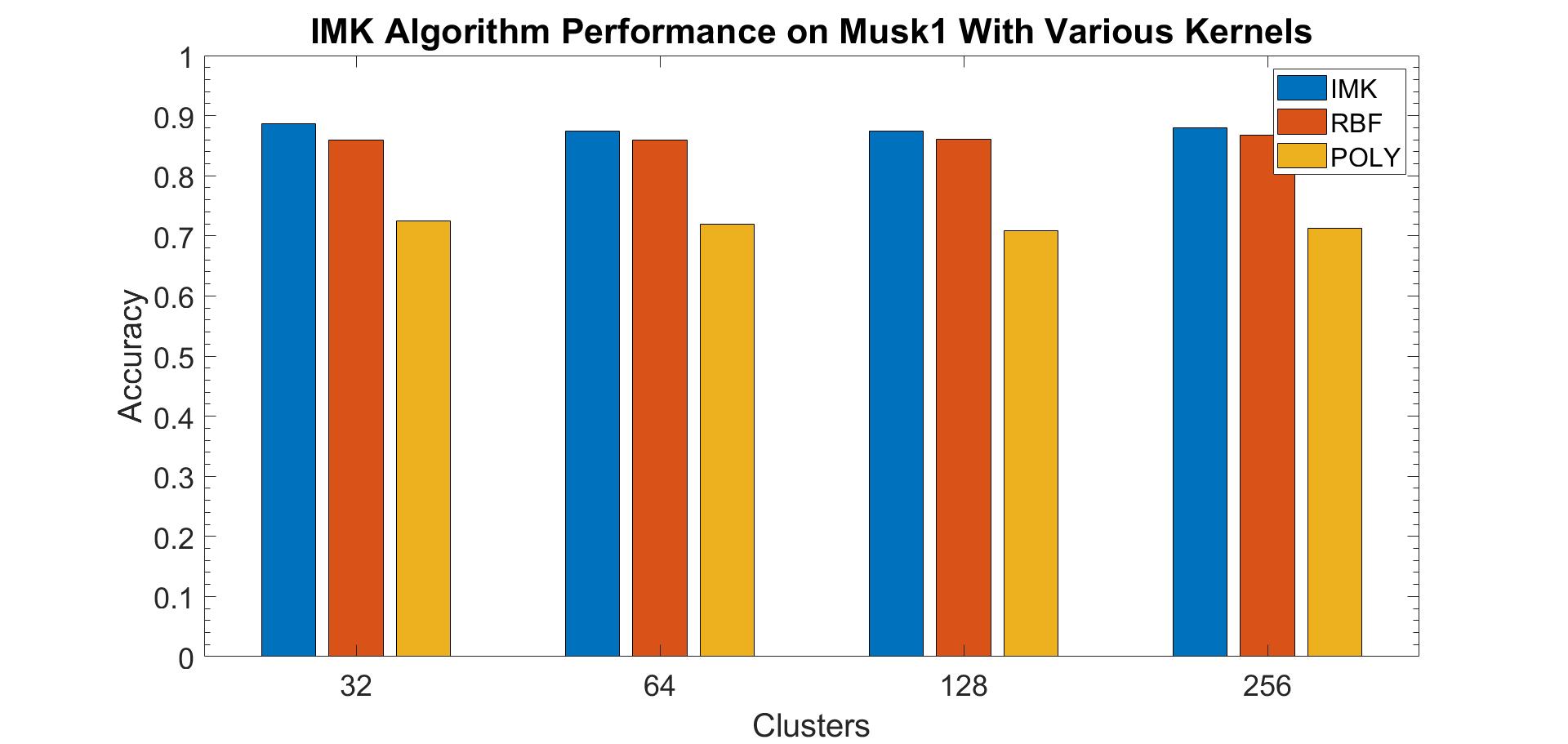


Figure 6: Effects of Kernels in IMK

### **Methodology**

To implement these algorithms, I implemented all of the mapping functions, distance functions and comparison metrics in Python. Implementations of SVM and Ababoost from *Sci-Kit Learn* were used for the standard classifier, though SVM outperformed Adaboost and was used on the final experiments. Hyper-parameter tuning (such as C for SVM) was done using 5 fold cross validation and Python’s *Multiprocessing* library (to speed up experimenting), though the amount of iterations ranged from 2 to 5 depending on the algorithm and dataset size. For methods that require clusters, I used an implementation of K-Means clustering from *Sci-Kit Learn*. Discussions on parameter tuning are in the *implementation* section of each paradigm.

All final results were found using 10 fold cross-validation for consistency with [1], though the number of iterations varied depending on the algorithm and dataset due to high run times for some of the algorithms implemented. I used the implementation of cross-validation my team had made for the 3 programming assignments. Unless otherwise noted, all data was standardized prior to training and testing using my own standardization implementation.

I tested the algorithms on 3 data sets: *Musk1, Elephant and Fox,* discussed in the introduction. I used the vocabulary sizing approach recommended in [1], where the maximum number of clusters tested for each data set was as follows: *Musk1:* 256; *Fox:* 512, *Elephant: 512.*

#### Results and Discussion

After all of the algorithms were tuned, and experiments were run to optimize each algorithm, I took the best performing algorithm with reasonable output for comparison. Figure 7, below shows the final results of all ES algorithms. As shown below, there is a large drop in accuracy with *Elephant and Fox,* which is likely because the image classification problems require multiple regions to exist in positive bags, thus requiring a more complex decision boundary. HBOW does not perform the best because it is sparse, which means most of the values in the mapped vectors are 0, giving less information to the classifier than approaches like DBOW. Even on the image data sets, where HBOW uses soft-assignment mapping functions optimized for image files, it is still outperformed by Simple MI and IMK. SMI may perform well on *Elephant* because it takes the average of all instances. If many regions are required to classify as positive, the average value may be sufficient. Additionally, Simple MI’s simplicity may help protect it from overfitting to noise in the training sets.

YARDS performs poorly relative to all of the methods, which is likely due to its use of raw instances as cluster centers. Approaches using a similar similarity function are used in DBOW and HBOW, though these methods implement it with clusters and perform superiorly to YARDS. IMK performs quite well on all 3 data sets, which is likely due to its combination of its use of clustering, and its ability to compare instances with these clusters. The IMK uses its unique kernel to express that two instances that are equidistant to the same instance can be considered the same. This allows IMK to make good use of all of its clusters, as each cluster is used to express something about a bag. This makes the IMK robust to different datasets, only some of which adhere to the SMI assumption.

After reviewing the data, all of the implemented ES methods perform worse on *Musk1* than the image data sets, which is likely due to the complexity of the decision boundary required for classifications. DBOW and IMK performed well on all 3 data sets, which can be attributed to the combination of their use of clustering and their use of distance-based mapping functions. If I had more time, I would have chosen to investigate some of the approaches that imbedded DBOW into Adaboost, such at the approach in Auer et al. Overall, the ES approaches I implemented were fairly easy to implement and provide high accuracies relative to other paradigms.

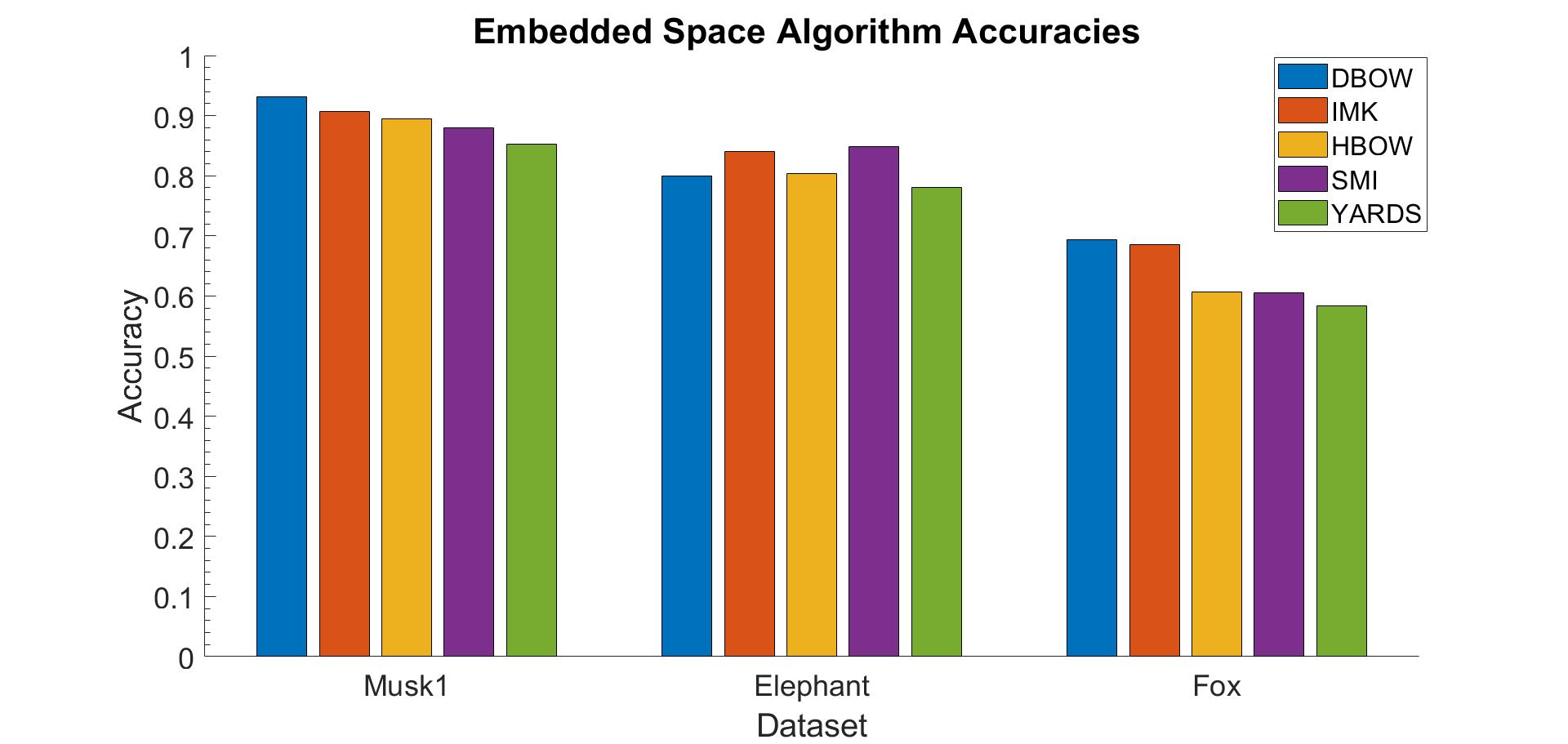


Figure 7: Final Accuracies

## References

[1] J. Amores, “Multiple instance classification: Review, taxonomy and comparative study,” *Artificial Intelligence*, vol. 201, 2013, pp. 81–105.

[2] Dong, L. A Comparison of Multi-instance Learning Algorithms, Master’s Thesis, University of Waikato, 2006

[3] Gärtner, Thomas, Peter A. Flach, Adam Kowalczyk, and Alexander J. Smola. "Multi-instance kernels." In *ICML*, vol. 2, no. 3, p. 7. 2002.

[4] Van Gemert, Jan C., Cor J. Veenman, Arnold WM Smeulders, and Jan-Mark Geusebroek. "Visual word ambiguity." *IEEE transactions on pattern analysis and machine intelligence* 32, no. 7 (2009): 1271-1283.

[5] Foulds, James Richard. "Learning instance weights in multi-instance learning." PhD diss., The University of Waikato, 2008.

[6] Auer, Peter, and Ronald Ortner. "A boosting approach to multiple instance learning." In *European Conference on Machine Learning*, pp. 63-74. Springer, Berlin, Heidelberg, 2004.

[7] Boughorbel, Sabri, Jean Philippe Tarel, and Nozha Boujemaa. "The intermediate matching kernel for image local features." In *Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005.*, vol. 2, pp. 889-894. IEEE, 2005.

## Embedded Space Clustering Algorithms- Anna Sedlackova

### Introduction

#### Use of clustering in the Vocabulary-Based Embedded Space (ES) Paradigm

In this section, we compare several clustering algorithms and their benefits towards the Embedded Space paradigm of Multiple Instance Learning (MIP). The embedded space paradigm focuses on extracting global information about each bag via an embedding that summarizes the bag’s content in a manner that can easily be operated on by standard classifiers [1]. The taxonomy for the ES paradigm is visible in Figure 1 below. It is split into vocabulary-based and not vocabulary-based categories, where vocabulary-based methods use classified instances of the bags in order to obtain the embedding. In popular vocabulary-based methods, the instances are assigned classes that are discovered in an unsupervised manner via traditional clustering algorithms such as k-means, k-median, etc. The clustering is performed over all instances from all bags, and then the cluster assignments are used to classify each instance.



Figure 1. Taxonomy of the Embedded Space paradigm adapted from [1].

At a high level, all vocabulary-based ES methods consist of three important parts:

1. A vocabulary V is a set storing information about K classes of instances. These classes usually correspond to a set of sensible prototypes, archetypes, and/or groupings that the instances fall into. For example, if a bag consists of documents, these classes might be the genre of each document (i.e. politics, economics, etc.). Each class has an identifier and is described by parameters , which might be the mean or standard deviation of the *j*th class of instances. Generally, each class corresponds to a cluster.
2. A mapping function which takes a bag *X* and the vocabulary V and maps the bag to a K-dimensional feature vector that summarizes the bag’s contents in a manner that can be easily operated on by standard classifiers.
3. A standard supervised classifiers such as Adaboost, SVM, random forests, etc. These classifiers use the summarized feature vectors obtained from the embedding and learn a discriminative function to separate bags on the basis of their summary vectors.

One of the methods in the vocabulary-based ES paradigm is known as Histogram-based Bag-of-Words (H-BoW). In this method, the K clusters in vocabulary V function as general prototypes. The mapping function M then counts the number of instances in each of the K prototypes, which yields a histogram that summarizes the bag’s contents. For example, if a bag contains a collection of documents, the histogram will contain counts of the number of documents in each genre (i.e. politics, economics, etc.).

A complementary method to the H-BoW algorithm is the Distance-based Bag-of-Words (D-BoW), which is closely related to the former. The H-BoW method uses the vocabulary to determine the counts of instances in each of the K clusters, producing a histogram or count vector. In the D-BoW, the vector consists of the minimum distance of all bag instances to each cluster centroid. This is advantageous because the distance function can be selected to achieve various desirable properties, e.g. the norm can be used to improve robustness to outliers.

In this work, we examine the performance of H-BoW and D-BoW methods when different clustering algorithms are applied to initially construct the vocabulary. The clustering algorithms we focus on are (1) k-means, (2) k-medians, (3) mean shift, and (4) fuzzy c-means. We perform numerical experiments on several standard datasets in the literature and compare the overall performance of the bag classifier.

#### Summary of Clustering Algorithms Investigated

##### k-means

The first method we investigate is k-means clustering algorithm as suggested by the works of Nowak et al. Nowak et al. addressed the problem of effective representations for automatic image categorization by using hard assignment clustering methods, specifically k-means, with a Bag-of-Words model to classify images [2]. k-means is a simple four-step algorithm.

First, the algorithm starts with initial estimates for the Κ centroids, which are chosen at random. If we have a set of centroids, . Each centroid defines one of the clusters. Each instance of a bag is now assigned to its closest cluster based on its Euclidian distance , where the distance function is the Euclidian distance. In the second to last step, we recompute the centroids by taking the mean of all instances assigned to each centroid such that, with being the set of points assigned to *i*th cluster. Now, the clustering algorithm will keep iterating between these two steps until we sufficiently minimize the distances, or the centroid coordinates do not change much between iterations.

In order to compute the best number of clusters, we track the squared error function below and adjust the number of clusters to see which minimizes it sufficiently.

where J is the objective function, k is the number of clusters, n is the number of instances (from all bags), and j represents the centroid cluster j

There are, however, a few issues with k-means and it is not always the best clustering algorithm to use (it does, however, run very quickly compared to some of the other methods in the report). It does not use a kernel function and often tends to be susceptible to outliers because the centroids are computed by taking the mean over all points in each cluster. It also tends to make spherically symmetric clusters (unlike mean shift clustering, which can do better when clusters are not symmetrical).

##### k-median

We also explore another clustering algorithm, k-medians, based on the works of Sivic et al. who experimented with text retrieval to object matching in videos using various clustering techniques [3]. They assign each frame to the nearest cluster and then generate matches for all frames throughout the video. k-medians tends to be less susceptible to outliers then k-means and is a very similar algorithm with the exception of the distance function. Each instance of a bag is now assigned to its closest cluster based on its Manhattan distance rather than the Euclidean distance, . The squared error function would also use Manhattan distances. Because we are no longer using the mean, the algorithm is less susceptible to outliers, which owes to the robustness of the median.

##### Mean shift

Mean shift is a hard-assignment method that has a lot of flexibility because we can choose different kernels such that we can maximize spatial or density preferences. If we want to compute centroids in a manner that weighs assigned points asymmetrically, this method can accommodate this objective and hence can yield asymmetric clusters. Unlike with k-means where we take a mean of points assigned to cluster, which then becomes the new cluster center, mean shift takes weighted mean using a kernel function where x represents the initial cluster centers. For testing purposes, we restrict ourselves to the Gaussian kernel shown above. For each iteration, we perform until conversion such that:

where N(x) are the neighbors of x, instances where

The strengths of mean shift lie in its ability to handle arbitrary feature spaces and recompute cluster centroids in a more general manner. However, it is significantly slower () to run than k-means and its bandwidth parameter is not trivial to tune and often requires adaptive bandwidth sizes (usually obtained from cross-validation).

##### Fuzzy c-means

Lastly, we look at one soft clustering algorithm, fuzzy c-means, which has been implemented by Boughorbel et al. in order to cluster local features of training images [3]. The advantage of using FCM and other soft clustering models is that they leverage information from multiple clusters when making updates on the centroids. Specifically, the updates are computed as follows:

:

(Adapted from Boughorbel et al. [4])

The major disadvantage of the above-mentioned methods is their lack of flexibility in cluster shape (relative to kernel methods) and non-probabilistic cluster assignment weights. Specifically, those methods do not always perform well on low dimensional data sets. The datasets we investigate in this work have decently large dimensions (at 160, 143, and 143 dimensions, respectively).

### Methodology

#### Implementation

To implement and fully test various clustering techniques with the mapping function and classifiers, I have programmed the four clustering methods from scratch in Python. I then slightly modified Eric Miller’s implementations of Histogram and Distance-Based Bag-of-Words models to take various clustering methods as input and imported Scikit Learn’s SVM implementation. I have also modified the cross-validation file from the previous projects to be compatible with testing different clustering methods. The parameters used for clustering, mapping and classifying are explained below.

#### Data Sets Used

The same datasets have been used as the one described in the introduction. The data has been standardized for the Distance-based and Histogram-based Bag-of-words mapping functions.

#### Experimental Procedure

Because the clustering algorithms are likely to converge to local optima, it’s important to alter the centroid initialization randomly. Therefore, our computations are done several times. Below I mention all the crucial setup parameters for the various clustering algorithms, mapping functions, and classifiers.

##### Number of cross-validation folds and iterations

For the Musk1 database, I used a ten-fold cross-validation approach run with ten iterations. Each time I have used 90% of the data set for training and 10% testing this way I had in total 100 rounds. This was done to see how my results compare to Amores et al.

For the Fox and Elephant databases, I used five-fold cross-validation with five iterations because it has triple the amount of instances and took too long to complete within the given time frame of the project.

##### Classifier parameters

All of my Histogram-based Bag-of-Words tests were run with the SVM classifier from Scikit Learn. I decided to use the default RBF kernel (extended Gaussian kernel), , where D is the distance function and Euclidian distance, . I used gamma set to auto for Histogram-based Bag-of-Words and the default C value.

For my Distance-based Bag-of-Words tests I used SVM with the distance function and gamma equal to scale. The Euclidian distance was used as well as the default C value.

##### Number of clusters

In order to find the optimal number of clusters, Amores et al. computed ten vocabularies for each vocabulary size using different random initializations. However, that strategy would create an enormous feature vector (of length ~ 5000 features) and takes very long to run for all my various experiments so I have decided to use the elbow method. Initially, I have graphed the sum squared error vs the number of clusters for all three data sets and k-means, k median to determine the best number of clusters to use with each data set and clustering algorithm.

Specifically, I ran those tests with the following parameters: SVM classifier, distance function, Histogram-based bag-of-words model, default C-value and 5 fold cross-validation.

|  |  |
| --- | --- |
| Chart | Chart |
| Chart | Chart |
| Chart | Chart |
| Fig 2. Investigating the optimal number of cluster centers for k-means and k-median using the elbow method. | |

Based on the graphs above I have decided to investigate the following cluster sizes:

|  |  |  |  |
| --- | --- | --- | --- |
|  | MUSK1 | Fox | Elephant |
| Number of clusters for k-means, k-median, FCM | 64, 128 | 256, 512 | 128, 256 |

Table 1. Cluster numbers for various data sets investigated

##### Determining other clustering parameters

For mean shift, I have decided to pool the bandwidth values between 2-8 for each data set to find the optimal value. Generally, the bandwidth value of 2 and 4 performed the best for the datasets.

In order to decide the correct value of the fuzzing variable for Fuzzy c-means, I have looked at the results of Schwämmle et al. who analyzed the optimal value for different data sets based on their size and dimensionality [5]. A threshold of 0.01 is sufficient for N ≥ 100 and D ≥ 5. I have decided to investigate the fuzzy parameter values of 1, 2 and threshold values of 0.01.

### Results

Altering the fuzzy parameter from 1 to 2 resulted in a small difference, so I have decided to not plot those results. However, the result of changing the bandwidth for each data set is in Figure 4.

|  |  |
| --- | --- |
| Chart | Chart |
| Chart | |
| Fig 3. Results from clustering methods and three data sets using the histogram and distance-based BoW. **Top left**: Musk1 performance with k-means and k-median clusters =128, mean shift bandwidth = 4, FCM fuzzy parameter = 2  **Top right**: Fox performance with k-means and k-median clusters =256, mean shift bandwidth = 4, FCM fuzzy parameter = 2  **Bottom**: Elephant performance with k-means and k-median clusters =256, mean shift bandwidth = 4, FCM fuzzy parameter = 2 | |

|  |  |
| --- | --- |
| Chart | Chart |
| Chart | |
| Fig 4. Results from various bandwidth values methods with mean shift for the three datasets  **Top left**: Musk1 mean shift performance 3 different bandwidth values  **Top right**: Fox mean shift performance with 4 different bandwidth values  **Bottom**: Elephant mean shift performance with 4 different bandwidth values | |

### Discussion

#### Results interpretation & Relating to other studies

##### Comparing Clustering Algorithms

It appears that mean shift is the best performing algorithm on all data sets except for the histogram-based BoW ran on the elephant data set. Mean shift can handle asymmetric clusters better than k-means and k-median as well as high dimensional data, which explains why it has a better performance. While tuning the bandwidth parameter I noticed that the optimal value is either radius of 2 or 4. This parameter controls how far the kernel reaches. If the bandwidth is large it will give a higher weight to points that are further away, but if the bandwidth is smaller it weights points that are close by.

K-means performed fairly poorly on the data with k-median being slightly better. That is possibly due to the fact that k-means and k-median cannot handle nonsymmetric clusters and our data sets-Musk1, fox and elephant likely contain asymmetric clusters.

Fuzzy c-means seems to be consistently the worse performer with the exception of the elephant data set. The algorithm would likely perform better on Musk1 and Fox if the clusters were comparably sized. Altering cluster sizes can pose an issue because smaller clusters can lead to unreliable information and skew the weights of the membership matrix and push the results away from larger cluster sizes in a disadvantageous manner.

##### Comparing Distance Functions

The difference of results between the histogram and distance-based bag of words can be explained due to a lack of tuning of the C value of the SVM classifier. I have also not altered the distance function and kept it Euclidean throughout the entire data collection. I attempted to keep my data consistent all across to compare the performance of the algorithms to one another rather than attempting to get the highest accuracy.

##### Comparing Data Sets

One of the issues we have come across with Musk1 is massive standard deviation values of our confusion metrics. The entire data set only has 92 bags which means that we are testing on <10 bags during cross-validation. However, Musk1 is also the data set with the highest average accuracy. Unlike Musk1, fox and elephant are image data sets and prove to be much more difficult to classify. However, this is consistent for all Multiple Instance algorithms that we have tested.

##### Critical Difference Diagram

****

Fig 5. Critical difference Diagram for 95% confidence interval for Musk1

Distance-based BoW highlighted in red, Histogram-based in yellow for clarification

I have decided to see which algorithms were statistically different for Musk1. Fig 5. displays a CDD with a 95% confidence interval. We can see that k-median and k-means are not different with 95% confidence for the Histogram-based BoW. The same can be said about Distance-based BoW k-means, k-median and Histogram-based Fuzzy c-means.

#### Conclusion

Clustering helps find latent structure in the data and is crucial for our Bag-of-Words models because it determines the vocabulary. Through investigating various methods on the Musk1, fox and elephant data sets, I have found mean shift to be the best performing algorithm although it requires significantly more computational time. Overall, the K-means performance is similar to the one by Amores et al. and in our group conclusion, we show how after tuning the C parameter and distant metric of the SVM classifier, mean shift performs the best out of all other clustering techniques and mapping functions as done by Eric in his mapping functions analysis.

### References

[1] J. Amores, “Multiple instance classification: Review, taxonomy and comparative study,” *Artificial Intelligence*, vol. 201, 2013, pp. 81–105.

[2] Nowak, Eric, Frédéric Jurie, and Bill Triggs. "Sampling strategies for bag-of-features image classification." In European conference on computer vision, pp. 490-503. Springer, Berlin, Heidelberg, 2006.

[3] Sivic, Josef, and Andrew Zisserman. "Video Google: A text retrieval approach to object matching in videos." In null, p. 1470. IEEE, 2003.

[4] Boughorbel, Sabri, Jean Philippe Tarel, and Nozha Boujemaa. "The intermediate matching kernel for image local features." In Proceedings. 2005 IEEE International Joint Conference on Neural Networks, 2005., vol. 2, pp. 889-894. IEEE, 2005.

[5] Schwämmle, Veit, and Ole Nørregaard Jensen. "A simple and fast method to determine the parameters for fuzzy c–means cluster analysis." Bioinformatics 26, no. 22 (2010): 2841-2848.

## 

## Bag Space Algorithms

## Kyle Pham: Bag-Space paradigm (K-Nearest Neighbors algorithms)

In this section of the paper, we examine the Bag Space Paradigm (BS) of Multiple Instance Learning. Unlike the Instance Space Paradigm, BS methods embed the content of the bags into a single feature vector, and transform the problem into supervised learning. BS treats the bags *X* as a whole, and the discriminant learning process is performed in the space of bags as supervised learning. BS methods then use distance metrics to compare different bags from the data and then use the distance function for a distance-based classifier such as kNN or SVM [1].

Each bag *X* can be considered as just a set of points in a *d*-dimensional space. Thus, we only need to come up with a metric that can compare the two sets of points *X* and *Y* in this context. A lot of distnace measues have been used, such as the Earth Movers distance [2], the Chamfer distance [3], and the minimal Hausdorff distance [4], which is used in my implementations.

Formally, if we let A = {a1, a2, …, am} and B = {b1, b2, …, bn}, then the Hausdorff distance is defined as

, where

The minimal Hausdorff distance, which is an extension of the Hausdorff distance, is defined as

, where is the Euclidean distance between two instances based on their features.

After having defined a distance measure, we then use a distance-based classifier for classification. For the project, I implemented the k-Nearest Neighbors (kNN) algorithm, the Citation k-Nearest Neighbors (Citation-kNN) from the seed paper [1] and the paper [4]. Then, I suggest an improvement on both of the algorithms by adding in weight factors in the classification step to create Weighted kNN and Weighted-Citation-kNN. Two other methods using SVM (MI-SVM and mi-SVM) are implemented by Gary Yao in our group.

### kNN Algorithm

**1. Introduction and Methodology**

The kNN algorithm is a lazy learning algorithm, where training examples are simply stored for future use rather than used to construct general and explicit description of the target function. The training examples will be stored so that for each testing example, we can get its distance from these training examples and find which k training examples it is closest to. However, in the context of multiple instance learning, we will pick the k bags that are the closest to a test bag. The bag will then be classified as the label that appears the most among these k nearest bag neighbors.

The performance of the algorithm depends on several factors. Firstly, the distance metric for bags is an important factor. If each bag contains many instances, distances between points from two bags have to be incorporated in the distance between the two bags. If the distances are significantly different, then some information might be lost. Secondly, the constant k is very important, as we consider more neighboring bags to classify a test bag. If we only consider a few neighbors, then only a few of the closest bags can have an influence on the predicted label of the bag. If we consider too many neighbors, then maybe the predicted label of the test bag is influenced by many bags at the same time, thus making the distance between bags less important in determining the predicted label. The best value for k is dependent on the dataset, for example, how complex it is, how many bags there are, and how many instances are there in each bag.

**2. Experimental Results**

We test the kNN algorithm on three different datasets: Musk1, Elephant, and Fox. We use 10-fold cross validation one time for each of the dataset, and keep track of the average accuracy, precision, and recall of the classifier. Furthermore, for each of the dataset, we will use different values of k from 1 to 6, and see which value works the best.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | k = 1 | k = 2 | k = 3 | k = 4 | k = 5 | k = 6 |
| Musk1 | Accuracy | **91.22** | 80.44 | 80.33 | 75.89 | 80.33 | 72.67 |
| Precision | **89.33** | 74.49 | 75.39 | 70.81 | 74.68 | 67.84 |
| Recall | 95.50 | **100.00** | 96.00 | **100.00** | **100.00** | **100.00** |
| Elephant | Accuracy | 71.00 | 69.50 | 75.50 | 72.50 | **76.00** | 74.50 |
| Precision | 70.00 | 63.63 | **71.63** | 67.08 | 71.62 | 68.26 |
| Recall | 75.00 | 94.00 | 88.0 | 93.00 | 90.00 | **97.00** |
| Fox | Accuracy | 60.00 | 58.50 | **63.50** | 62.00 | 61.00 | 62.00 |
| Precision | 61.03 | 56.23 | **66.23** | 58.61 | 59.96 | 58.87 |
| Recall | 63.00 | 82.00 | 65.00 | **85.00** | 68.00 | 82.00 |

Table 1 - kNN performance on the three different datasets

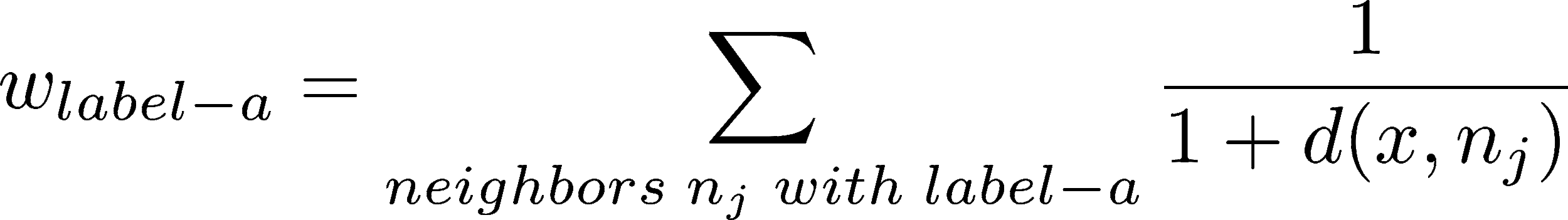
**3. Discussion**

From this table, we can see that the best k value for the kNN algorithm is dependent on the datasets. Musk1 is a relatively simple dataset (92 bags, 460 instances, 5 instances per bag) so the value of k = 1 works well. For more complex datasets like Elephant (200 bags, 1391 instances) and Fox (200 bags, 1320 instances), we want to consider more neighbors to determine the label of each bag.

### Weighted-kNN Algorithm

**1. Introduction and Methodology**

A common and practical improvement of the kNN algorithm can be implemented where we add in weights to the k nearest neighbors. This is based on the intuition that a closer neighbor should have a higher impact on determining the label of the current example. To do this, we can set up a weight function that has an inverse correlation with distance [5]. There are many ways to do this. Here, we simply add to the weight for each label by the reciprocal of the distance:

[](http://www.texrendr.com/?eqn=w_%7Blabel-a%7D%20%3D%20%5Csum_%7Bneighbors%20%5C%3B%20n_j%20%5C%3B%20with%20%5C%3B%20label-a%7D%20%5Cdfrac%7B1%7D%7B1%20%2B%20d(x%2C%20n_j)%7D%0)

, where *x* is the current example being labeled, *wlabel-a* is the weight of *x* being of label *label-a*, and *d* is the distance function.

Here, for the context of kNN algorithm for multiple instance learning, the examples and the neighbors are bags and the distance measure is the distance between bags instead of instances.

**2. Experimental Results**

We run the Weighted-kNN algorithm on the three datasets Musk1, Elephant, and Fox again, still using the Hausdorff distance for the distance between bags. We use 10-fold cross validation one time for each of the dataset, and keep track of the average accuracy, precision, and recall of the classifier. Furthermore, for each of the dataset, we will use different values of k from 1 to 6, and see which value works the best.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | k = 1 | k = 2 | k = 3 | k = 4 | k = 5 | k = 6 |
| Musk1 | Accuracy | **91.22** | **91.22** | 81.44 | 88.11 | 81.44 | 83.67 |
| Precision | **89.33** | **89.33** | 76.33 | 84.10 | 75.24 | 77.62 |
| Recall | 95.50 | 95.50 | 96.00 | 98.00 | **100.00** | **100.00** |
| Elephant | Accuracy | 71.00 | 71.00 | 75.50 | 74.50 | **76.00** | **76.00** |
| Precision | 70.00 | 70.00 | 71.63 | 71.22 | 71.62 | **71.86** |
| Recall | 75.00 | 75.00 | 88.00 | 86.00 | **90.00** | 88.00 |
| Fox | Accuracy | 60.00 | 60.00 | **63.50** | 64.50 | 61.00 | 66.00 |
| Precision | 61.03 | 61.03 | **66.23** | 66.66 | 59.96 | 65.30 |
| Recall | 63.00 | 63.00 | 65.00 | **67.00** | 68.00 | 73.00 |

Table 2 - Weighted-kNN performance on the three different datasets

**3. Discussion**

As can be seen from the table, compared to the result of the normal kNN algorithm, Weighted-kNN shows some improvement for the kNN algorithm for the cases where k is even. This result is intuitively reasonable, since for kNN, where k is even, there are cases where the algorithm will have to randomly pick a lable if there are equal numbers of neighbors for each label. Adding in the weights for each neighbor helps so that the algorithm will prefer to pick the label where the neighbors are closer to the bag. This helps the algorithm more frequently pick the “better” label deterministically and possibly improve the accuracy of the algorithm. However, sometimes, the label of which the neighbors are closer to the current bag may not actually be the correct label for the bag. Thus, picking deterministically in Weighted-kNN will decrease the recall.

For an odd value of k, the number of cases where the algorithm will have to randomly pick a label is zero, since we only have two bag labels, either positive or negative, so one label will appear more than the other amount an odd k neighbors.

### 

### Citation-kNN Algorithm

**1. Introduction and Methodology**

We then consider another variation of the standard kNN algorithm for the multiple instance learning problem, given by Wang, Jun, and Jean-Daniel Zucker. The inspiration of Citation-kNN was from the notion of *citation* from information science. In this domain, finding related documents are important, the relation between documents are not necessarily bidirectional. For a research paper, if it appears or is cited by another paper, then the other paper is called a *citer*. If the paper cites or references another paper, then the other paper is called a *reference*. Thus, citers and references are related to or neighbors of the research paper, but in different manners.

Wang, Jun, and Jean-Daniel Zucker suggests that this technique can be incorporated into kNN for multiple instance learning by taking into account not only the neighbors of a bag B, but also the bags that count B as a neighbor. Here, we can define the *r-nearest references* of an example B as the *r-nearest neighbors* of B using the same distance metric as kNN (Hausdorff distance). For the citers, defining nearest neighbors can be more complex. Here, the authors use ranking to determine the citers of an example B.

Let n be the number of all example bags BS = {B1, B2, B3, …, Bn}. Then, for example B ∈ BS, all other examples BS \ B = {Bi | Bi ∈ BS, Bi ≠ B} can be ranked according to how it is similar to B. For each example B, define the rank number to be *Rank(B’, B)*, with Rank(B, B) = ∞. Then, for any example B in BS, we define the *c-nearest citers* of B to be *Citers(B, C)*, where Citers(B, C) = {Bi | Rank(Bi, B) ≤ c, Bi ∈ BS}. Alternatively, instead of using the rank, we can use a different distance measure instead.

We further demonstrate the concepts of references and citers by giving an example. For example, assuming there are 6 bags of instances {B1, B2, B3, B4, B5, B6} in a data set. The nearest neighbors are shown in the following table.

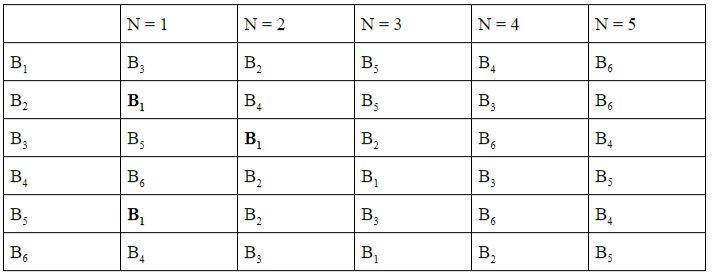


Table 3 - The nearest neighbors of 6 bags {B1, B2, B3, B4, B5, B6} with N being the nearest rank number, adapted from Wang & Zucker (2000)

In this example, if we let r = c = 2, then for the bag B1, its r-nearest references are B3 and B2, while its c-nearest citers are B2, B3, and B5.

Now concentrate on how to combine the r-nearest references and the c-nearest citers of an unseen bag b to derive its class. For the R-nearest references, let the numbers of positive and negative bags be Rp and Rn respectively. For the c-nearest citers, let the numbers of positive and negative bags be Cp and Cn respectively. Let p = Rp + Cp and n = Rn + Cn. If p > n, then a bag B is classified as positive and otherwise it is negative.

**2. Experimental Results**

For the implementation of Citation-kNN, we slightly modify the implementation of the standard kNN by adding in Citers and References, and then follow the rules presented in the methodology section to determine the label of each test bags.

We then run the Citation-kNN algorithm on the three datasets Musk1, Elephant, and Fox again, still using the Hausdorff distance for the distance between bags to find the r-nearest references. We use 10-fold cross validation one time for each of the dataset, and keep track of the average accuracy, precision, and recall of the classifier. Furthermore, for each of the dataset, we will use different values of r from 1 to 6 and corresponding values of c = r + 2, and see which value works the best. The main intuition for picking c = r + 2 is to reflect the fact that citers are usually more references. The label of a bag should be more influenced by the citers, which influence the bag, than the bags that it influences.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | r = 1 | r = 2 | r = 3 | r = 4 | r = 5 | r = 6 |
| Musk1 | Accuracy | 89.00 | 87.99 | 85.78 | **90.11** | 86.89 | **90.11** |
| Precision | 90.17 | **91.00** | 86.83 | 87.81 | 84.81 | 88.14 |
| Recall | 88.50 | 84.50 | 87.50 | **96.00** | 94.00 | **96.00** |
| Elephant | Accuracy | 72.50 | 79.00 | 80.00 | 80.50 | 80.50 | **81.50** |
| Precision | 75.63 | **81.32** | 78.86 | 78.86 | 78.27 | 79.80 |
| Recall | 67.00 | 77.00 | 84.00 | 84.00 | **87.00** | 86.00 |
| Fox | Accuracy | 60.50 | 64.00 | 62.50 | 62.50 | 57.50 | **65.00** |
| Precision | 62.23 | **69.54** | 64.96 | 63.27 | 57.30 | 65.72 |
| Recall | 55.00 | 54.00 | 56.00 | 61.00 | 56.00 | **62.00** |

Table 4 - Citation-kNN performance on the three different datasets, with c = r + 2

**3. Discussion**

As can be seen from the result Table 4 and Table 1 for normal kNN, the Citation-kNN algorithm returns better results in the Elephant (Citation-kNN: 81.50% accuracy and kNN: 76.00% accuracy) and Fox datasets (Citation-kNN: 65.00% accuracy and kNN: 63.50%), but it returns worse result in the Musk1 dataset (Citation-kNN: 90.11% accuracy and kNN: 91.22% accuracy). We believe the difference is based on the datasets. For Musk1, which is a simple dataset (92 bags, 460 instances, 5 instances per bag), a simple structure with only neighbors is enough, and a more complicated structure with citers and references may do more harm. With more citers and references, the label of a bag may be badly influenced by the irrelevant bags and thus the accuracy decreases compared to the kNN algorithm. For the Elephant and Fox datasets, which contain many more bags, instances, and instances per bag, a more complex structure like in Citation-kNN is preferred, as the bags have more complicated relations that is not represented by the simple minimal Hausdorff distance in kNN. In fact, if there are more instances in each bag, the minimal Hausdorff distance is equal to the distance between a pair of instances, one from each bag. This ignores a lot of information about the other instances in each bag, and thus, with less information, the Hausdorff distance will not capture a lot of relations between the bags in the datasets, thereby leading to a lower accuracy.

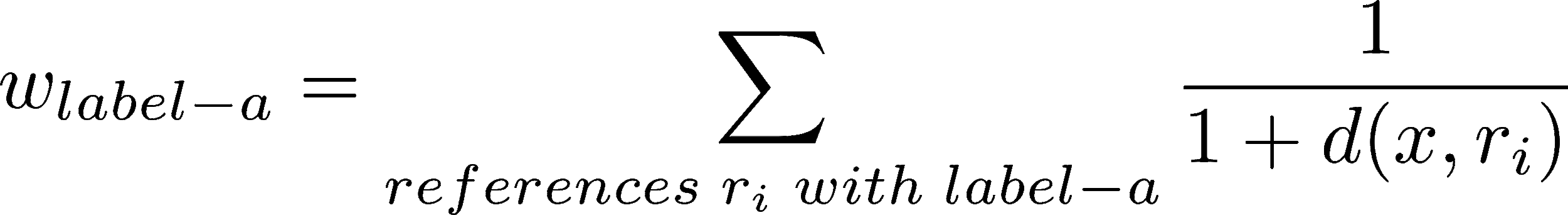
Additionally, one can notice that the results for recall of Citation-kNN is about 10% less than that of kNN, trading off for about a 10% increase in precision. The main reason for this is also probably because of the more complex structure of Citation-kNN. Since there are more factors influencing the label of a bag in Citation-kNN, the precision of the algorithm is higher. However, as it requires more information to determine the label of a bag, the recall of the algorithm is less than kNN, with less generalization made. This is also similar to our observation for the Weighted-kNN algorithm compared to kNN, where Weighted-kNN also add more complexity to the kNN model.

Another disadvantage of the Citation-kNN algorithm compared to kNN is the runtime. Since we have to calculate the rank to determine the c-nearest citers in addition to the Hausdorff distance for the r-nearest references, the runtime of training and testing the Citation-kNN algorithm increases significantly compared to kNN. But overall, Citation-kNN is better than kNN in complex datasets because of its more complex structure.

### Weighted-Citation-kNN

**1. Introduction and Methodology**

We further improve the Citation-kNN algorithm by adding in a weight system to each label for each bag as we did in Weighted-kNN. The weights can be similarly determined as the weights in Weighted-kNN. Since there are citers and references, we also can add to the weights differently, based on the assumption that citers are more impactful than references. Here, we can define the weight of a label to be:

[](http://www.texrendr.com/?eqn=w_%7Blabel-a%7D%20%3D%20%5Csum_%7Breferences%20%5C%3B%20r_i%20%5C%3B%20with%20%5C%3B%20label-a%7D%20%5Cdfrac%7B1%7D%7B1%20%2B%20d(x%2C%20r_i)%7D%20%0)[](http://www.texrendr.com/?eqn=%20%2B%20%5Csum_%7Bciters%20%5C%3B%20c_j%20%5C%3B%20with%20%5C%3B%20label-a%7D%20%5Cdfrac%7B1%7D%7B1%20%2B%20d(x%2C%20c_j)%5E2%7D%20%0)

, where *x* is the current example being labeled, *wlabel-a* is the weight of *x* being of label *label-a*, and *d* is the distance function.

**2. Experimental Results**

We then run the Weighted-Citation-kNN algorithm on the three datasets Musk1, Elephant, and Fox again, still using the Hausdorff distance for the distance between bags to find the r-nearest references. We use 10-fold cross validation one time for each of the dataset, and keep track of the average accuracy, precision, and recall of the classifier. Furthermore, for each of the dataset, we will use different values of r from 1 to 6 and corresponding values of c = r + 2, and see which value works the best. The main intuition c = r + 2 is previously described for Citation-kNN.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  |  | r = 1 | r = 2 | r = 3 | r = 4 | r = 5 | r = 6 |
| Musk1 | Accuracy | **90.11** | 85.78 | 84.67 | 89.11 | 85.78 | 89.11 |
| Precision | **88.00** | 85.00 | 82.19 | 85.29 | 81.81 | 84.52 |
| Recall | 95.50 | 89.00 | 94.00 | 98.00 | 96.00 | **100.00** |
| Elephant | Accuracy | 75.00 | **82.11** | **82.11** | 82.00 | 81.50 | 80.22 |
| Precision | 73.63 | **79.32** | 76.22 | 76.86 | 76.11 | 78.44 |
| Recall | 73.00 | 80.00 | 85.50 | 85.78 | **87.00** | 86.00 |
| Fox | Accuracy | 65.50 | **68.11** | 64.67 | 65.22 | 61.00 | **68.11** |
| Precision | 63.23 | **69.89** | 65.34 | 64.47 | 58.30 | 67.72 |
| Recall | 58.00 | 56.24 | 57.33 | 62.23 | 57.85 | **63.24** |

Table 5 - Weighted-Citation-kNN performance on the three different datasets, with c = r + 2

**3. Discussion**

As can be seen from Table 4 and Table 5, comparing Citation-kNN and Weighted-Citation-kNN, there is a considerable improvement overall for Weighted-Citation-kNN. Generally, there is a 2-3% improvement for each of the measure (accuracy, precision, and recall) for each of the dataset. The main reason for this is because we were able to differentiate references and citers using weights. Citers, using our formula for the weights, have more influence than references. Thus, with a more complex structure compared to Citation-kNN and kNN, we get a better result overall than Citation-kNN.

The runtime for Weighted-Citation-kNN is similar to Citation-kNN, since adding the weights is not significantly expensive in terms of calculation. Therefore, adding weights to Citation-kNN seem to be a good practice overall

### Overall Discussion

Overall, after examining the different variations of the kNN algorithm (kNN, Weighted-kNN, Citation-kNN, Weighted-Citation-kNN), we can see that Weighted-Citation-kNN is generally the best algorithm for complex datasets such as Fox and Elephant. However, for a more simple dataset such as Musk1, a simple algorithm such as kNN is good enough, as the simple structure of the algorithm fits well with the simple structure of the dataset.

As for the distance measure, the min Hausdorff distance works well for the Musk dataset, with only a few instances per bag, but not on the Elephant and Musk datasets, with a lot of instances per bag. Since the Hausdroff distance is equal to the distance between two instances, one from each bag, a lot of information about other instances in the bags is ignored. Thus, with a large number of instances, we lose information about the bags and the instances, another distance function should be used that takes into account more than thus an instance from each bag. Thus, we may need to use a different weight metric for the nearest neighbors algorithms as well as for BS algorithms overall.

Furthermore, the weight extensions in Weighted-kNN do not necessarily improve the performance and accuracy compared to the normal kNN as well as the results from the seed paper. The main reason for this might be that the distance between instances in two bags are similar, and thus incorporating the weights based on the distances are not changing the label result much. Additionally, for an odd value of k, the number of cases where the algorithm will have to randomly pick a label is zero, since we only have two bag labels, either positive or negative, so one label will appear more than the other amount an odd k neighbors. However, the weight extensions in Weighted-Citation-kNN considerably improves the results as it further favors the citers over the references in the complex structure designed for this. Thus, we can conclude that adding weights to kNN algorithms might be a good consideration.

### References

[1] J. Amores, “Multiple instance classification: Review, taxonomy and comparative study,” *Artificial Intelligence*, vol. 201, 2013, pp. 81–105.

[2] Zhang, Jianguo, et al. "Local features and kernels for classification of texture and object categories: A comprehensive study." *International journal of computer vision* 73.2 (2007): 213-238.

[3] Belongie, Serge, Jitendra Malik, and Jan Puzicha. "Shape matching and object recognition using shape contexts." *IEEE Transactions on Pattern Analysis & Machine Intelligence* 4 (2002): 509-522.

[4] Wang, Jun, and Jean-Daniel Zucker. "Solving multiple-instance problem: A lazy learning approach." (2000): 1119-1125.

[5] Samworth, Richard J. "Optimal weighted nearest neighbour classifiers." *The Annals of Statistics* 40.5 (2012): 2733-2763.

## Gary Yao: Support Vector Machine for Multiple Instance Classification

As a part of our exploration of the Multiple Instance Learning Problem, I extensively researched support vector approaches to solving the Multiple Instance Learning Problem. Via explorating the datasets, we found that multiple instance tasks are mostly exclusively binary, and uses relatively small datasets. Given that our target datasets have feature counts comparable to number of bags and are small, I believed that support vector machines should result in the best performances within a reasonable runtime. To thoroughly exploring this classic machine learning model, I stepped across the paradigm of Multiple Instance Learning and compared how support vector machines are formulated in all three branches of multiple instance learning. In total, I implemented the miSVM and MISVM models under the instance learning paradigm; the HassdorfSVM, ChamferSVM and EMDSVM algorithms under the bag space paradigm and collaborated with Eric Miller’s (also in this group) implementations of embedded learning KBOW-SVM models to draw extensive conclusions of this approach to MIL.

### Bag Classification as a Support Vector Problem

The chief learning task of a Support Vector Machine is to find the intercepts and slopes of a label-separating hyperplane of largest margin of a linear separable dataset. However, two issues arise to this naive formulation. Firstly, most of the time, datasets are inseparable by a hyperplane. Secondly, finding extreme cases of separable hyperplanes often leads to less-than-optimal results in face of generalizability and cross validation. To tackle the linear inseparable issue, we introduce kernel functions to the classifying function to introduce additional dimensionality and non-linearity that help the classifier to find linearly separable planes. To tackle the overfitting issue, we add to SVM objective formulations slack terms that allow some “misclassifications” across margins. Together, a support vector machine learning task has full classification formulation:



Where ***ξi*** is positive only when sample i is misclassified, and data points have been projected by a kernel function to a higher dimension.

It is easy to see that the multiple instance learning tasks does not fit this native formulation of support vector machine learning tasks, because different bags have different multiples of features while SVMs have a set number of learned feature multipliers. There exist two manipulatable parts in the SVM formulation, the kernels and the objective function. Symmetrically, the instance-space paradigm of MIL manipulates the objective function based on the SMI Assumption, the bag-space paradigms manipulates kernel functions and embedded-space paradigms leaves the SVM formulation alone and instead reduce bagging dimensions to fit the SVM model[1]. Comparing SVMs approaches in MIL is a way to compare the three paradigms themselves and would provide key insights on the nature of the multiple instance learning problem.

#### Instance Learning: Reformulating the Objective Function

The key idea to the Instance Space paradigm is to build a classifier for individual instances whose classification supposedly leads to aggregated bag-wise classification. Many important aggregation method utilizes the SMI assumption[2]. Under the SMI assumption, a bag X is considered positive if and only if at least one of its content instance is positive. As such, we can formulate the label of a bag in terms of:

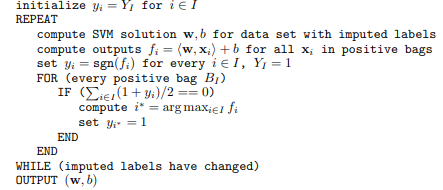
Using this assumption, Andrew, et. al[3], were able to derive two approaches formulate SVM models for multiple instance learning and two heuristic learning algorithms to optimize these models.

The first approach was simply named Multiple Instance Support Vector Machine (miSVM). In this mode, instances and models were assigned binary labels y, Y . Using the MSI assumption, an instance level classifier contributes to bag-level labels under the relation:

To find a classifier that maximally satisfy the eq. , the first SVM objective formulation was the following



This quadratic programming problems is under the classic formulation of the SVM optimization. The authors penalized the model for each instance whose decision function falls inside the margin. In other words, every instance wishes to have its decision function exceeding its parent’s label in absolute terms. This quadratic program was heuristically optimized using the following algorithm.

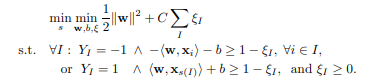


Simply put ,we repeatedly assign a soft label to every instance; and for those instances with mismatched labels from its parent, we attempt to trick the instance of the least mismatched decision function to flip over by changing its soft label for the next iteration. This si-SVM was the first of many models experimented in this section.

In the same paper, Andrew, et. al, provided another SVM formulation of the multiple instance learning problem that ignored the MSI assumption.



In this formulation, there exists one most significant instance within every bag that determine the classifier’s labeling of the entire bag. Such instances were named selector variable by the authors and would satisfy the constraints:  Thus, the authors derived the corresponding quadratic program:



The corresponding heuristic follows the same idea as the previous where we assign selector variables to the maximally indicating instance per bag and solve the quadratic program until selector variables become stable. This method is deeply dependent on the existence of selector variables per every bag and I wish we had the time to experiment on text-based datasets where we can manually compare the keywords of an article to the selected keywords of articles. This MI-SVM model was the second model experimented in this section.

#### Bag-Space: Formulation as Kernel Tricks

Constructing the instance-space paradigm, bag-space paradigms views bags as whole entities. In terms of SVM formulation, there now requires a way to transform every bag’s collection of instances into a vector that is sufficiently descriptive for classification. While this vectorization process implies a learning task, methods in the Bag-Space paradigms mostly avoids learning a globally optimal embedding multiple instances and instead focus on formulating pairwise relationships as description vectors for every bag.

Kernel functions in SVM often maps feature vectors onto higher, more linearly separable feature mappings[4]. Some simpler kernels uses combinations of polynomial products between features such as the polynomial kernel K(xi,xj) = (xi\*xj+1)^p. More complex kernels often inherently utilizes distances between distances such as the very widely used RGF kernel:

As such, Gartner et, al. experimentally discovered a kernel based on principles of convolutionary kernels, set kernels and normalization such that this kernel supports and specializes in projecting each bag’s distance to other bags onto a polynomial feature vector, based on a predefined multiple instance distance function.



Similar in idea, these three distance functions also proposed to project multiple instances onto a descriptive vector of distances to other bags.

1. Chamfer Distance[5]



1. Earth Mover Distance[6]



1. Hausdorff Distance[7]



The most interesting distance function among the three was the EMD measurement where a learnable multiplier indicates the importance and weakness between feature differences’. This weight matrix has some feature selection functionalities similar to artificial neural networks and filter feature selection models. Additionally, EMD creates an extremely expressive hypothesis space that contrasts the typical SVM. Thus, the seed paper indicated that the EMD+SVM method had by far the highest accuracy among all examined methods on most datasets. At the time of submission, Chamfer Distance and Hausdorff Distance were fully implemented and experimented to train two separate bag space models using linear kernels. Earth Mover Distance was partially implemented and will be analyzed in a later subsection.

#### Embedded Space: Finding Vector Representations of Multiple Vectors

As introduced in a previous section, another way to fit multiple instance learning into a SVM framework is to learn a way to represent the multiple instance vectors in every bag as a single characteristic vector in a globally optimal way. Otherwise, then translating bags into characteristic vectors, the SVM model is unaffected.

This paradigm was utilized in the IMF SVM algorithm, as multiple kernels were used in conjunction to project series of instances onto a single characteristic vector. This algorithm was implemented and experimented by Eric Miller previously and I will be referencing his results in analysis.

### Methods

To compare the three multiple instance learning paradigms, six models were considered in which I implemented five, and referenced Eric’s implementation for that other model. In this section, I will introduce my experimental conditions.

#### mi-SVM & MI-SVM

The paper greatly specific experimental conditions. The authors provided that they exclusively used radial basis function kernels, and specific hyper-parameters gamma and C. With minor hyperparameter tuning through increment, slight accuracy gain was achieved. With tuned hyperparameters, 3 iterations of 10 fold cross-validation were experimented and recorded.

#### SVM with Chamfer Distance and Hausdroff Distance

These methods were derived from separate studies and therefore lacked referenceable hyperparameters or kernel specifications. Projection onto distance matrices made hyperparameter calculations very difficult. As a result, I hyperparameter-tuned C with a linear kernel and tuned C and gamma with an RBF kernel. The tuning method was mostly multiple for loops enclosing a cross-validation with smaller increments in parameters as accuracy increases. The result was a C/Gamma combination on an RBF kernel, but I think this hyperparameter configuration is still far from optimal. At this configuration, each model experimented 3 iterations of 10 fold cross-validation.

#### EMD+SVM

The Earth Mover Distance was particular in that it had learnable parameters that need to be optimized along with the SVM model. At the time of submission, I have yet to implement such gradient descent into SVM. Nevertheless, I demonstrated the effects of a learnable distance weight matrix by pseudo-learning using 15 trials of training SVM models using randomized weight matrices. As with previous distance-based SVM, I used hyperparameter tuning prior to running the 15 trials.

#### IMF SVM

Please Refer to Eric’s Section.

#### Results

The six SVM models for MIL have the following accuracies on the Musk1, Fox and Elephant datasets.

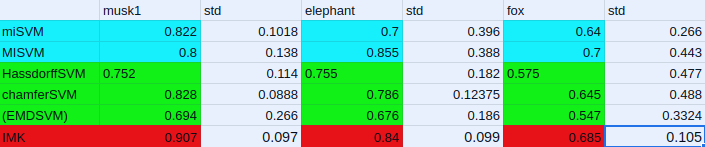


Figure 1. Models in blue are instance space. Models in Green are Bag Space approaches. Models in Red are Embedded Space Approaches.

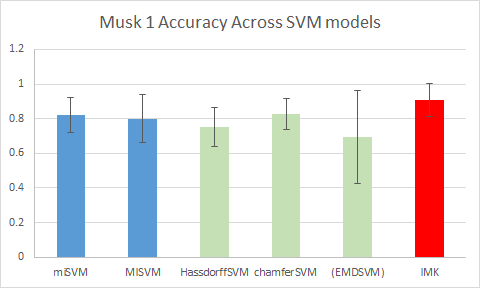


Figure 2. Bar graph for Accuracies on Musk1. The same color scheme for all models.

The Musk1 dataset is the smallest and has significantly more negative bags than positive bags. IMK easily performed the best among the six models. My implementations for miSVM and MISVM both matched the results of original papers. HassdorffSVM and ChamferSVM both underperformed by about one standard deviation. While EMDSVM vastly underperformed. I believe the bag space algorithms suffered from my rough hyperparameter tuning.

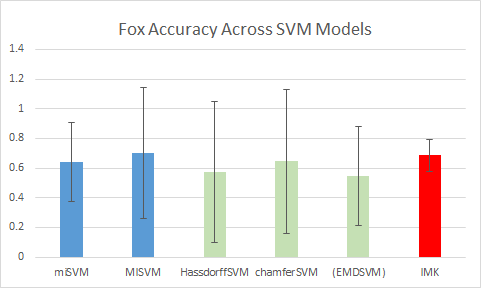
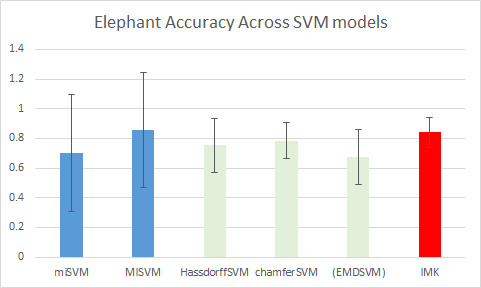


Figure3. Results to Elephant Dataset Figure4. Results to the Fox Dataset

The elephant and datasets showed that the MISVM algorithm produced the best accuracy. This result suggests that these datasets contained instances who are very good indicators of their parent bag, confirming image datasets’ tendency to be “middle focused”. Strangely, miSVM performed far worse than its similar cousin in MISVM. This may be a result of miSVM’s small model capacity. IMK remained a relatively strong algorithm throughout all experiments. Despite its longer running time, this algorithm creates both interpretable and strong models. Once again, I believe the bag space methods have far more potential if I had more time to finely tune parameters.

Overall, the MIL classification task indicated that the two more flexible parts of the SVM formulation are the objective formulation and input formulation. Kernels are often designed with mathematical backings on the distribution of the dataset and may not be easily modified for complex tasks.

### References

[1] J. Amores, “Multiple instance classification: Review, taxonomy and comparative study,” Artificial Intelligence, vol. 201, pp. 81–105, 2013.

[2] Solving the multiple-instance problem with axis-parallel rectangles T.G. Dietterich, R.H. Lathrop and T. Lozano-Pérez Artificial Intelligence, 89 (1997), pp. 31-71

[3] Andrews, Stuart, Ioannis Tsochantaridis, and Thomas Hofmann. Support vector machines for multiple-instance learning. Advances in Neural Information Processing Systems. 2002.

[4] Multi-instance kernels T. Gärtner, P.A. Flach, A. Kowalczyk and A.J. Smola Proc. of International Conference on Machine Learning (2002), pp. 179-186

[5]Local features and kernels for classification of texture and object categories: A comprehensive study J. Zhang, M. Marszalek, S. Lazebnik and C. Schmid Int. J. Comput. Vis., 73 (2) (2007), pp. 213-238

[6] Shape matching and object recognition using shape contexts S. Belongie, J. Malik and J. PuzichaIEEE Trans. Pattern Anal. Mach. Intell., 24 (24) (2002), pp. 509-522

[7] Solving the multiple-instance problem: A lazy learning approach J. Wang and J.-D. Zucker Proc. of International Conference on Machine Learning (2000), pp. 1119-1125

#### 

#### 

# 

# Part 3

## Results

To visualize how our results fit with one another based on the various data sets used, we have graphed all three types of algorithms - instance space, bag space and embedded space with the three different data sets. Additionally, for clustering, we have adjusted to match the C parameter to match the data collected by Eric to compare the accuracy of the highest performing clustering algorithm to the default.

|  |  |  |
| --- | --- | --- |
| Chart | Chart | Chart |
| Chart | Chart | Chart |
| Chart | Chart | Chart |
| Fig. 1 Accuracies of various algorithms compared to the three data sets  **Top Row**: MUSK-1 results for best performing IS, BS, ES algorithms  **Middle Row**: FOX results for best performing IS, BS, ES algorithms  **Bottom Row**: Elephant results for best performing IS, BS, ES algorithms | | |

## Comparative Study

From the paper, *Multiple Instance Classification: Review, taxonomy and comparative study* by Juan Amores, we worked in groups of two to implement algorithms from Instance Space, Bag Space, and Embedded Space. For instance space, Gareth, David, and Kyle implemented Axis-Parallel Rectangles, Diverse Density. For bag space, Gary implemented miSVM, MISVM, and distance based SVMs for a comparative study; and Kyle implemented HassdorfSVM, KNN, and citation KNN. For embedded space, Eric implemented H-BoW, D-BoW, YARDS, IMK, SMI and Anna implemented k-means, k-median, mean shift, and fuzzy c-means clustering algorithms. For these algorithms, we ran 10 fold cross-validation on various amounts of iterations for the datasets *Musk1*, *Fox*, and *elephant*. For the Musk1 dataset, the paper’s implementation of IS algorithms achieved between 70 and 90 percent, which matches our results. For BS, the paper found an accuracy of 85-90 percent, which our results match as well. For ES, the paper found an accuracy of 80-95 percent, which our results align with for the most part.

### Instance Based

Our results, as well as the results of the source paper, found that instance-level algorithms perform slightly lower than that of bag space and embedded space. This may be because of how instance classification works. Here, the procedure assumes the existence of a single “feature” that a conformation must have -- in other words that each attribute falls within the APR boundary -- to be classified as positive. However, if in truth there are multiple “boundaries” that the conformation can fall into, my algorithm won’t classify very accurately.

In addition, instance space maps instances individually and does not take into account groups in instance space -- which one might view as a cluster. A learning algorithm that operates on a higher level than instances can train data based on proximity to clusters of data and can use these “distance” measurements to classify new bags based on proximity. This also gives flexibility compared to the SMI assumption. The algorithm can learn whether the dataset demands SMI to correctly classify, or if bags require a certain number of positive instances.

The APR algorithm was not implemented in the seed paper, but many instance-level algorithms were. The all-positive, simple APR, and outside-in APR implementations were comparable in performance. The three algorithms found around 75% accuracy, and the seed paper’s algorithms found the accuracy of 70-90 percent for the different instance-level classifiers.

For the comparison between DD and EMDD, the paper achieved a result of 88.9 percent accuracy on the Musk1 data set and 82.5% accuracy on the Musk2 data set. For our testing, we replicate the testing on Musk1, but achieved a lower accuracy of 80 percent compared with the paper. The 88.9 percent performance seems underperformed than most other algorithms that are in the bagged space and embedded space, which are mostly above 90 percent. However, in the DD paper, a result graph shows that running on Musk1 dataset most other instance space algorithms are in their 80 - 90 percent range but below 88.9 percent, besides the EMD + SVM algorithm. DD was not directly compared with other algorithms in the graph. However, 88.9 percent accuracy is higher than MI - SVM, EM - DD, SBMIL, Wrapper algorithm referenced in the graph. In the EMDD paper, a high 96 percent accuracy was achieved, which is much higher than the 67 percent in my implementation. The low accuracy could be the result of mistakes in the implementation. However, the 96 percent accuracy does not match the mid-80 percent result on Musk 1 from the Amores paper [1]. From the comparison graph between different algorithms in the paper, EMDD’s mid-80 percent accuracy is lower than many other algorithms such as EMD, Hauss, and Gartner. In general, DD and EMDD do not show noticeable advantages over other instance spaced algorithm. However, EMDD has a noticeable improvement over DD in both its accuracy and runtime speed. Comparing with bag space algorithms and embedded space algorithms, the performance of DD and EMDD are generally lower. Both EMDD and DD algorithm can be improved through a more strategic way of selecting the initial random example.

### Bag Space

For the BS paradigm, we first consider the lazy learning approach using the kNN algorithms, where training examples are simply stored for future use rather than used to construct general, explicit descriptions of the target function. Even though the kNN algorithms are relatively simple to implement, they perform well on the datasets. If a good distance measure is picked, which the min Hausdorff distance seems to be for the datasets, exploiting the information from the whole bag provides a more robust method than IS. IS is not robust because it does not use information about the bags and the performance is heavily based on the instances. Meanwhile, in BS methods, the discriminant learning process is performed in the space of bags, which makes it more robust and take into account more information about the bags while performing classification.

However, the BS algorithms do not work well without a good distance measure. This is shown in the Fox and Elephant datasets, where there are a lot of instances in each bag, as the accuracy is lower. Similarly, if the number of bags are lagging, the distances from one bag to other bags can be similar, so nearest neighbor algorithms will not perform well. In this case, one may consider using vocabulary-based algorithms instead. Adding in the weights for Weighted-kNN and Weighted-Citation-kNN does not solve this problem since the weights are dependent on the Hausdorff distance. This is similar to the results from the original paper, where the accuracy of BS algorithms on complex datasets is significantly lower than ES methods that use vocabulary for the embedding.

### Embedded Space

For the ES paradigm, we implemented several popular approaches include Distance and Histogram based Bag of Words (DBOW, HBOW), Intermediate Matching Kernel (IMK) and Simple Multiple Instance (SMI) approaches. Within each sub-paradigm of ES, we experimented with several parameters of the approaches, including varying vocabulary size, mapping functions, clustering methods, and kernels. ES algorithms attempt to extract information about the instances within each bag, which makes ES algorithms perform quite well on a variety of dataset types. The experiment results discussed in [1] show that the ES algorithms are some of the best performing algorithms across a variety of datasets.

After implementing several ES algorithms, it is clear that these algorithms are robust due to their high adaptability. For HBOW, we were able to test 5 different mapping functions, each making different assumptions about the data. HBOW can be easily modified to fit the data; for example, hard-assignment may be used when the SMI assumption is valid for the data (such as the drug activity prediction problem), whereas *plausibility* or *uncertainty,* two soft assignment mapping functions, may be used when the classification is more complex (such as image classification).

Four different clustering functions were tested with HBOW and DBOW, and we have shown that using Mean shift appears advantageous to k-means as used in [1]. Hard clustering algorithms; however, outperform soft clustering algorithms.

The ES algorithms such as HBOW or DBOW typically outperform the IS algorithms due to the flexibility that HBOW or DBOW provide (which is shown our results, as DBOW consistently outperforms all but one of the classifiers) . IS algorithms are typically bound to either the SMI or collective assumption, which is used to determine how instances contribute to the class of a bag, By mapping instances to a vector, and then using a classifier like SVM, ES algorithms are much less restricted in the decision boundaries they can use to classify bags. With the help of various distance and mapping functions, classic ES algorithms can be modified to not only represent the SMI and collective assumptions, but also many other relationships between the class of a bag and the instances held within it.

The results found in [1] support this conclusion, as IS generally underperforms on all of the data sets considered, but drastically underperform on the image classification datasets used, yield accuracies around 50 percent. Though one of the IS methods implemented (MI-SVM) performs well on *Fox* and *Elephant*, these approaches would likely be much less accurate on the image classification problems tested in [1], which have much lower dimensionality per instance but many more instances per bag when compared with *Fox* or *Elephant.* The data we generated also generally support this conclusion, with the aforementioned MI-SVM being an outlier in the IS space.

## Discussion

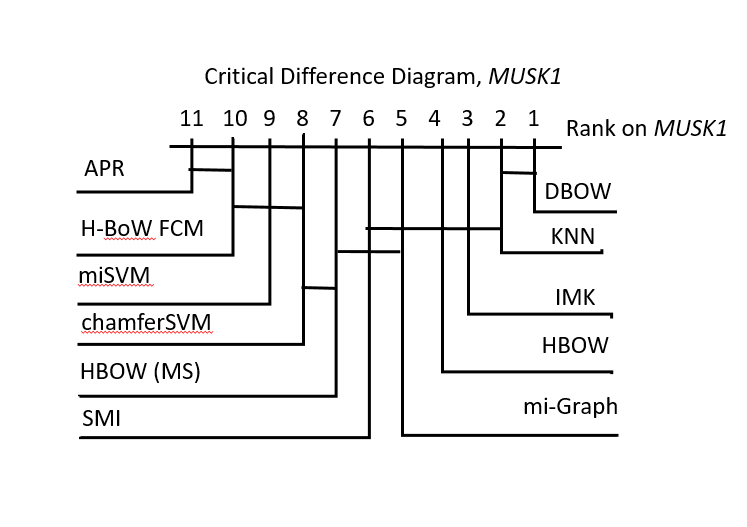


Fig 2. 95% Confidence Critical Difference Diagram for Musk1

Figure 2, above, shows the Critical Difference Diagram (CDD) for the algorithms we implemented for this project. The CDD shows which algorithms were statistically significantly different on *Musk1* with 95 percent confidence. To make this diagram, we followed the steps outlined in Lecture 18 for comparing classifiers. We established a null hypothesis that the difference in error rate, *E*, is 0, which is assumed true for all pairs of classifiers. This is formally expressed as:

For each pair of classifiers, given their accuracy and number of iterations of cross-validation performed, we found the 95 percent confidence interval for the difference in error rates using the following formula:

For all pairs of classifiers, if the lower bound of the difference of error rates is negative, we say that the null hypothesis cannot be rejected at 95% confidence, which means that the classifiers are not statistically significantly different on *Musk1.* Figure 2 shows the results of the aforementioned hypothesis testing for all of the classifiers that we implemented. The bars linking classifiers in Figure 2 show the pairs or groups of classifiers for which the null hypothesis was unable to be rejected at 95% confidence. Figure 2 also shows the rankings of each classifier on *Musk1*, which can be read from right to left.

Overall, we have shown that the Instance Space methods are the poorest performers across all three data sets. The Bag Space and Embedded space methods perform similarly with few outliers.

## References

[1] J. Amores, “Multiple instance classification: Review, taxonomy and comparative study,” *Artificial Intelligence*, vol. 201, 2013, pp. 81–105.