ABSTRACT

Title of dissertation: Semi-supervised and Active Image Clustering with Pairwise Constraints from Humans

Arijit Biswas, Doctor of Philosophy, 2014

Dissertation directed by: Prof. David W. Jacobs
Department of Computer Science
University of Maryland, College Park

Clustering images has been an interesting problem for computer vision and machine learning researchers for many years. However as the number of categories increases, image clustering becomes extremely hard and is not possible to use for many practical applications. Researchers have proposed several methods that use semi-supervision from humans to improve clustering. Constrained clustering, where users indicate whether an image pair belong to the same category or not, is a well-known paradigm for semi-supervision. Past research has shown that pairwise constraints have the potential to significantly improve clustering performance.

There are two major components to constrained clustering research: how pairwise constraints can be used to improve clustering (e.g: constrained clustering algorithms, dis-

tance or metric learning methods) and determining which constraints are most useful for improving clustering (e.g.: active or interactive clustering methods). In this thesis we propose three different approaches to improve pairwise constrained clustering spanning both of these components. First, we propose a distance learning method in non-vector spaces, where the triangle inequality is used to propagate the pairwise constraints to the unsupervised image pairs. This approach can work with any pairwise distance and does not require any vector representation of images. Second, we propose an algorithm for active image pair selection. A novel method is developed to choose the most useful pairs to show a person, obtaining constraints that improve clustering. Third, we study how pairwise constraints can effectively be used to cluster large image datasets. Complete clustering of large datasets requires an extremely large number of pairwise constraints and may not be feasible in practice. We propose a new algorithm to cluster a subset of the images only (we call this subclustering), which will produce a few examples from each class. Subclustering will produce smaller but purer clusters and can be used for summarization, category discovery, browsing, image search, etc.... Finally, we make use of human input in an active subclustering algorithm to further improve results. We perform experiments on several real world datasets such as faces, leaves, videos and scenes and empirically show that our approaches can advance the state-of-the-art in clustering.

Semi-supervised and Active Image Clustering with Pairwise Constraints from Humans

by

Arijit Biswas

Dissertation submitted to the Faculty of the Graduate School of the
University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2014

Advisory Committee:
Prof. David W. Jacobs, Chair/Advisor
Prof. Larry S. Davis
Prof. Hal Daumé III
Prof. Ramani Duraiswami
Prof. Rama Chellappa, Dean's Representative

Dedication

I dedicate this thesis to my parents.

Acknowledgments

First, I would like to thank my advisor Dr. David Jacobs for the strong support and freedom he has provided me throughout my PhD life. The lively and involved meetings with David are still the most cherished hours of the last five years. Whenever I lost direction in my research, he would guide me until I had new ideas to pursue. I strongly appreciate his precious advice in the initial years of my PhD life while I was still an amateur in computer vision and machine learning. He has taught me how to find interesting problems, study them and then figure out the best possible way to solve them. His unique style of research has always encouraged me and will continue to do so until the end of my research career.

I would also like to thank my committee members Larry Davis, Rama Chellappa, Hal Daume and Ramani Duraiswami. They have provided many useful comments during my proposal and defense which helped me to significantly improve my PhD thesis. I am also very much grateful to the professors at the Computer Science Department who have taught excellent courses and helped me to develop my background in Computer Science.

I would like to thank my internship supervisors Dr. Devi Parikh at Virginia Tech (previously at Toyota Technological Institute at Chicago (TTIC)) and Dr. Eugene Bart at Xerox Palo Alto Research Center (PARC) who guided me to understand and work on several practical aspects of computer vision research.

I am thankful to many CS and UMIACS personnel without whom my research and official work could have stopped any time. I would like to make special mention of Jennifer, Fatima, Janice, Adelaide, Arlene and Jonathan.

My PhD studies would not have been so much fun without the friends I had during my stay at UMD. First and foremost, I would like to thank my academic siblings Carlos, Anne, Joao, Daozheng, Abhishek, Angjoo and Jin for the interesting discussions and feedback they have provided about my research during the last five years. I should also thank other members of the Computer Vision lab who have enriched my research experience while I was at UMD: Sumit, Ashish, Raviteja, Austin, Chinglik, Ajay, Kaushik, Nazre, Sima, Jayshankar, Ming-Yu, Mohammed, Sameh, Jonghyan, Vishal and everyone else. I am also grateful to my past and present roommates Ranchu, Biswadip, Siddharth, Anup, Ranjith who have always agreed to cook good food when I was just beginning to attend Cooking 101. I must mention many other friends in UMD without whom my journey would have been incomplete: Ashwin, Vaibhav, Krishna, Soumyadeep, Soham, Shawon, Sofia, Titir, Arti, Anand, Jonathan, Pritam and Wrick. I am also grateful to my friends from high school and Jadavpur University who have always been supportive throughout my life.

This dissertation would not have been possible without constant support of my parents. Words would not be enough to express my gratitude to them. This dissertation is a testament to their commitment towards my education and sacrifices they have made in shaping my life. I dedicate this dissertation to them.

Table of Contents

List of Figures

Chapter 1: **Introduction**

Clustering, or *unsupervised learning*, is a critical part of the analysis of data. There has been a huge volume of past work on clustering [40], producing many interesting and effective algorithms. Clustering images has also been a problem of interest, where a set of unlabeled images are automatically grouped based on identity or a common characteristic. We discuss a few possible applications of image clustering below:

- **Labeling:** Clustering images can reduce the labeling cost significantly. If we have a set of unlabeled images that are required to be labeled, we can cluster them and then someone can label each cluster just by looking at a few images from that cluster. This could significantly reduce the labeling cost compared to an exhaustive method where a person will have to go through each image individually for labeling.

- **Video Surveillance:** Nowadays many places such as airports, parking lots, shopping centers etc... are kept under constant surveillance. Cameras usually store very large amounts of video data every day. It is often important to locate a person or an action of interest by looking at those videos. It is not possible for someone to go through all of the video frames to find these actions or persons of interest. However we can cluster frames based on the actions or the persons present in them and

then one can look at a few of these frames in each cluster to detect anomalies in those video streams. This can reduce the time required for humans by a significant amount.

- **Category Discovery:** Computer vision algorithms have recently been applied to many species identification tasks, such as identifying leaves [48], birds [12, 27] etc.... Users often upload images without any labels to these systems. It is extremely important to know if any of their species are unknown to the community or if they are found in a new location. This information could potentially increase our existing knowledge about biodiversity. We also note that a biologist or an enthusiast might not have time to go through the list of millions of images to find such an exception. If we cluster the unlabeled images it becomes significantly easier for someone to browse through a few images from each cluster to discover an unknown category.

Although getting reasonable image clustering results with a low number of clusters is possible, with an increase in the number of clusters, performance drops significantly. With large numbers of clusters, images from different categories are more likely to overlap in the feature space making clustering an extremely difficult task. Semi-supervised clustering [4] is a possible way to improve clustering performance by incorporating human supervision. Human supervision can take many possible forms. Pairwise constrained clustering is a well-known form of supervision in which humans indicate whether an image pair belong to the same category (must-link constraints) or not (can't-link constraints). These constraints can be used to improve clustering performance.

Pairwise constraints are useful because often it is difficult for people to assign a label to an image (they may not be able to assign a name to a face in a surveillance video, or to a particular plant specimen), but much easier to assess whether two images come from the same class (humans can compare face images with 99% accuracy [49]). For example, while clustering images of biological organisms such as leaves or birds, a person without previous knowledge might find it difficult to label an image. But she might be able to say if two leaf/bird images belong to the same species or not. In this thesis we focus our attention on developing methods to improve pairwise constrained clustering.

As an example of a semi-supervised problem, suppose we have access to surveillance video data from an airport and we would like to group the face images based on identity. This makes it easier to scan the people who have been present in the airport. Automatic clustering of these images is extremely hard because of the large number of clusters and large variation in pose, expression and lighting present in these faces. However we know that faces that are tracked through the video should be faces of the same person, giving rise to many useful must-link constraints. Also, if two face tracks appear together in some frames, then faces in these two tracks must be of different persons, even if they look very similar, providing several can't-link constraints. These constraints can be used to improve face clustering in videos [20, 73, 76]. Semi-supervision also arises when people have labeled a subset of images, in which case these labels can be converted to must-link or can't-link constraints.

While we use pairwise constraints to improve clustering we should look into the two major components of constrained clustering:

- **How do we improve clustering with pairwise constraints?**

  Many past research works have studied how pairwise must-link and can't-link constraints can be used to improve clustering. There are two possible ways to do that. First, cluster the elements such that the specified constraints are satisfied as much as possible. There are several past works [6, 7, 74] in this area. Second, learn better distances between images using the pairwise constraints and use the new set of distances to cluster images. A recent survey [10] describes most of the important works in this area. However all of the past distance/metric learning works assume that an Euclidean distance is computed between images in a vector space and requires vector representation of all the images.

- **Which pairwise constraints are most useful to improve clustering?**

  If there are a total of N images to be clustered, it is possible to obtain $O(N^2)$ pairwise constraints. However all of these constraints are not equally important. If two images are very close in the feature space it is not very useful to know that they are must-linked; this constraint is unlikely to improve clustering. Similarly if two points are very far from each other in the feature space, a can't-link between them will not improve clustering. Also obtaining all of the $O(N^2)$ pairwise constraints is expensive and not feasible in practice. So we are required to figure out which image pairs if constrained will provide us maximum information for better clustering and try to get as many of those as possible.

In the pairwise constrained clustering setup we solve three related problems that can improve image clustering. In all these problems we assume that we know the number

of clusters beforehand. The first problem is an approach for improving clustering using the must-link/can't-link constraints. The next two problems address how we can obtain useful pairwise constraints from humans to improve clustering of medium and large sized datasets. We briefly discuss all of the problems here and then we discuss each of these problems in detail in Sections 1.1-1.3.

1. We propose a method for learning better image distances from pairwise constraints and use these distances to cluster images. When we obtain a set of pairwise constraints it is important to propagate them to the unlabeled image pairs. Our approach uses the triangle inequality constraints to propagate constraints to the pairs for which no supervision is available. The proposed approach is a general method for distance learning that can work with any kinds of pairwise distances and does not require any vector representation of images.

2. We propose an active method for obtaining pairwise constraints from humans. This method chooses the image pair for which the overall expected change in clustering is maximized. We try to obtain complementary information from humans that we do not already have from the algorithms. Our approach produces state-of-the-art active clustering results in three different domains.

3. Finally, we propose a method for clustering a subset of the data (we call this subclustering) only, and see how pairwise constraints can help in subclustering. Subclustering is a novel problem and can help in browsing, summarization, category discovery etc... if the database size is very large (size 50,000 or more). We first propose a passive method for subclustering which does not require any human in-

tervention. Then we propose an active algorithm to choose image pairs to improve the passive subclustering solution.

In Figure 1, we summarize the problems that we solve in this thesis and how they are connected to the constrained clustering research area.

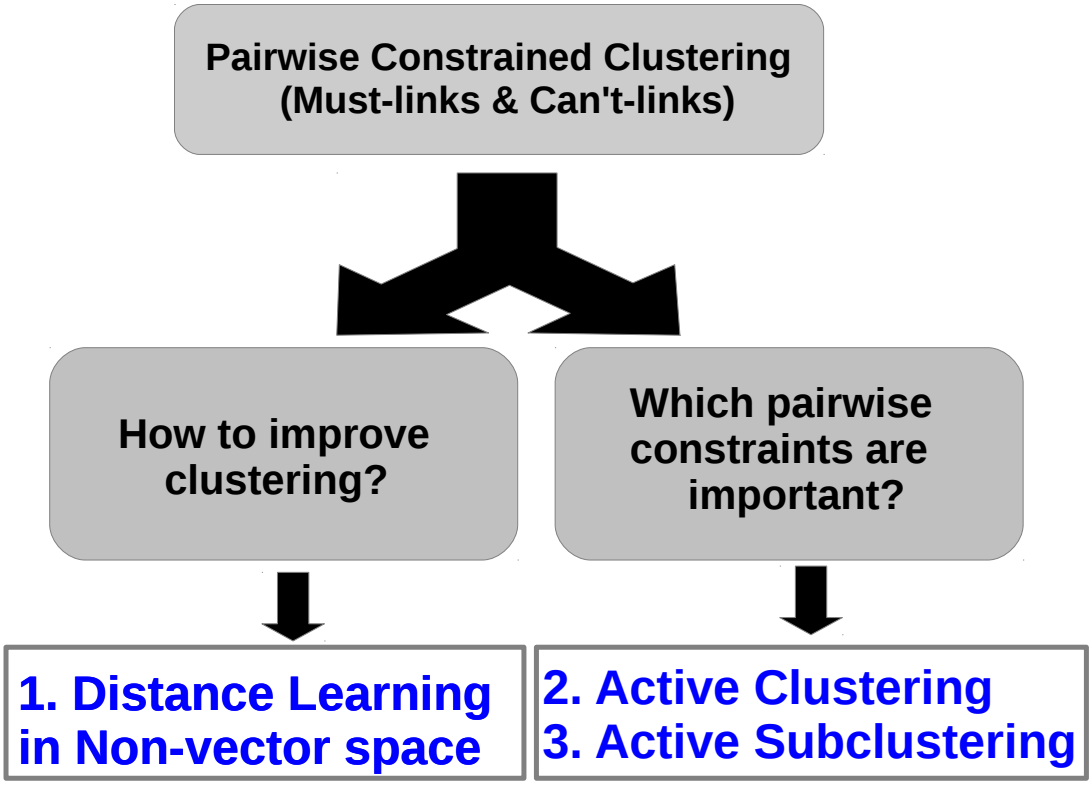


Figure 1: A diagram that motivates how the problems that we solve are related to pairwise constrained clustering.

We perform experiments on clustering face images, leaf images and scene images. We also perform some experiments on clustering face images extracted from videos. We discuss the experimental details in the relevant chapters. Some example face images and

leaf images are shown in Figures 2 and 3 respectively.

1.1 Distance Learning in Non-vector Spaces

First, we develop a distance learning method for learning better image distances from pairwise constraints. There are several previous metric learning approaches that can use pairwise constraints to modify distances for unsupervised pairs, when the image distances can be represented as Euclidean distances in vector spaces [24, 77]. However, in many cases, distances between images are not computed this way. Many authors have proposed manifold based distances [29, 55, 64], in which a geodesic distance between two images is computed on a manifold. Robust distance measures [25, 39] are also used, where large differences in feature values are not taken into account while computing the pairwise distance between images. Often a classifier is trained to tell whether two images belong to the same class, and the classifier output [9] is converted to an image distance. In shape analysis, distances between shapes are often computed using combinatorial approaches such as bipartite matching [11, 52]. We are particularly interested in semi-supervised clustering of fine-grained categories, as these have many meaningful categories that are challenging to distinguish without supervision. If we look at the top 10 distance measures in each of two important domains (LFW faces [1] and leaf shapes [2]) related to fine-grained classification, we find that 80% of the methods use non-vector space distances. These distances can be used for clustering images, but are not suitable for existing metric learning algorithms. It is also not clear how one can develop a parametric method for distance learning that will work effectively with all of the above-mentioned distances.

We note that in order to propagate constraints from supervised to unsupervised pairs, some structure must be assumed on the set of possible distances. Otherwise, the distance between supervised pairs could be altered without affecting the distance between unsupervised pairs. Perhaps the weakest assumption that we can make about a distance is that it obeys the triangle inequality which states that if a,b,c are three points and $d(.,.)$ is a distance function, then $\forall$ a,b,c, $d(a,b) + d(b,c) \geq d(a,c)$. Enforcing the triangle inequality allows us to propagate constraints; if a constraint alters one distance, other distances must also change to maintain triangle inequalities. In Section $2.2.1$ (Chapter 2) we use toy examples to explain how triangle inequality constraints effectively propagate pairwise constraints to unlabeled image pairs.

The triangle inequality always holds for metric distances, even if the distance is computed in a non-vector space (e.g., manifold distances). For many other interesting distances, the triangle inequality is not guaranteed to hold. However, we empirically find the triangle inequality almost always holds for distances computed for fine-grained classification even when not explicitly enforced. For example, with the distances we use in Pubfig [9, 49] faces and Leafsnap [47], 99.98% and 97.48% of the triangle inequality constraints hold respectively. This strongly motivates us to enforce the triangle inequalities when we alter distances to incorporate the pairwise constraints. We then find empirically that by enforcing the triangle inequality we can improve performance on several real-world datasets.

Our main contribution is a novel method for using the triangle inequality to learn image distances from pairwise constraints. We use these distances along with a constrained clustering algorithm [74] to achieve state-of-the-art clustering results. The pro-

posed approach is a general framework to learn distances using any kind of pairwise distances between all the images to be clustered and does not require any vector space representation of images.

Our approach formulates a quadratic optimization problem, where the pairwise distances between images are modified such that pairwise constraints and triangle inequality constraints are satisfied as much as possible. Since enforcing $O(N^3)$ triangle inequalities is computationally expensive, we propose a graph based approach, where only $O(n(M + C))$ triangle inequalities are sampled for use in the QP (N is the total number of images, n is the number of edges in the n-nearest neighbor graph, M and C are the number of must-link and can't-link constraints respectively). We empirically show that this sampling approach works well in practice.

We theoretically analyze a simplified case in which only one constraint is present, to gain insights into our fast approach. Our sampling approach is based on the intuition that clustering is predominately affected by small distances, and is not sensitive to the exact value of larger distances. We prove that our sampling approach produces the same set of small distances that would be obtained by enforcing all constraints.

We perform experiments on leaf and face image datasets and show that distances obtained by our method achieve state-of-the-art clustering results. We also run experiments on a real world video dataset, where extracted faces are clustered based on identity. Our approach outperforms recent constrained clustering methods on this video dataset.

1.2 Active Clustering

In this second work we propose an active method for image pair selection to get constraints from humans. Our goal is to meld human and automatic algorithms by directing valuable human attention to those judgments that are most critical to improving clustering produced by automatic means.

We may then summarize the active clustering setup which we work on as having the following characteristics:

- We begin with a collection of objects that can be grouped into a set of disjoint clusters.
- We have an automatic algorithm that can give us some useful information about the similarity of two objects.
- A person can make these judgments with much greater accuracy than existing algorithms.

Given this problem formulation, we have proposed an algorithm that does the following:

1. Cluster objects into groups, combining automatically computed distances with any constraints provided by people.
2. Choose a useful question to ask a person. The person will compare two objects and indicate whether they belong to the same group (or answer "don't know").
3. Repeat, using the information provided by the person as a new constraint.
4. Continue asking questions until a reasonable clustering is achieved or the human

budget is exhausted.

We show the pipeline of our algorithm in Figure 4 (face images from [58]).

We perform experiments to evaluate our algorithm in three different domains: face, leaf and scene images. We demonstrate that our novel approach is much better than the state of the art algorithms. Since we assume that people are highly accurate, in experiments we can simulate their behavior using ground truth. We also relax the assumption that humans are always correct and perform some experiments with real humans in the loop using Mechanical Turk. We find that our algorithm still produces useful results and is better than the state-of-the-art approaches.

1.3 Active Subclustering

Humans often search for images in large databases. Let us take the example of video surveillance in which a user might be looking to see whether a person of interest was present in the airport on a particular day. If we can create an organized collection of face images containing a few different images of each person present in the airport, it will be significantly easier for an analyst to browse them to detect the presence of that person of interest. Similarly for a category discovery application it is useful to create a leaf/bird image collection from the unlabeled images that only contains a few images from each class present in the data, allowing easy browsing.

In both examples we start with a large collection of unlabeled images. These images can be clustered using traditional clustering algorithms [40, 44, 56, 66] to facilitate browsing, allowing a user to examine meaningful subsets of images. As we pointed out

earlier, accurate clustering is extremely difficult when there are a large number of clusters. To cope with this challenge, we introduce the idea of *subclustering*, in which we cluster only a small subset of images. Our goal is to create clusters that represent each object in the image set with a few images. For example, in the surveillance scenario, subclustering aims to create clusters containing a few images of each individual. Subclustering has the potential to produce more efficient algorithms, and to produce more accurate clusters by ignoring images that are hard to categorize. We also assume that a modest amount of labeling effort can be obtained from humans to improve subclustering. We again abstract human input as providing must-link and can't-link constraints[1].

Our two main contributions in subclustering are:

1. We propose a new passive subclustering algorithm where no human input is required. We propose a cost function suitable for subclustering and optimize it with an algorithm motivated by the Partition Around Medoids (PAM) [44] algorithm. In experiments we show that our approach produces much better subclusters than those produced by several baseline algorithms. Our subclustering algorithm reduces run-time by $O(N)$ compared to a similar complete clustering algorithm like PAM, where N is the total number of images in the dataset.
2. We also propose a new active subclustering algorithm that uses pairwise constraints on the clusters. The algorithm repeatedly selects an image pair and asks a user whether they belong to the same category or not. Pairs of images are chosen such that existing errors in a subclustering are corrected with limited human interaction.

[1]We will use classes/categories to imply the ground truth and clusters/subclusters to imply algorithms' output throughout this thesis.

In experiments we show that we can accurately subcluster a face image dataset with $51,418$ images of 200 classes with around $143,000$ pairwise constraints from humans (equivalent Mechanical Turk cost: \$358). In general active algorithms for complete clustering require much more human supervision [6, 78] and are often too slow to run on large scale datasets [14].

We run experiments in two different domains: face images (the complete Pubfig dataset [49]) and leaf images [47]. We show that our proposed algorithms perform much better than baseline algorithms. A pipeline of our proposed system for subclustering is depicted in Figure 5.

1.3.1 Why Subclustering?

Over the last decade active/semi-supervised clustering algorithms [6, 14, 78], including our own work, have produced much better clustering results. However, as the data size (number of images and classes) increases, the required number of pairwise constraints increases so much that human-in-the-loop clustering algorithms become impractical. Therefore we introduce subclustering that aims to produce a clustering that contains n images from each category with $n < N_k$ (where N_k is the actual number of instances of class k present in the full dataset). This will produce smaller but purer clusters and will require significantly less human input. Since large unlabeled image collections are highly prevalent these days subclustering can have a wide range of applications including browsing image databases, image search, summarizing image databases, category discovery etc....

Input to a subclustering algorithm will include a pairwise distance matrix, the number of clusters and the number of images required ($n \geq 1$) from each class. We emphasize that a subclustering algorithm's goal is to extract any n images from each class. For large real world applications, where each class is expected to have many (more than 100) images, we find that $n = 10$ is reasonable. It is not too large that subclustering becomes error-prone and also not too small that images from the same class do not show variation. However in Section $4.6.2$ (Chapter 4), we perform experiments to see how robust our approach is to the subcluster size. In Figure 6 we show some typical subclusters produced after using our passive and active algorithms. Although the number of examples from each class is small, they have enough intra-class variation that humans can determine how these people look and compare them to a new face image. We note that subclusters are not intended for training classifiers because they are small and may not represent the complete distribution of each class. Subclustering and its use for browsing large image databases is partially motivated by the fact that humans can learn reasonable object models even from a small set of examples [51].

Although it is possible to create subclusters from complete clusters, algorithms solely built to solve subclustering create purer subclusters. Intuitively it is clear that points that are difficult to cluster can affect the results of clustering algorithms in a way that degrades the clustering of easier points.

1.4 Conclusion

Although image clustering has a wide range of applications, clustering is an extremely hard problem especially when the number of clusters is high. Pairwise constraints from humans can help image clustering significantly. In this thesis we have proposed three methods to improve the performance of pairwise constrained clustering. The first approach uses the constraints to learn better image distances for image pairs. Next two methods were proposed to improve clustering by obtaining useful pairwise constraints from humans for medium and large scale datasets. We performed experiments on real world datasets of faces, leaves, videos and scenes and showed that our approaches can improve state-of-the-art image clustering by significant margins.

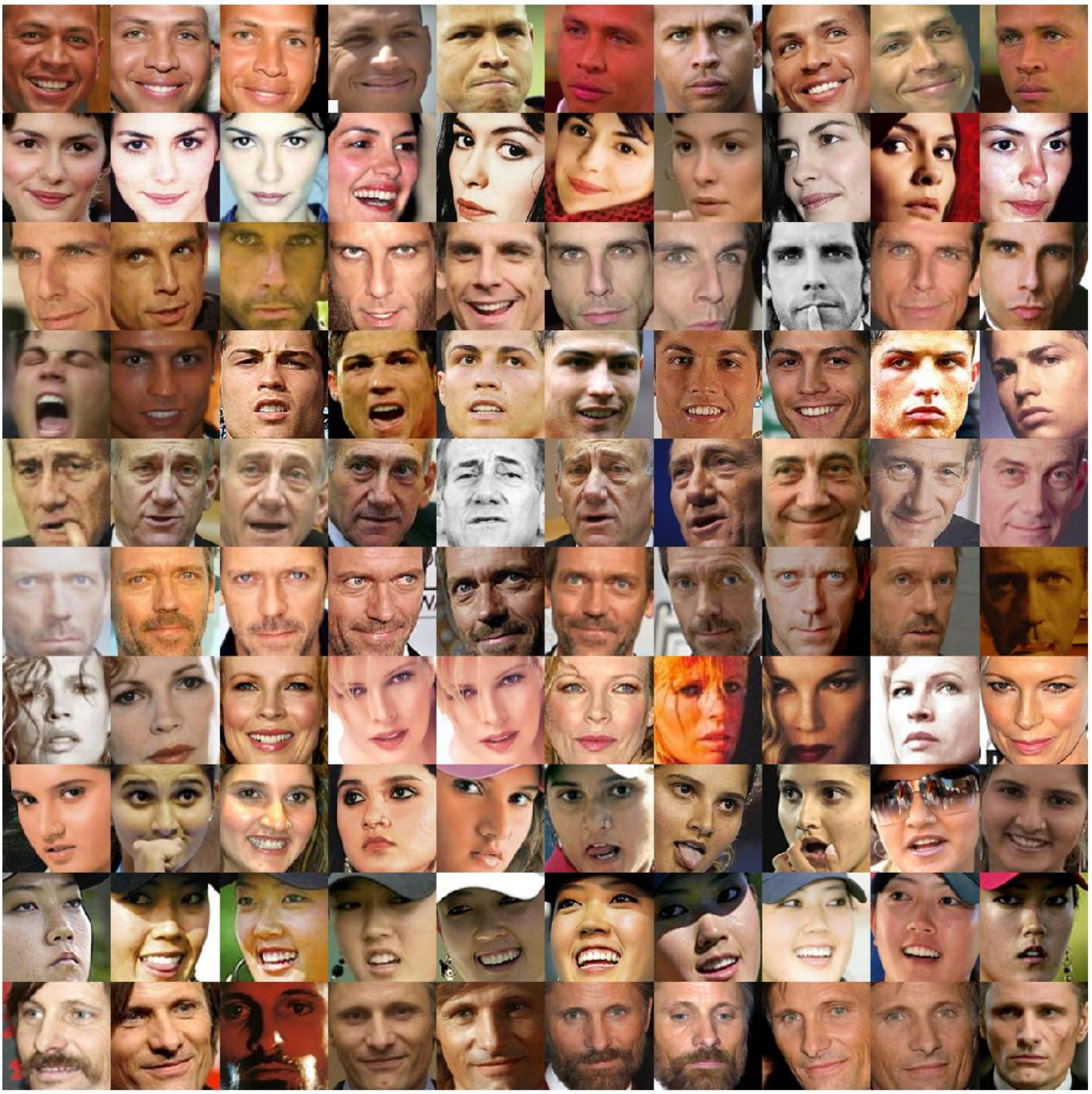


Figure 2: We show some example face images from the dataset Pubfig[49]. Each row shows some example face images of a single person. Variation in lighting, expression and pose is so high that automatic clustering algorithms fail badly, especially with a large number of clusters.

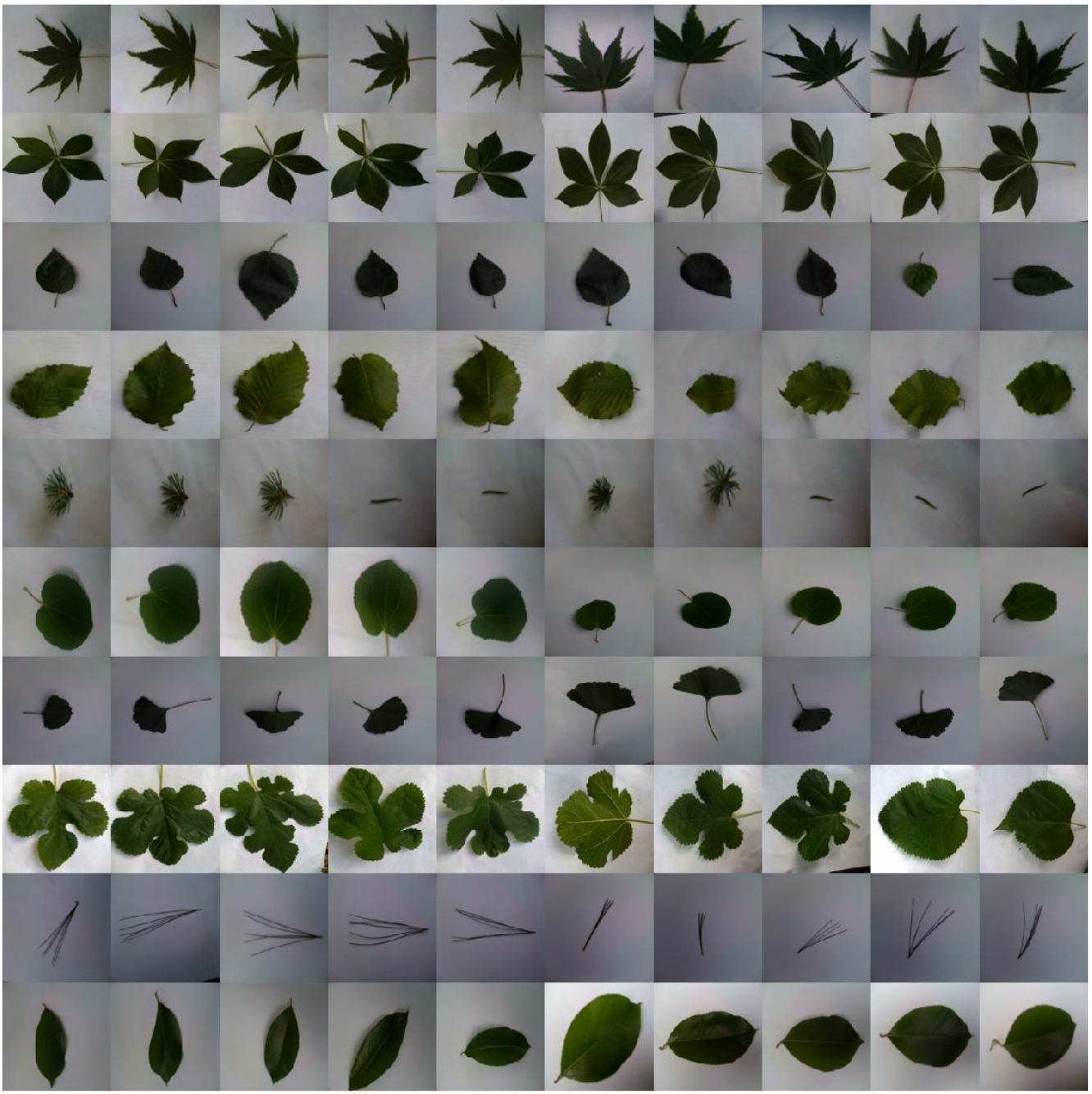


Figure 3: We show some example leaf images from the dataset used in Leafsnap[47]. Each row shows some example leaf images of a particular plant species. Due to the presence of specularity and shadow automatic clustering algorithms often fail badly, especially with a large number of clusters.

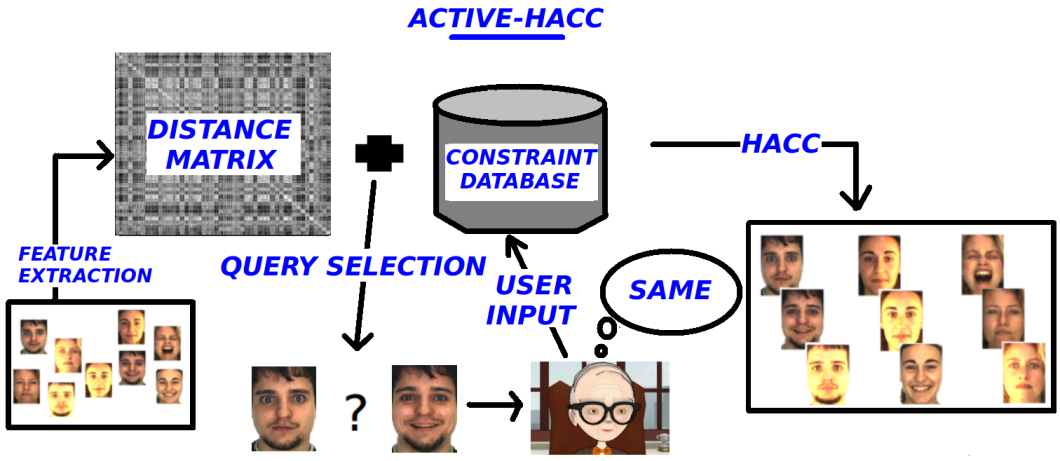


Figure 4: Pipeline of our system: Active-HACC is our proposed active algorithm for selecting data-pairs to get constraints and HACC is a constrained clustering algorithm.

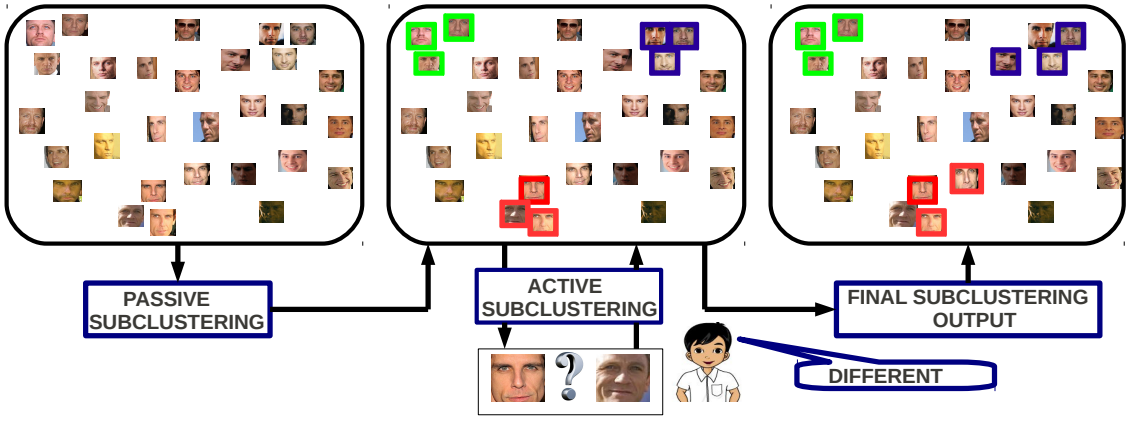


Figure 5: The pipeline of our proposed subclustering algorithms. First we start with an unlabeled set of images. Next, we apply our passive subclustering algorithm. Green, red and blue boxes are used to denote three different subclusters of size three. Then our active algorithm chooses pairs of images to show to a user to obtain constraints; we update the subclustering with those constraints. At any point, we can produce a subclustering of the image data that users can browse easily.

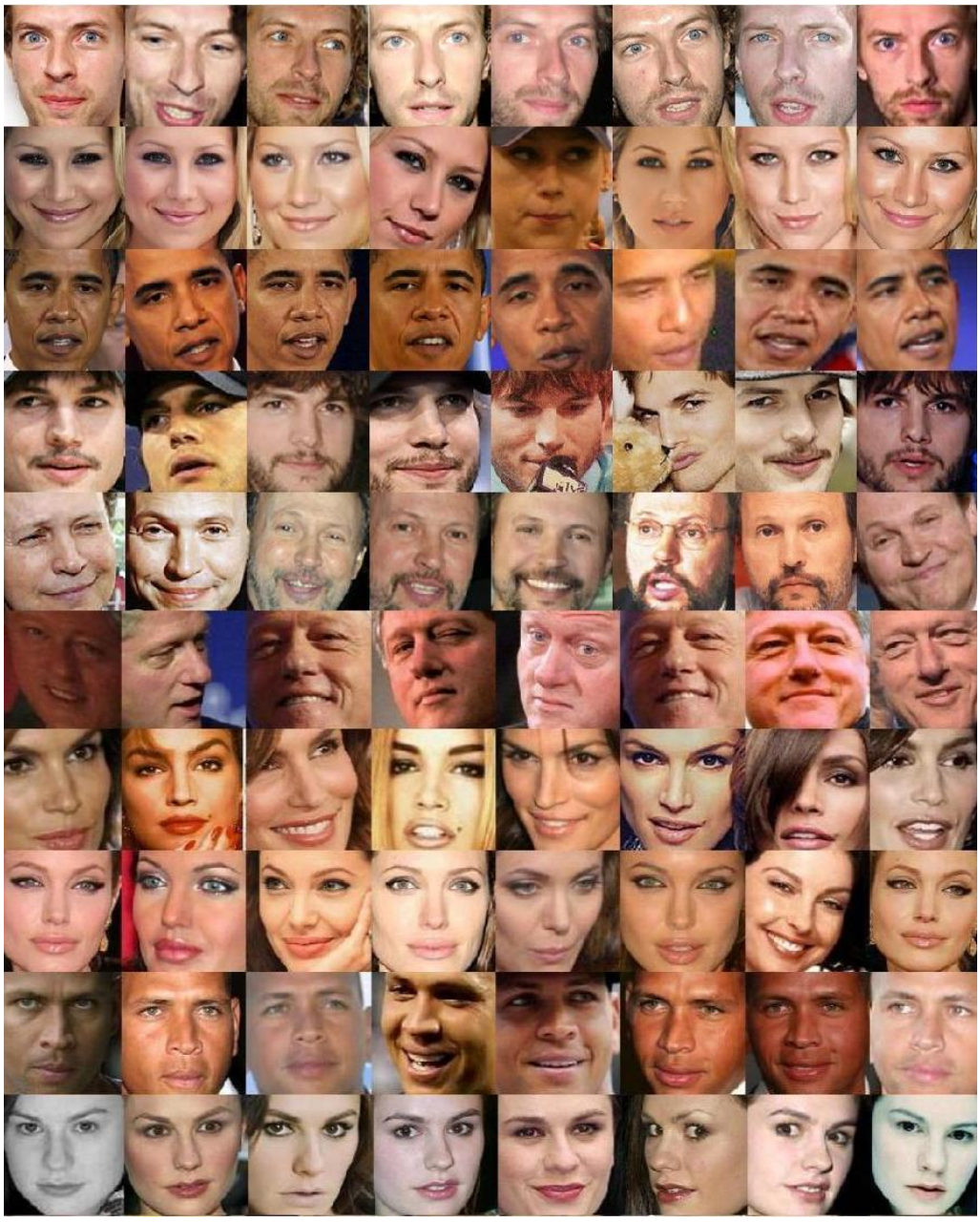


Figure 6: 10 example subclusters produced after we run our proposed algorithms. Each row contains one person's faces.

Chapter 2: **Distance Learning in Non-Vector Spaces**

In this chapter, we propose an approach to use the triangle inequality to propagate pairwise constraints to the unsupervised data and learn a new set of distances. We assume that no vector space representation of images is available. The learned distances will be used for clustering where all the training data and pairwise constraints are available ahead of time. We note that we are not considering an out-of-sample extension in this work.

First, we formulate a brute-force Quadratic Programming (QP) method that modifies the distances such that the total change in distances is minimized but the final distances obey the triangle inequality. Then we propose a much faster version of the QP that enforces only a subset of the inequalities. We show experimentally that this faster QP produces stronger clustering results on datasets of face, leaf and video images, outperforming state-of-the-art methods for constrained clustering. To gain insight into the effectiveness of this algorithm, we analyze a special case of the semi-supervised clustering problem, and show that the subset of constraints that we sample still preserves key properties of the distances that would be produced by enforcing all constraints.

2.1 Related Work

Our work draws on prior work in constrained clustering and metric learning. One of the earliest constrained clustering algorithms was proposed by Wagstaff et al [74] in 2001. The authors in [74] proposed an algorithm in which points are assigned to clusters such that none of the specified constraints are violated. The authors in [7] proposed a method in which a Hidden Markov Random Field (HMRF) is formulated along with must-link and can't-link constraints and MAP estimation is performed on the HMRF. The authors in [6] proposed a soft constrained clustering algorithm, where the cost function for clustering takes care of the pairwise constraints. However this approach is more suited for the active constrained clustering setup proposed in [6], where constraints are obtained in an interactive manner. The authors in [53, 54] proposed spectral clustering methods where pairwise constraints are propagated to the entire dataset. However spectral methods usually require vector representation of the images for clustering. In [45] the authors proposed a method for constrained clustering where the must-links are propagated between other image pairs, by computing the shortest path between all pairs. A straight-forward modification to the Floyd-Warshall algorithm [21] is used to perform the fast all-pair-shortest-path computation.

There is a significant amount of past research work on metric learning. A recent survey [10] nicely describes the past work in this area. We discuss a couple of well known methods for metric learning. The authors in [77] proposed one such method where a Mahalanobis Distance metric was learned, such that must-linked image pairs come close to each other and can't-link pairs move away from each other. However as pointed out by

[24], this method can be slow and inaccurate. The authors in [24] proposed a widely used metric learning approach called Information Theoretic Metric Learning (ITML), which works well in practice; both these methods require a vector representation of images. Perhaps the most similar past work is on non-parametric kernel learning methods [35, 82], where a kernel matrix is learned from the similarity matrix and user provided constraints by using semi-definite programming. However we demonstrate in the experimental results that such approaches do not work well for image datasets, especially when there are many clusters.

Recently several researchers have applied constrained clustering techniques for clustering faces in videos. In [20], the authors proposed a method for metric learning from constraints and used that for face identification. The authors in [73] also proposed an approach to cluster face images with pairwise constraints for movie content analysis. In the most recent work on face clustering [76], a method was proposed where a Hidden Markov Random Field based objective function is formulated that incorporates the pairwise constraints; it is optimized using the simulated field algorithm [18].

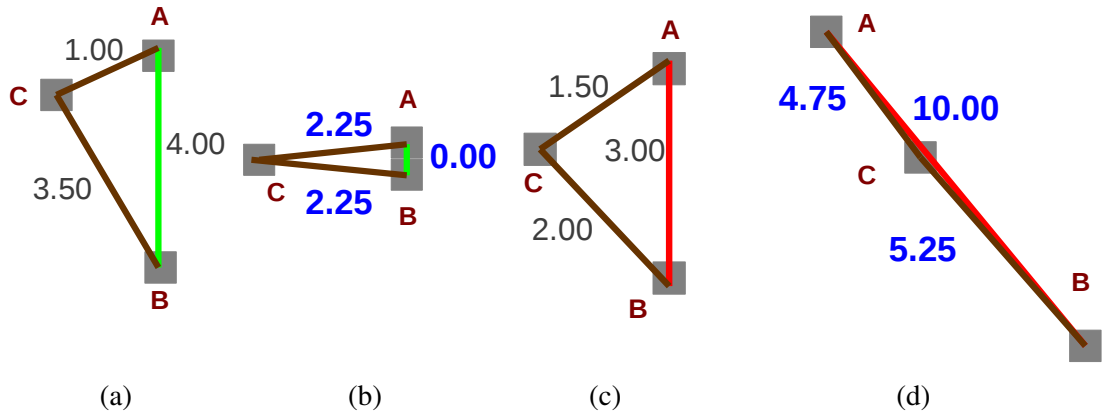
2.2 Our Approach

Semi-supervised clustering with pairwise constraints is most useful when the constraints are propagated to other image pairs as well. In this chapter we propose an approach for learning new distances using only the initial pairwise distances between images. We formulate a quadratic program to modify the distances such that total distance modification is minimized but the final distances obey the triangle inequality. We define the ideal

scenario QP in Section 2.2.3 in which all constraints are enforced and discuss the high computation cost for solving the ideal case. Next, in Section 2.2.4 we propose a graph based QP formulation which is much faster and applicable to larger datasets in practice. In Section 2.2.5, we theoretically demonstrate why the graph based approach is a good alternative to the brute-force method.

2.2.1 High Level Intuition

In this subsection, we describe toy examples to explain how our approach uses the triangle inequality to propagate pairwise constraints. In Figure 1a, we have 3 two dimensional points A, B and C and a must-link between A and B. Since there is a must-link constraint present between points A and B, the distance (referred to using the function $d(\cdot, \cdot)$ from now on) between A and B is set to zero. Now intuitively it is clear that C should become equidistant to both A and B. Similarly, in Figure 1c, a can't-link between A and B alters $d(A, B)$ to a very high value (say, 10). In this scenario C cannot stay close to both A and B and should move away from both or at least one of them. We find that by enforcing triangle inequality constraints while changing the distances as little as possible, we can set $d(A, C)$ equal to $d(B, C)$ (in Figure 1b) and move C far from both A and B (in Figure 1d). These examples show how we can learn better distances by enforcing triangle inequality constraints.



(a) (b) (c) (d)

Figure 1: Toy examples to motivate our distance learning approach. We have three two-dimensional points A, B and C with a must-link/can't-link between A and B. Initial pairwise distances are written in black. (a) Must-link constraint (shown in green). (b) Distances learned using our approach (in blue). (c) Can't-link constraint (shown in red). (d) Distances learned using our approach (written in blue).

2.2.2 Notation

We begin with a set of N unlabeled images $\mathcal{U}$ ($\boldsymbol{x} \in \mathcal{U}$) from K classes. We are also provided with initial distances between all the image pairs, i.e., $d_I(\boldsymbol{x}_i, \boldsymbol{x}_j)$ is given $\forall i, j$ (note that d_I denotes initial distance). Let $\mathcal{M}$ denote the set of must-link constraints such that any pair $(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{M}$ implies that $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong to the same class. Similarly $\mathcal{C}$ denotes the set of can't-link constraints such that any pair $(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{C}$ implies that $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ belong to different classes.

2.2.3 Problem Formulation

We learn a new set of distances for all the image pairs given the pairwise constraints $\mathcal{M}$ and $\mathcal{C}$. A quadratic optimization problem is formulated to find these distances. Let us assume that the set of final distances are given by $d_F(\boldsymbol{x}_i, \boldsymbol{x}_j)$, which we can obtain by solving the following quadratic optimization problem:

$$
\begin{aligned}
\underset{d_F}{\text{minimize}} \quad & \sum_{(\boldsymbol{x}_i,\boldsymbol{x}_j)\notin\mathcal{M}\cup\mathcal{C}} (d_F(\boldsymbol{x}_i,\boldsymbol{x}_j) - d_I(\boldsymbol{x}_i,\boldsymbol{x}_j))^2 \\
\text{subject to} \quad & (i)\ \ d_F(\boldsymbol{x}_i,\boldsymbol{x}_j) \leq U,\ \ (\boldsymbol{x}_i,\boldsymbol{x}_j)\in\mathcal{M} \\
& (ii)\ \ d_F(\boldsymbol{x}_i,\boldsymbol{x}_j) \geq L,\ \ (\boldsymbol{x}_i,\boldsymbol{x}_j)\in\mathcal{C} \\
& (iii)\ \ d_F(\boldsymbol{x}_i,\boldsymbol{x}_j) + d_F(\boldsymbol{x}_j,\boldsymbol{x}_k) \geq d_F(\boldsymbol{x}_i,\boldsymbol{x}_k), \\
& \qquad \forall i,j,k \\
& (iv)\ \ d_F(\boldsymbol{x}_i,\boldsymbol{x}_j) \geq 0, \forall i,j
\end{aligned}
\tag{2.1}
$$

In the formulation in equation 2.1, distances between all image pairs are modified such that the constraints (i) to (iv) are satisfied. The objective function minimizes the total sum of changes in pairwise distances. Constraints in (i) are added such that distances corresponding to the must-link image pairs are minimized as much as possible, i.e., are upper bounded by a small user defined constant U. Similarly, constraints in (ii) are added such that can't-link image pairs move as far as possible, i.e., are lower bounded by a large user defined constant L. One of the major contributions of our approach is to add the triangle inequality constraints in (iii), which enforce that the final distances $(d_F(\boldsymbol{x}_i,\boldsymbol{x}_j))$ should obey the triangle inequality for all possible triplets of images. These triangle inequality constraints propagate the information about must-link and can't-link constraints to other image pairs. The final set of constraints in (iv) ensure that all the quadratic optimization variables, i.e., the pairwise distances remain non-negative.

In the QP formulation as stated in 2.1, there are $O(N^2)$ QP variables and $O(N^3)$ triangle inequality constraints. Although this brute force approach can be applied to small problems with 100 or fewer images, as the problem size grows larger the QP becomes

extremely large and it is not possible to solve in practice. Also with all the pairwise and triangle inequality constraints there may not exist a feasible solution. To avoid these issues we reduce the size of the QP significantly by determining which triangle inequalities are crucial for the purpose of clustering using a novel graph-based formulation.

2.2.4 Graph Based QP (DistLQP)

In this section we describe a faster approach, based on enforcing a subset of the constraints in equation 2.1. We make use of the intuition that when a dataset is clustered, comparatively smaller distances usually determine the clusters. When two points are far apart, the exact distance between them is not critical in determining the clusters. We can build a nearest neighbor graph to determine which distances are small and possibly important for clustering. Distances smaller than or equal to a fixed threshold t_h are called nearest neighbors or "small" and distances larger than t_h are called "large". We focus our method on accurately determining the small distances, while allowing some inaccuracy in the large distances.

In Section 2.2.5, we prove a theorem that states that if we enforce a subset of the triangle inequalities based on the nearest-neighbor graph, then when one must-link or can't-link constraint is added we will still compute the same distances as computed by the brute-force method for small distances, while accurately computing the qualitative values of larger distances. That is, distances that are large before and after the full set of constraints are enforced will also be large using the sampled constraints, though their exact value may not be preserved by the sampled constraints. Since large distances are

unlikely to affect clustering it is not required to compute the exact change in the distance. This theorem gives useful insight and motivates us to use the faster version, which we will describe next. Although we use a hard threshold t_h for these proofs, in practice we find it more reasonable to adapt the threshold locally. So we use an n-nearest neighbor graph, in which the parameter n is selected depending upon the problem domain.

First, a n-nearest neighbor graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ is created from the N images, where each node (image) is connected by edges to its n nearest neighbors. We also add edges when there is a must-link or can't link constraint between two images but the corresponding distances are not lower than U or more than L respectively. If two images are already very close and must-linked, it is not required to modify the corresponding distance. Similarly, if two images are very far and are can't-linked, we do not need to change the corresponding distances. $\mathcal{E}$ denotes the nearest neighbor edges and must-link/can't-link edges of the graph.

One of our chief contributions is to determine which triangle inequalities out of $O(N^3)$ possible inequalities should be enforced in the QP. We only enforce triangle inequalities for triplets that contain at least one nearest neighbor edge and at least one must-link/can't-link edge. This reduces the number of triangle inequalities to $O(n(M + C))$. In Section 2.2.5, we prove that under some simplified assumptions even with this small subset of triangle inequalities we obtain distances with the same key properties as those obtained by enforcing all the constraints. Now $\mathcal{E}$ is augmented to make $\mathcal{E}' = \mathcal{E} \cup \mathcal{E}_{\mathcal{A}}$, where $\mathcal{E}_{\mathcal{A}}$ includes edges between two nodes that are connected to a third node with one nearest neighbor edge and one must-link/can't-link edge in $\mathcal{E}$. All edges in $\mathcal{E}'$ are used as the variables of the QP.

We also add slack variables to the must-link and can't-link constraints in the QP such that a feasible solution can be obtained even when all the pairwise constraints are not satisfied. Slack variables could also be added to the triangle inequality constraints if required, however that would increase the number of QP variables significantly. We have not found this to be necessary. The final QP (with the graph-based formulation and the slack variables for pairwise constraints) that we optimize is provided below ($d'_F(\boldsymbol{x}_i, \boldsymbol{x}_j)$ denotes the final distances obtained from the following QP):

$$
\begin{aligned}
\underset{d'_F, \xi_m, \xi_c}{\text{minimize}} \quad & \sum_{(\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{E}' - \mathcal{M} \cup \mathcal{C}} (d'_F(\boldsymbol{x}_i, \boldsymbol{x}_j) - d_I(\boldsymbol{x}_i, \boldsymbol{x}_j))^2 + \\
& \lambda_1 \sum_m \xi_m + \lambda_2 \sum_c \xi_c \\
\text{subject to } & (i)\ d'_F(\boldsymbol{x}_i, \boldsymbol{x}_j) \leq U + \xi_m, \ \ (\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{M} \cap \mathcal{E}' \\
& (ii)\ d'_F(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq L - \xi_c, \ \ (\boldsymbol{x}_i, \boldsymbol{x}_j) \in \mathcal{C} \cap \mathcal{E}' \\
& (iii)\ d'_F(\boldsymbol{x}_i, \boldsymbol{x}_j) + d'_F(\boldsymbol{x}_j, \boldsymbol{x}_k) \geq d'_F(\boldsymbol{x}_i, \boldsymbol{x}_k), \forall i, j, k \\
& \text{such that } E = \{(\boldsymbol{x}_i, \boldsymbol{x}_j), (\boldsymbol{x}_i, \boldsymbol{x}_k), (\boldsymbol{x}_j, \boldsymbol{x}_k)\} \subset \mathcal{E}' \text{and} \\
& \exists\ e_p, e_q \in E \text{ such that } e_p \neq e_q, e_p \in \mathcal{M} \cup \mathcal{C} \text{ and } e_q \in \mathcal{E} \\
& (iv)\ d'_F(\boldsymbol{x}_i, \boldsymbol{x}_j) \geq 0, \forall i, j \\
& (v)\ \xi_m \geq 0, \forall m \\
& (vi)\ \xi_c \geq 0, \forall c
\end{aligned}
\tag{2.2}
$$

where variables ξ_m and ξ_c refer to the slack variables corresponding to the must-link and can't-link constraints respectively. λ_1 and λ_2 are user defined constants. The QP in equation 2.2 is a convex problem and can be solved by using any standard QP solver.

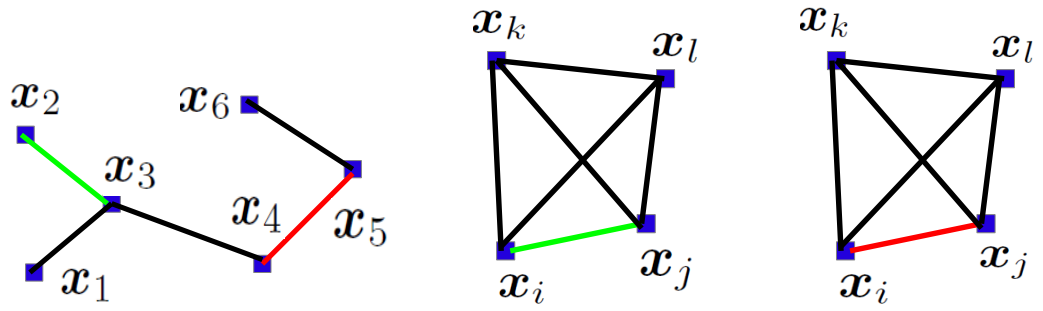
We used the interior point method for the quadratic optimization. We will refer to our proposed approach for distance learning as DistLQP (**Dist**ance **L**earning with **Q**uadratic **P**rogramming) from now on.

In Figure 2a, we consider a simple 2-d example with six points $\boldsymbol{x}_i$, where $i = 1, .., 6$. The nearest neighbor edges are shown in black, the must-link edge is shown in green and the can't link edge is shown in red. In this scenario, distances $d(\boldsymbol{x}_1, \boldsymbol{x}_3)$, $d(\boldsymbol{x}_2, \boldsymbol{x}_3)$, $d(\boldsymbol{x}_3, \boldsymbol{x}_4)$, $d(\boldsymbol{x}_4, \boldsymbol{x}_5)$, $d(\boldsymbol{x}_5, \boldsymbol{x}_6)$, $d(\boldsymbol{x}_1, \boldsymbol{x}_2)$, $d(\boldsymbol{x}_2, \boldsymbol{x}_4)$, $d(\boldsymbol{x}_3, \boldsymbol{x}_5)$, $d(\boldsymbol{x}_4, \boldsymbol{x}_6)$ are used as variables in the QP. Triangles $\Delta \boldsymbol{x}_1 \boldsymbol{x}_2 \boldsymbol{x}_3$, $\Delta \boldsymbol{x}_2 \boldsymbol{x}_3 \boldsymbol{x}_4$, $\Delta \boldsymbol{x}_3 \boldsymbol{x}_4 \boldsymbol{x}_5$ and $\Delta \boldsymbol{x}_4 \boldsymbol{x}_5 \boldsymbol{x}_6$ are used for the triangle inequality constraints.

We note that our approach is developed for clustering images, in which all distances and constraints are available when we learn the new distances. The out of sample problem is also of interest, in which a query image retrieves its nearest neighbors using the learned distances. We have performed preliminary experiments using our approach to distance learning for retrieval, but have not found that it significantly improves performance. This may be an interesting topic for future research.

2.2.5 Theoretical Analysis

Although our graph based approach is much faster than the brute-force version and empirically found to be better than the state-of-the-art approaches, we theoretically analyze a simple scenario to obtain further insight into our faster approach. We consider the case in which one constraint is added to a set of distances that obey the triangle inequality (as mentioned in the introduction, the distances we use generally do obey the triangle



(a) Six points in 2-d. (b) Must-link. (c) Can't-link.

Figure 2: Simple examples in 2-d Euclidean space (green denotes must-link edge and red denotes can't-link edge) to explain our proposed approach.

inequality). We state and prove a theorem in this section which shows that in this case, even with the subset of triangle inequalities we enforce, key properties of the brute-force QP solution are preserved.

Note that $d_I(\boldsymbol{x}_i, \boldsymbol{x}_j)$ are the initial distances, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_j)$ are the distances obtained by solving the QP in equation 2.1 and $d'_F(\boldsymbol{x}_i, \boldsymbol{x}_j)$ are the distances obtained by solving the QP in equation 2.2. All the lemmas and the theorem are proved for four points $\boldsymbol{x}_i$, $\boldsymbol{x}_j$, $\boldsymbol{x}_k$, $\boldsymbol{x}_l$ in Figure 2b and 2c for simplicity. However they can easily be extended to any number of points in general. The final distance is set to be 0 for a must-link and L for a can't-link, where L is the maximum of all initial distances. Due to space constraints we provide most of the proofs of our lemmas and the theorem in the Appendix 6.1 except lemma 1. We include the proof for lemma 1 in this chapter to give the reader some general idea about our proofs.

Here we note some initial triangle inequality constraints, which we will use to prove

the lemmas:

$$d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_i, \boldsymbol{x}_l) \tag{2.3a}$$

$$d_I(\boldsymbol{x}_j, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_j, \boldsymbol{x}_l) \tag{2.3b}$$

$$d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_i, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \tag{2.3c}$$

$$d_I(\boldsymbol{x}_j, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \tag{2.3d}$$

Lemma 1. *If there is a must-link constraint between points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, for any other point $\boldsymbol{x}_k$, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) \geq min(d_I(\boldsymbol{x}_i, \boldsymbol{x}_k), d_I(\boldsymbol{x}_j, \boldsymbol{x}_k))$.*

Proof. We will refer to Figure 2b and the QP in 2.1 for this proof. We find the minimum cost solutions for the graph (Figure 2b) considering triangles $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$ and $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_l$ and show that this solution does not violate the triangle inequality for $\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_l$ and $\Delta \boldsymbol{x}_j \boldsymbol{x}_k \boldsymbol{x}_l$. When an optimal solution obtained using a subset of the constraints ($\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$ and $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_l$) also satisfy other constraints ($\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_l$ and $\Delta \boldsymbol{x}_j \boldsymbol{x}_k \boldsymbol{x}_l$) which were not enforced, it is not required to change the optimal solution. Thus the solution which we obtain in this proof is optimal.

There is a must-link constraint between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. To minimize the objective function in 2.1, and to avoid violation of the triangle inequality for triangle $\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_j$, we will have $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = d_F(\boldsymbol{x}_j, \boldsymbol{x}_k) = \frac{d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_k)}{2}$. Similarly for $\Delta \boldsymbol{x}_i \boldsymbol{x}_l \boldsymbol{x}_j$, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) = d_F(\boldsymbol{x}_j, \boldsymbol{x}_l) = \frac{d_I(\boldsymbol{x}_i, \boldsymbol{x}_l) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_l)}{2}$.

Now we show that the triangle inequality holds for $\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_l$ and $\Delta \boldsymbol{x}_j \boldsymbol{x}_k \boldsymbol{x}_l$ with $d_F(\boldsymbol{x}_k, \boldsymbol{x}_l) = d_I(\boldsymbol{x}_k, \boldsymbol{x}_l)$. Adding equations 6.1a and 6.1b, we get $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq$

$d_F(\boldsymbol{x}_i, \boldsymbol{x}_l)$. Similarly adding equations 6.1c and 6.1d, we get $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_k, \boldsymbol{x}_l)$. By symmetry $d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_F(\boldsymbol{x}_i, \boldsymbol{x}_k)$. So the triangle inequality holds for $\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_l$. Similarly, we can prove that triangle inequality holds for $\Delta \boldsymbol{x}_j \boldsymbol{x}_k \boldsymbol{x}_l$. So $d_F(\boldsymbol{x}_k, \boldsymbol{x}_l) = d_I(\boldsymbol{x}_k, \boldsymbol{x}_l)$ is not required to change. Thus $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) \geq min(d_I(\boldsymbol{x}_i, \boldsymbol{x}_k), d_I(\boldsymbol{x}_j, \boldsymbol{x}_k))$ because $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = \frac{d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_k)}{2}$. □

Lemma 2. *If there is a must-link constraint between points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, only distances $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k)$ and $d_F(\boldsymbol{x}_j, \boldsymbol{x}_k)$,$\forall k$, differ from the initial distances. All other distances remain the same.*

Proof. Follows from the proof for Lemma 1. □

Lemma 3. *If there is a can't-link constraint between points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, for any other point $\boldsymbol{x}_k$, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) \geq d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$.*

Proof. This proof is provided in Appendix 6.1. □

Lemma 4. *If there is a can't-link constraint between points $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$, only distances $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k)$ and $d_F(\boldsymbol{x}_j, \boldsymbol{x}_k)$,$\forall k$, may differ from the initial distances. All other distances remain the same.*

Proof. Follows from proof for Lemma 3. □

Now we state a theorem and show that even with the triangle inequality constraints that we enforce in QP in equation 2.2, distances are changed in a similar way as they would have, had we enforced all the triangle inequality constraints (as in equation 2.1). The proof requires the above lemmas and is provided in the Appendix 6.1. We note that

distances that are smaller than or equal to a fixed threshold t_h are called nearest neighbors or "small" and distances larger than t_h are called "large" for all d_I, d_F and d'_F.

Theorem 5. *If there are N points $\{\boldsymbol{x}_1, \boldsymbol{x}_2, ..., \boldsymbol{x}_N\}$ with a must-link/can't-link constraint present among them, for any fixed threshold t_h and for any pair $\{\boldsymbol{x}_i, \boldsymbol{x}_k\}$,*

I. if $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) > t_h$ and $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) > t_h$, then $d'_F(\boldsymbol{x}_i, \boldsymbol{x}_k) > t_h$.

II. if $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) > t_h$ and $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) \leq t_h$, then $d'_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = d_F(\boldsymbol{x}_i, \boldsymbol{x}_k)$.

III. if $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) \leq t_h$ and $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) > t_h$, then $d'_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = d_F(\boldsymbol{x}_i, \boldsymbol{x}_k)$.

IV. if $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) \leq t_h$ and $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) \leq t_h$, then $d'_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = d_F(\boldsymbol{x}_i, \boldsymbol{x}_k)$.

We prove that when one must-link or can't-link constraint is added distances learned using only a subset of triangle inequalities will be exactly the same as the distances learned by the brute-force approach except when a large ($> t_h$) distance remains large using the brute-force approach. Since large distances are unlikely to affect clustering, we prove that if a large distance remains large using the brute-force approach, it will also be large using our fast approach but we are not required to compute the exact change in those distances.

We note that using our method we can modify a maximum of $4n$ distances with each pairwise constraint, which is much higher than the one pairwise distance modification when no constraint propagation happens. If there are a total of $(M + C)$ pairwise constraints available, $4(M + C)n$ pairwise distances may get modified.

2.3 Experimental Results

We run experiments on three different domains, leaves, faces and detected faces from video images. We describe the datasets below:

- **Leaf-1042 [14]** : This data set contains 1042 leaf images from 62 classes. Chi-square distances [14] between curvature histograms are used as the initial distances for our algorithm.
- **Face-1000 & Face-10000 [49]** : These datasets are subsets of the Pubfig dataset. The complete Pubfig dataset has around $58,797$ celebrity images of 200 persons. Since images in Pubfig are taken in completely uncontrolled settings with non-cooperative subjects there is a large variation in pose, lighting, expression, and imaging conditions. Due to this large set of variations, it is extremely hard to cluster face images from Pubfig. Distances are calculated based on the output of a pre-trained classifier [9]. Face-1000 has 1000 images from 50 classes, with each class having 20 images. Face-10000 is a much larger dataset with 10000 images from 50 classes, where each class has 200 images.
- **BF0502-subset-1 and BF0502-subset-2 [26]**: These datasets contain extracted faces from videos. The original video dataset BF0502 [26] has $27,504$ extracted faces of 11 main cast members and others from the TV series "Buffy the Vampire Slayer". BF0502-subset-1 has 687 faces of 6 persons and is exactly same as the BF0502-subset used in [76]. Since BF0502-subset-1 had only 6 clusters we created another subset called BF0502-subset-2 which contains 600 face images of 11 persons. Unlike the other datasets, previous work on this video data has shown strong

results using vector space representations. So one goal of these experiments is to provide a comparison to a state-of-the-art method on its home turf.

Since our method does not require a vector space representation, we are free to make use of a non-vector space, robust distance on this data, which we find improves results. For each image pair, the squared difference is computed in each dimension. Then only the smallest 80% of those differences are summed and the square root is taken to obtain a robust distance. Note that while such robust distances are common in computer vision, they cannot be represented as a Euclidean distance in any vector space.

We describe the algorithms below that we compare in our experimental evaluation. We compare our approach to previous vector space methods by first using Multidimensional Scaling (MDS) [15] to embed the images in a vector space.

- **K-means [56][1]:** In this approach the distances corresponding to the must-link pairs are set to zero and can't-link pairs are set to a very high value. This version of K-means works better than the K-means without any distance modification.
- **K-means + DistLQP:** Distances learned from our approach are used along with the K-means clustering algorithm.
- **COP-Kmeans [74]:** This is a traditional constrained clustering algorithm, which avoids constraint violations.
- **COP-Kmeans + DistLQP:** Distances learned from our approach are used along with the constrained clustering algorithm COP-Kmeans.

[1] Since we do not have the vector representation of images we compute the medoid instead of the mean in the update step.

- **ITML [24] + MDS [15]:** MDS is used to project the distances back to a vector space and Information Theoretic Metric Learning (ITML) is used to learn distances.
- **HMRF-com [76] + MDS [15]:** MDS is used to project the distances back to vector space and HMRF-com is used for clustering.
- **COP-Kmeans + NPKL [35]:** Distances are learned using a Non-parametric Kernel Learning (NPKL) approach and are used along with COP-Kmeans.

The major steps in our approaches K-means + DistLQP and COP-Kmeans + DistLQP are:

1. We have the initial distances and the randomly generated pairwise constraints.
2. We set the distances corresponding to must-link to 0 and can't-link to a high value.
3. We learn a new set of distances using our approach.
4. K-means or COP-kmeans is run with the new distances.
5. We get a clustering and evaluate the clustering solution.

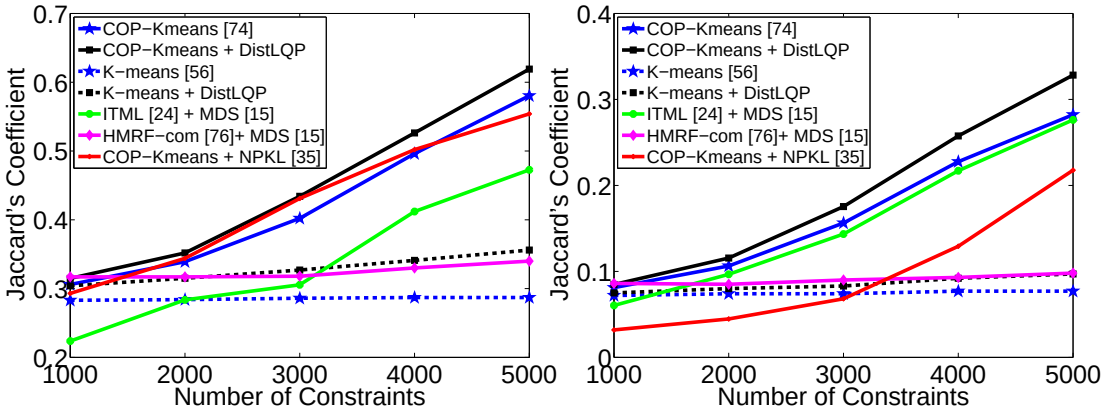
In the baselines K-means and COP-Kmeans all of the above steps are used except step 3, where we learn a new set of distances using our approach.

We use Jaccard's coefficient [68], which varies from 0 to 1 (1 is the best), as the clustering evaluation metric for Leaf-1042, Face-1000 and Face-10000. Jaccard's Coefficient of a clustering C with respect to a ground truth clustering G is defined as:

$$\text{JCC}_G(C) = \frac{SS}{SS + SD + DS} \tag{2.4}$$

where SS is the number of pairs that are in the same cluster in both C and G, SD is the number of pairs that are in same cluster in G but in different clusters in C, and DS is the number of pairs that are in different clusters in G but are in same cluster in C.

For BF0502-subset-1 and BF0502-subset-2, we use the same evaluation metric as used by [76], which computes the accuracy based on the confusion matrix derived from the predicted labels and the ground truth labels. Each algorithm is run with 25 random initializations and with 5 different random sets of pairwise constraints for a fixed constraint set size. The average of all of them is reported. Since the video data has one fixed set of constraints, the average of 25 random initializations are reported. For Leaf-1042 and Face-1000, the total number of constraints is varied from 1000 to 5000, with 20% of the constraints being always must-link constraints. For Face-10000, one fixed constraint size of $25,000$ (20% must-link) is used. However with the video dataset BF0502-subset-1 we have a fixed set of 687 must-link constraints and 180 can't-link constraints. For BF0502-subset-2, we have 600 must-link and 189 can't-link constraints available. These constraints are produced following the same method as used by [76]. The nearest neighbor graph size n is set to a fixed value of 20 for all datasets. The must-link threshold U is always set to a low value of 0.01, whereas the can't-link threshold varies from one dataset to another depending on the maximum pairwise image distance. We set that to $L = \frac{max(d_I(\boldsymbol{x}_i,\boldsymbol{x}_j),\forall i,j)}{\alpha}$, where α is a parameter. α is set to be 8 for leaves, 3 for faces and 2 for the video subsets. λ_1 is set to be 10 and λ_2 is set to be 1 for all the datasets.



(a) Leaf-1042 (b) Face-1000

Figure 3: Comparison of our proposed distance learning approach with other constrained clustering methods.

Table 2.1: Results for Face-10000 (HMRF-com [76] and NPKL [35] were not fast enough for this dataset).

K-means [56]	K-means+DistLQP	COP-Kmeans [74]	COP-Kmeans +DistLQP	ITML [24]+MDS
0.0784	0.0803	0.0818	**0.0831**	0.0711

Table 2.2: Results for BF0502-subset-1 and BF0502-subset-2 (confusion matrix based evaluation [76]).

Algorithms	BF0502-subset-1	BF0502-subset-2
K-means [56]	0.3640	0.2578
K-means + DistLQP	0.3749	0.2635
COP-Kmeans [74]	0.3865	0.2603
COP-Kmeans + DistLQP	0.4035	**0.2695**
ITML [24]	0.3566	0.2177
HMRF-com [76]	**0.4576**	0.2633
COP-Kmeans+NPKL [35]	0.3223	0.2440

2.3.1 Discussion of Results

Our approach outperforms all other algorithms for all datasets except for BF0502-subset-1, where HMRF-com uses the actual vector representation of the images, and only six clusters are present. HMRF-com usually works well with a small number of clusters and when the actual vector representation of the images are available. However when the number of clusters increases, this approach does not perform well because there are not usually enough samples from each class to learn the parameters of the Gaussian distributions used in their approach. Also when we perform MDS to convert the distances to a vector space representation, the representation is often inaccurate. When these vectors are used along with HMRF-com and ITML, their performance degrades because of the poor vector space representations. We used 60 dimensional MDS for all the datasets except the video subsets, where we had access to the original vector space representations. We analyze the results of each dataset below:

- **Leaf-1042:** We provide the results for Leaf-1042 in Figure 3a. Our approach outperforms all other algorithms by a significant margin. For example, with $5,000$ pairwise constraints, our approach reduces the error (1-Jaccard's Coefficient) from 0.42 (with the second best approach) to 0.38, eliminating 9.5% of the error. The proposed method takes around 4 minutes (in matlab) with $5,000$ pairwise constraints to learn a new set of distances for this dataset.
- **Face-1000:** Results for Face-1000 are shown in Figure 3b. In this dataset also our approach outperforms all other approaches by a significant margin. For example, when $5,000$ pairwise constraints are available, our approach reduces the error from

0.72 (with the second best approach) to 0.67, eliminating 7% of the error. Our approach takes around 2 minutes to learn a new set of distances with $5,000$ pairwise constraints for this dataset.

- **Face-10000:** Even for this dataset, our approach outperforms other datasets (see Table 2.1). We run our approach on this dataset to see if our approach can help to cluster larger datasets. Although the proposed approach improves clustering for this dataset, it is still far from perfect. From the performance of all the algorithms, it seems that clustering such a large dataset is an extremely hard problem and it is probably hard to obtain a reasonable clustering with state-of-the-art image features. It takes around 40 minutes for our approach to learn the new distances with the $25,000$ constraints we use. However HMRF-com [76] and NPKL [35] were not fast enough that we could run these algorithms for Face-10000.
- **BF0502-subset-1 and BF0502-subset-2:** Results for these datasets are provided in Table 2.2. For these two datasets we used the original vector space representation (after dimensionality reduction with Principal Component Analysis as suggested by [76]) for HMRF-com [76] and ITML [24]. We did not have to do Multidimensional scaling to obtain a vector space representation for these datasets. Although for BF0502-subset-1, HMRF-com outperforms our approach by a significant margin, we note that we do not use the actual vector space representation in our approach. However for BF0502-subset-2, we outperform all other approaches, which demonstrates that HMRF-com's performance may be more sensitive to the number of clusters. For the video subsets, it took around 1.3 minutes to learn a new set of distances for clustering. We note that while our approach is not really designed for

this scenario, in which a vector space representation of the images is available, it still performs very well, outperforming other approaches when the number of clusters is 11. In part, this is because our approach offers the freedom to make use of robust distances that deviate from the original vector space.

2.4 Conclusion

In this chapter we propose a novel method for learning distances between images when pairwise must-link and can't-link constraints are provided. Our method uses only pairwise distances between all images and does not require the vector space representation of the images. We apply these learned distances for clustering images. We run experiments for leaf and face clustering. We also use our approach for clustering faces extracted from videos. We demonstrate that our approach outperforms other approaches and produces state-of-the-art clustering results. Although the experimental results are shown for images, the proposed approach can be applied to any other domain.

Chapter 3: **Active Image Clustering**

In this chapter we present an algorithm to actively solicit human input to cluster images using pairwise constraints. Algorithms give information about similarity of images but humans are much more accurate in judging if two images are the same or not. We incorporate that information (must-link and can't-link constraints) from humans and use that in a clustering algorithm. Since human effort is costly we would like to minimize the total human effort required for perfect clustering. We propose a novel algorithm to choose an image pair out of $O(N^2)$ image pairs to show to a user to get a constraint on it. In a clustering assignment each image pair may be either in the same cluster or in different clusters. We simulate inverting those pairwise relationships and see how that affects overall clustering produced by human input. We choose the pair for which the expected change in overall clustering is maximum.

3.1 Related Work

Recently computer vision researchers have been interested in melding active learning [3, 65] with computer vision. We will discuss a few recent works in this area. In [72], the authors propose a method for active selection of a batch of images for labeling using

a SVM classifier where the total cost for labeling at each iteration is constrained by a predefined budget. The proposed method is shown to be useful for object recognition, activity recognition and image retrieval. The authors in [70] propose a method of training a large scale object detector in an active manner. The proposed algorithm iteratively poses annotation requests to humans and gathers relevant images for labeling from Flickr, where the true labels of the data are not known beforehand. In [71], the authors propose a method for active frame selection for labeling such that a video sequence is labeled (at the pixel level) with as little human input as possible. They propose two methods, one is a basic flow-based approach while the other is a more sophisticated approach that uses a flow model within a Markov Random Field [19]. [17] propose an interactive human-in-the-loop method for bird classification. Humans will answer simple questions on visual attributes of birds such that a bird can be classified with limited human interaction. In [16], the authors propose a method for large scale interactive learning of deformable part models [28]. The authors in [67] propose an active technique for learning appearance and contextual models for scene understanding simultaneously. This is the first multi-class active learning technique that takes contextual interactions between different parts of a scene image into account.

The authors in [32] proposed an active learning method for labeling unlabeled data given a set of labeled instances. They use an optimistic information theoretic way to use the unlabeled data; that is to find the instance whose best possible label assignment could maximize the mutual information about the labels of the remaining unlabeled instances. Also in [22], the authors propose an active learning method, which uses an information-theoretic diversity measure, to label samples from a large collection of unlabeled images.

In [43] the authors propose a method for active learning that uses the expected value of information to label images. However all of these approaches want to get labels of images and do not try to partition the dataset as we do. Note that we are only computing the expected change in clustering, which is different from pure information theoretic approaches, because it is not clear what is the best possible way to quantify information gain for active learning in clustering. However our approach is inspired by information theoretic approaches for active learning in general. In [36] the authors propose an active learning technique for object recognition, which aims to minimize the entropy of the current unlabeled set of images given a labeled set. The authors in [42] proposed the first multi-class active learning technique that requires binary (yes or no) user inputs. This method points out the difficulty of providing category labels for large multi-class problems and suggests using pairwise similarity questions for labeling with access to an already available labeled set. In a clustering problem we usually do not have any labeled data available for selection of these pairwise questions. In [41] the authors proposed an active learning scheme for large multi-class problems where the goal is to select the data point that strengthens the existing classification model the most in terms of its discriminative capabilities.

Clustering has been an interesting topic of research in machine learning, computer vision and natural language processing for more than 50 years. We refer to [40] for a nice and comprehensive review. Combining active learning with constrained clustering [8] has long been an area of interest for machine learning researchers. In [6], the authors propose an active clustering algorithm. They suggest two major phases in an active learning setting namely "Explore" (cluster center initialization) and "Consolidate" (data points are added

to cluster centers). In problems with a large number of clusters (which is very common in the image clustering domain), the "Explore" stage itself takes a large number of questions to initialize the distinct cluster centers. The authors in [57] have proposed an approach that uses a min-max criterion to find informative questions. They also rely on the "Explore" stage in the beginning as [6] does. There are also a couple of active clustering algorithms [75, 79] based on spectral eigenvectors, but they are good for two-cluster problems only. Very recently the authors in [78] proposed an active spectral clustering algorithm with k-nearest neighbor graphs that can be used for more than two-cluster problems. We compare with their approach and show that our proposed approach is much better. In [38], the authors have proposed an active framework for constrained document clustering. This paper is philosophically similar to our approach, i.e., they also try to ask questions to maximize the gain. They begin with a skeleton structure of neighborhoods covering all the clusters. They then search for an informative data pair to match an unlabeled data point to one of the centroids of the existing neighborhoods. Also, they use an "Explore" stage to build an initial skeleton structure, which we already know to be a potential problem when there is a large number of clusters. Another approach by Huang *et al.* [37] has an active constrained clustering algorithm for documents with language modeling; it is not clear how we could adopt this algorithm for image clustering. In [5] the authors proposed a semi-supervised method for clustering using seeding, where labeled data is used to generate better initial cluster centers and to constrain the clustering solution; this approach does not have any active learning component. Some methods [30] were also developed to use multiple humans to cluster datasets, where each user is asked to locally cluster a part of the dataset and finally those results are aggregated to find a complete

partitioning.

Some active constrained clustering approaches are also known for image clustering [13, 31]. In [13], the authors have proposed a heuristic, which works well but has several parameters that must be tuned properly for different datasets. In [31], the authors take a fuzzy clustering based approach to find images near cluster boundaries to form useful data pairs.

As described in this section, there are many approaches in active learning, which want to maximize the information obtained from humans. Some of these methods were proposed for tasks, that are different from clustering with pairwise constraints. However there are some approaches [38, 57], that were proposed specifically for clustering with constraints and were designed to maximize the information gain from humans. However these active approaches measured the information gain using a local approach, where they do not consider the affect of a constraint on the complete dataset. For example, some constraints might be useful locally, i.e., can change clustering of a few points, but may not have high impact if the overall clustering of the dataset is considered. The novelty of our approach is in judging the effectiveness of a constraint by measuring how much it changes the overall clustering (a global approach), i.e., we expect a useful constraint to change assignment of many points in the dataset. A local approach can be useful when we already have a reasonable clustering to start with and we want to correct a few existing mistakes in that. On the other hand, a global approach like ours is more suitable for a problem like image clustering, where the initial clustering itself is very poor and it needs large changes to achieve a reasonable clustering. Thus, our method can ask useful questions from the very beginning and can achieve a good clustering by asking fewer

questions than other methods.

3.2 Our Approach

We now describe our approach to active clustering. We first motivate this approach with a simple example, and then describe technical details.

3.2.1 High Level Intuition

We have developed a novel algorithm that determines which questions to ask a person in order to improve clustering. This algorithm is based on the intuitive idea of asking questions that will have the largest expected effect on the clustering. This really has two components. First, if we ask a person to compare two objects, her answer will only affect the clustering immediately if it differs from current clustering; that is, if the person says either that two objects that are not currently in the same cluster should be, or that two objects that are in the same cluster should not be. Any answer that differs from the current clustering must result in moving at least one object to a different cluster, but some answers will affect many more objects. So the second component of our algorithm is to ask questions whose answers might have a big effect on the way objects are clustered.

To provide a better intuition about our approach, we consider the simple toy example of clustering a set of 2D points. Figure 1 shows a collection of such points as black disks. The circles indicate four clusters that might be formed by an automatic clustering algorithm. We have marked five of these points with the letters “A” to “E” for ease of reference. Now suppose that an expert can compare two of these points and tell us whether

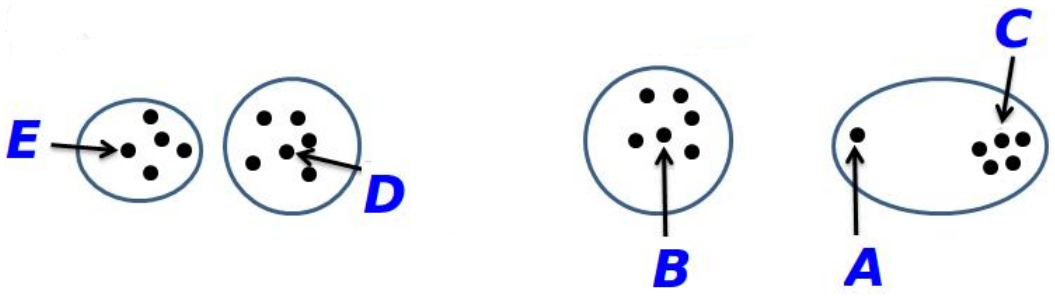


Figure 1: Toy example to motivate our approach.

they truly belong in the same cluster. Considering the following possibilities, we find:

- Comparing B and C is not that desirable, since it is likely that we have already correctly placed them in different clusters. A human opinion about these two points is unlikely to change anything.
- Comparing A and B (or A and C) will tell us in which cluster A truly belongs to. Since A is between two clusters, it is quite possible that this question will change the cluster to which we assign A. However, the answer to this question will only affect A.
- Comparing D and E will provide the most information. These two clusters are close, and it is somewhat ambiguous whether they should be separated into two clusters, or joined into one. So it is reasonably likely that a person might say D and E belong in the same cluster. If they do, this will lead us not only to treat D and E differently, but in fact to join the two clusters together, affecting the grouping of many points.

Consequently, we select questions for human attention that maximize the product of the probability that the answer will cause a change in our current clustering and the size of

this change, should it occur. Finding the best such question can potentially require a large amount of computation. If we are clustering N objects, then there will be $O(N^2)$ same-or-different questions to consider, and for each we must determine its possible effects. For this reason, we adopt a simple, greedy clustering algorithm. Without human assistance, this algorithm does not perform well, but by using a simple clustering algorithm, we can more easily and quickly select good questions to ask a human, and rapidly improve our clustering performance. In Section 3.2.3 we explain the brute force version of our proposed algorithm and in Section 3.2.4 we describe a faster version of our algorithm. This faster version is possible due to the use of a simple and greedy clustering algorithm. In order to further speed up our algorithm, we have also experimented with two additional heuristics:

First, when our estimate of the probable response of a human indicates that it is very likely that the human response will agree with the current clustering, we do not bother to simulate the results of a different response. For all datasets we exclude simulation of pairs which are very unlikely to be in the same cluster. For larger datasets (of size more than 1000), we initially use K-means [56] to group very close points together. These points are always clustered together. We represent them using their centroids and then run our algorithm. We refer to this heuristic as H1.

Second, we observe that when we simulate an additional constraint between a data pair, change in clustering assignments is often limited to clusters that contain the points in that pair. Determining those changes is much faster than checking for all possible changes. We perform experiments with this approximation and we find that it makes our algorithm's performance a little worse but much faster. We refer to this heuristic as H2.

3.2.2 Components of Our Algorithm

We now explain our algorithm and its components formally. We define the best image pair that we will present to a person for labeling as:

$$(\hat{d_i}, \hat{d_j}) = \arg\max_{d_i, d_j} E_J(d_i, d_j) \tag{3.1}$$

$E_J(d_i, d_j)$ is the expected change in the clustering if a question is asked about an image pair d_i and d_j. Since we do not know the ground truth clustering, we cannot choose image pairs that are guaranteed to increase the quality of the clustering. One of the main insights of our work is that by finding pairs that most rapidly *change* the clustering we quickly converge to a good clustering. Now, we formally describe how we compute the components needed to determine $E_J(d_i, d_j)$ given that we have a matrix containing all pairwise distances for a dataset.

3.2.2.1 Consensus Clustering based Pairwise Probability Distribution

We first need to estimate the probability that each data pair will belong to the same cluster. We have borrowed ideas from consensus clustering [62] (also referred to as aggregation of clustering) to find those probabilities. Consensus clustering typically computes multiple clusterings of the same dataset and then produces a single clustering assignment that reflects a consensus. This can be developed as an optimization problem that is known to be NP-complete. Consensus clustering in unsupervised learning is similar to ensem-

ble learning in a supervised framework. However, we avoid optimization and just use multiple clusterings to estimate the probability that a pair of points belong to the same cluster.

Specifically, if we have N data points and S clusterings, let $\mathbf{A_s}$ be the $N \times N$ matrix where element (i, j) is 1 if the i-th and j-th data points (referred as d_i and d_j respectively from now on) are in the same cluster in the s-th clustering, and zero otherwise. We use the K-means algorithm to produce clusters, each time beginning with different random initial cluster centroids. Now if $\mathbf{P}$ is the probability matrix for N data points where again element (i, j) is the pairwise probability of d_i and d_j being in the same cluster,

$$\mathbf{P} = \frac{1}{S} \sum_{s=1}^{S} \mathbf{A_s} \tag{3.2}$$

If (d_i, d_j) is any pair of points and R is a random variable corresponding to pair (d_i, d_j) resulting from the above random process then we estimate:

$$P(d_i, d_j) = \text{prob}(d_i = d_j | R) = R \tag{3.3}$$

$d_i = d_j$ implies d_i and d_j are from same class. We experimentally verify that this method produces reasonable values. In Figure 2 (using images and pairwise distances from [48]), we plot prob$(d_i = d_j|R) = R$ and a histogram generated from the results of those experiments showing the fraction of times that a pair with a given R is from the same cluster. Our heuristic provides a good estimate of prob$(d_i = d_j|R) = R$, which becomes more accurate with larger data sets.

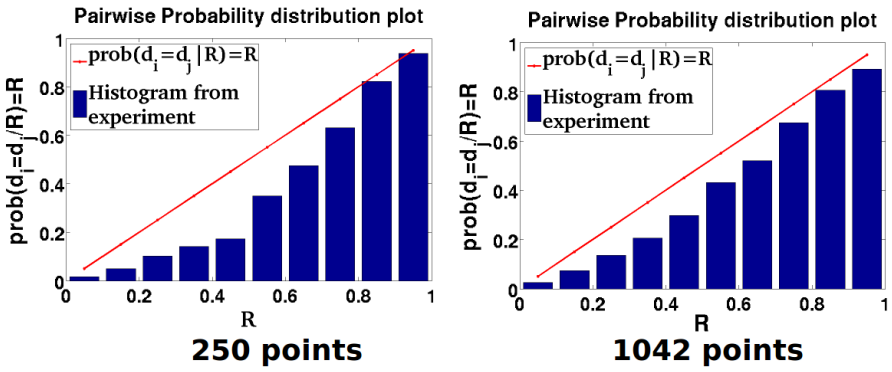


Figure 2: Pairwise probability distribution of real data.

3.2.2.2 Minimum Spanning Forests (MSF)

We perform clustering by creating a *Minimum Spanning Forest* (MSF). We define an MSF as a collection of trees which we get if we run Kruskal's algorithm [46] and stop when we have K spanning trees. We can think of clustering as finding a forest with K spanning trees from a set of disconnected nodes. We use a constrained clustering algorithm that is very similar to Kruskal's algorithm, but also respects constraints. In the clustering community a similar approach is well-known as bottom up or hierarchical agglomerative clustering (HAC). We assume we are provided with the distance matrix for a given dataset. We can consider each image of the dataset as an individual node in a complete graph $G = (V, E)$ and let their mutual distances represent edge weights. We can also have information about a pair of images being in the same cluster ("must-link" constraints) or different clusters ("can't-link" constraints). Let us assume we create a copy of the graph G without any edges and call it $F = (V, \emptyset)$. First, we add edges corresponding to must-link constraints to the forest F. Next we sort the edges in E in an ascending order of their weights and store them in a queue E_p. We start popping edges from E_p and add them to F. While doing this, we always maintain the tree structure, i.e. do not add an edge between two

nodes if the nodes are already connected. We will also not add an edge between two trees if there is any can't-link constraint between any of the pairs of nodes in those two trees. We continue doing this until we have K trees in the forest (referred as MSF in future). We refer to this algorithm as HAC with constraints (HACC).

We selected MSF as the base clustering algorithm for several reasons. First, MSF does not have any randomness involved. Since we want to measure the change in clustering for all possible pairs, we would not like the change in clustering to be affected by anything but the added constraint. Second, our high level idea is to simulate the results of all questions so that we can choose the question that has the greatest expected affect on the clustering. Doing this in a brute force way, even with the simplest clustering algorithm is not practical. So it is critical that we use a semi-supervised clustering algorithm in which it is possible to perform $O(N^2)$ incremental clustering operations efficiently. We have been able to develop fast ways of simulating the incremental effects of single constraints while using MSF as the semi-supervised clustering algorithm. Third, MSF's sensitivity to constraints, i.e., addition of some constraints can potentially change the MSF clustering result by a large amount, which helps us to measure the effectiveness of a constraint in a better way. However we emphasize that our use of this constrained version of minimum spanning forests is not meant to suggest that this semi-supervised clustering algorithm is competitive in performance with other methods. We expect MSF to work well along with the pairwise constraints which we obtain using our proposed active algorithm and that is demonstrated more in the experimental results (Section 3.3).

Since we are working with hierarchical clustering with constraints, we have to discuss feasibility and dead-end issues [23]. The feasibility problem is defined as given a

set of data and set of constraints, does there exist a partitioning of the data into K clusters? In our problem the answer is obviously yes because all of the constraints are true. However determining whether there is a feasible solution which satisfies all constraints is NP-complete [23]. In HACC, dead-end situations (reaching an iteration with more than K clusters, where no further pair of clusters can be joined without violating any of the constraints) can occur in principle, but in practice we do not encounter this problem.

3.2.3 Our Algorithm (Active-HACC)

In 3.2.2.1, we estimate a distribution that allows us to determine the probability that a person will provide a constraint that differs from the current clustering. In this section, we determine the magnitude of the change this will have on the current clustering. To do this, we define a measure of similarity between two clustering assignments, which we call Relative Jaccard's Coefficient, by analogy to the Jaccard's Coefficient [68] (detail in Chapter 2). If C_1 and C_2 are two clustering assignments of the same dataset, the Relative Jaccard's Coefficient of clustering C_2 with respect to C_1 is defined as:

$$\text{JCC}_{C_1}(C_2) = \frac{SS}{SS + SD + DS} \tag{3.4}$$

where SS is the number of pairs that are in the same cluster in both C_1 and C_2, SD is the number of pairs that are in same cluster in C_1 but in different clusters in C_2, and DS is the number of pairs that are in different clusters in C_1 but are in same cluster in C_2. This becomes the traditional Jaccard's coefficient if C_1 is the ground truth clustering.

Now, we describe our algorithm, assuming that we have asked q questions and need

to determine the $(q+1)$-th question to ask. We define:

D: The dataset, that should be clustered.

N: Number of images in the dataset.

K: Number of clusters.

d_i, d_j: Any pair of images from D.

C_q: The set of can't-link constraints obtained after we have asked q questions.

M_q: The set of must-link constraints obtained after we have asked q questions. Note $|C_q| + |M_q| = q$.

$H_q = \text{HACC}(D, C_q, M_q)$: HACC is the clustering function on a given dataset D, using the must-link constraints (M_q) and can't-link constraints (C_q). H_q is the clustering that is produced.

$J_{q+1}^{d_i,d_j,\mathbf{y}} = \text{JCC}_{H_q}(\text{HACC}(D, C_q, M_q \cup d_i = d_j))$: Relative Jaccard's Coefficient of a clustering after $q+1$ questions with respect to H_q, if the $(q+1)$-th constraint is that d_i and d_j are in same cluster.

$J_{q+1}^{d_i,d_j,\mathbf{n}} = \text{JCC}_{H_q}(\text{HACC}(D, C_q \cup d_i \neq d_j, M_q))$: Relative Jaccard's Coefficient of a clustering after $q+1$ questions with respect to H_q, if the $(q+1)$-th constraint is that d_i and d_j are not in same cluster.

Now, we ask the user about the pair:

$$(\hat{d}_i, \hat{d}_j) = \arg\max_{d_i,d_j} E_J(d_i, d_j) \tag{3.5}$$

Where $E_J(d_i, d_j)$ is the expected change in the Relative Jaccard's Coefficient and

is defined as follows:

$$E_J(d_i, d_j) = |\text{JCC}_{H_q}(H_q) - (P(d_i, d_j)J_{q+1}^{d_i,d_j,\mathbf{y}} + (1 - P(d_i, d_j))J_{q+1}^{d_i,d_j,\mathbf{n}})| \tag{3.6}$$

Note that $\text{JCC}_{H_q}(H_q) = 1$ and $P(d_i, d_j)$ is the probability of d_i and d_j being in the same cluster.

Now we can see that if points d_i and d_j are in the same cluster after q questions, then $\text{HACC}(D, C_q, M_q \cup d_i = d_j) = H_q$ and if they are in different clusters then $\text{HACC}(D, C_q \cup d_i \neq d_j, M_q) = H_q$. So we have:

- d_i and d_j are in the same cluster after q questions:

$$E_J(d_i, d_j) = |(1 - P(d_i, d_j))(1 - J_{q+1}^{d_i,d_j,\mathbf{n}})| \tag{3.7}$$

- d_i and d_j are in different clusters after q questions:

$$E_J(d_i, d_j) = |P(d_i, d_j)(1 - J_{q+1}^{d_i,d_j,\mathbf{y}})| \tag{3.8}$$

Using this approach, we find the data pair that will produce the greatest expected change in the clustering. After receiving an answer from the user, we update our constraints and continue.

When we plot the clustering performance, we show the Jaccard's Coefficient relative to ground truth, as the number of questions increases. This curve does not always increase monotonically, but it generally increases. Pseudo code of our proposed active

algorithm is shown in 1.

Algorithm 1 Active-HACC

Given: D, Max_questions, $M_q = \emptyset$, $C_q = \emptyset$, num_questions=0
while num_questions $\leq$ Max_questions **do**
  HACC(D,M_q,C_q)
  For all pairs (d_i, d_j), evaluate $E_J(d_i, d_j)$
  Find the pair (d_i, d_j) with maximum $E_J(d_i, d_j)$
  Ask and Update M_q and C_q
  num_questions=num_questions+1
end while
Output: HACC(D,M_q,C_q)

3.2.3.1 Complexity of the algorithm

We now discuss the rather high complexity of the brute-force version of Active-HACC. We will then consider several optimizations and heuristics to improve this run time. For each question we have $O(N^2)$ (N is the number of images) possible pairs. To simulate what will happen for each pair in a brute force manner, we will have to run the constrained clustering algorithm for each pair. We now describe the complexity of Active-HACC.

Kruskal's algorithm [46] for minimum spanning tree runs in $O(N^2 \log N)$ time (if $|E| = O(N^2)$). HACC also has $O(N^2 \log N)$ complexity from a very similar analysis to Kruskal's, except for the issue of keeping track of the can't-links. To do this efficiently, we maintain an $l \times l$ lower triangular matrix **A** in which l is the current number of clusters. $A(m, n) = 1$ if $m > n$ and there is a can't-link constraint between clusters m and n, and $A(m, n) = 0$ otherwise. Before merging two clusters m and n, we check that $A(m, n) = 0$ ($m > n$). In this case, we assign cluster n to have identity m. We update A by setting row m equal to the OR of rows m and n, and removing row and column n. This update takes $O(N)$ time, and can occur in $O(N)$ iterations. So enforcing the can't-link

constraints adds $O(N^2)$ time to Kruskal's algorithm, which still runs in $O(N^2 \log N)$ time.

If we run this variation on Kruskal's algorithm for $O(N^2)$ pairs, the complexity of choosing each question will be $O(N^4 \log N)$. Even with moderate N (say N=100) we cannot ask $O(N)$ questions with this much cost for each question. So we will propose a much less computationally complex version of Active-HACC in Section 3.2.4.

In part, this complexity helps motivate our use of a very simple clustering algorithm such as HACC. Since we must simulate $O(N^2)$ possibilities for each question, it is important that the clustering algorithm be relatively cheap. Moreover, as we will see, HACC lends itself to incremental clustering, in which simulating the effects of one constraint can be done efficiently. At the same time, HACC is quite interesting because the addition of one constraint can in some cases significantly alter the clustering.

3.2.4 FAST-Active-HACC

Our previous analysis assumes that we rerun HACC $O(N^2)$ times to simulate the effects of every question we consider asking. This is wasteful, since most new constraints only have a small effect on the clustering. We save a good deal of effort by incrementally computing these changes starting from the existing clustering. However, these changes can be somewhat complex. When a can't-link constraint is added to points in the same cluster, we must remove an edge from the current MSF, to disconnect these points. Removing one edge can have a domino effect, since there may be other edges that would have been added if that edge had not been present. Similarly, adding a must-link constraint might

require us to merge two clusters that have can't-link constraints between them, requiring us to remove edges to separate any points connected by a can't-link constraint. We must simulate any effects of these changes.

Our complete algorithm is quite complex, and is described below. Here we provide a couple of key intuitions. First, we save a good deal of time by creating data structures once that can be used in $O(N^2)$ simulations. Whenever we add a can't-link constraint between two points in a cluster, we must remove the last edge added to the MSF that is on the path between these points. To facilitate this, we keep track of the path on the MSF between all pairs of points in the same cluster. Also, as we perform the initial clustering, whenever an edge is not added to the MSF because of a can't-link constraint or because it would destroy the tree structure, we keep track of any edge which blocks it. That is, edge B blocks edge A if edge A would be added if not for the presence of edge B. Then, whenever an edge is removed, we can reconsider adding any edge that was blocked by this edge. Of course, as we "go back in time" and make changes to the MSF, we must carefully account for any later decisions that might also change.

In a second optimization, we notice that we do not need to simulate the effects of all Can't-Link constraints. If a cluster has N_c elements, there are $\binom{N_c}{2}$ possible Can't-Link constraints, but only N_c possible edges that can be removed. This means that many can't-link constraints will cause us to remove the same edge from the MSF, and have the same effect on the clustering. These can be simulated together.

Since this overall procedure is complex, code has been made publicly available.

3.2.4.1 Definitions

We are given a set of points and pairwise distances. As we mentioned earlier, we consider a complete graph $G = (V, E)$ where each pair of distinct nodes are connected by a unique edge and each edge has a weight given by the distance matrix. We are computing a minimum spanning forest for G. To do this, we sort the edges, E by distance, place them in a priority queue, and pop them off the queue in turn. As we progress, we can divide E into four disjoint groups.

Active Edges (AE): This is the set of edges that have been added to the MSF. The number of active edges may vary from $(N - K - M)$ to $(N - K)$ (M is the number of must link constraints).

Cannot Link Edges (CLE) : CLE is the set of edges that have been popped from the queue but have not been added to the MSF due to cannot link restrictions. We should not confuse a Cannot Link Edge with a Cannot Link Constraint between two points.

Redundant Edges (RE): An edge is included in set RE if it was not added to the MSF because the pair of points connected by this edge were already in the same cluster. Since we want to maintain a forest, we avoid all cycles.

Other Edges (OE): We call all the edges in E which are still on the queue, other edges.

We also put a time stamp, which is the index of an element in the sorted list, on each edge. An edge with lower weight will have a lower time stamp than an edge with higher weight. In Figure 3, it is explained how we build the sorted list and apply time stamps. We use the abbreviations AE, CLE, RE and OE as both set and as an element of a set.

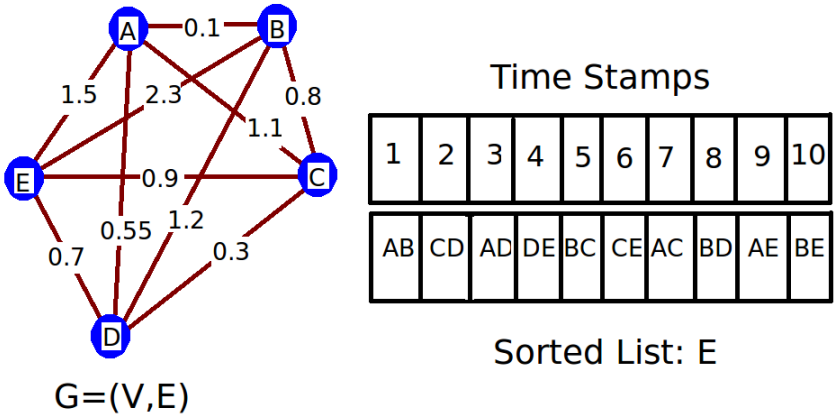
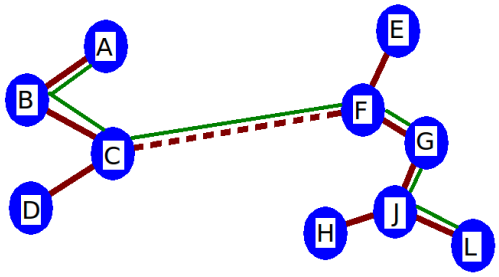


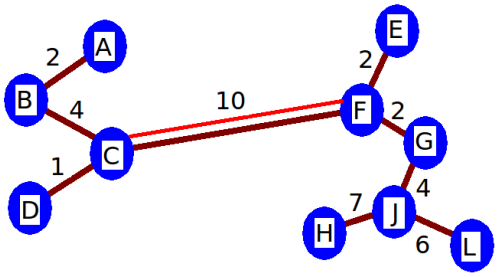
Figure 3: Complete graph G=(V,E) with edge weights on the edges and a sorted edge list.

3.2.4.2 Data Structures

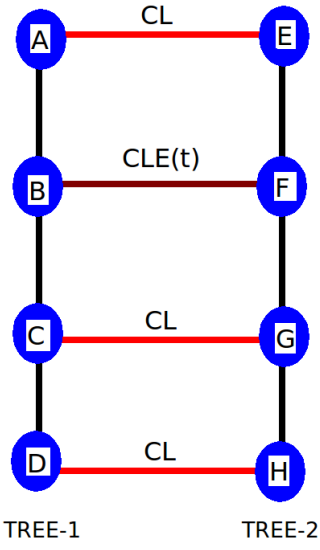
In this section we discuss some data structures required for the faster version of Active-HACC. The basic intuition is as follows. When we simulate adding a cannot link edge to the same tree in an MSF, we must remove some edge, to break the tree in two and separate the tree into two components that each contain one of the nodes in the cannot link edge. We must then simulate HACC's behavior from the point at which the removed link had



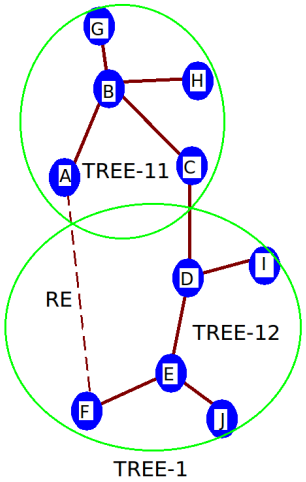
(a) When we add an edge between two clusters we update Store Simple path with paths between all nodes in the two trees.



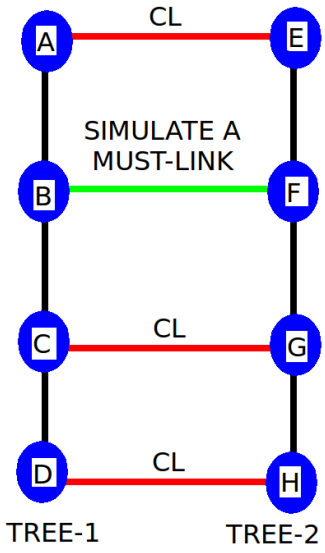
(b) Simulation of SCC.



(c) An example that shows how CLEs are dependent on active edges (see text).



(d) An example that shows how REs are dependent on active edges (see text).



(e) Simulation of DCM.

Figure 4: Simple examples for explaining the faster version of our algorithm.

been initially added. Similarly, if we simulate adding a must link edge that connects two trees that contain cannot link edges, we must join these two trees, and then remove edges to break this new tree apart, so that again no tree contains a cannot link edge. In either event, we must undo some edge or edges that were added to the MSF, and determine which subsequent decisions might be altered by this change. To make this efficient, we maintain data structures that keep track of the dependencies between various decisions we have made. It is important to note that these data structures need to be constructed only once, for all $O(N^2)$ questions whose answers we simulate. For each question, we only need to simulate changes to the data structures that can occur in simulating the changes to the cluster that this question induces; these changes are typically not large.

Store simple path between all pairs of points: Store-simple-path is a structure that stores the *simple path* between any pair of points in the same tree in the forest. A simple path in a graph is a path containing edges, where no vertices repeat. In a tree a simple path between any two points is unique. We update Store-simple-path whenever we add an edge while running HACC. For example in Figure 4a when we add an edge between trees 1 and 2 (i.e, add an edge between C and F), we can store the simple paths between any pair of points in $T_1 \times T_2$, where $T_1 = \{A, B, C, D\}$ and $T_2 = \{E, F, G, H, J, L\}$. For example, the simple path between A and L is $Store - simple - path(A, L) = Store - simple - path(A, C) \cup Store - simple - path(F, L) \cup (C, F)$. Store-simple-path should already have stored the simple paths between A and C, and F and L.

Active edge and Cannot Link edge dependency: AEs and CLEs are mutually dependent. While we build a MSF, a CLE is not added because some set of active edges,

with lower time stamps, were already present in the forest. If one or more of those active edges should be removed there is a possibility that a CLE can be added back to the MSF, without violating any constraints. In FAST-Active-HACC, when we simulate a new constraint, we look at which AEs must be removed to incorporate a simulated constraint and as a result which CLEs can be added to the forest. But these additions and removals of edges should be consistent with the given set of constraints including the one we are simulating.

We consider an example scenario in Figure 4c to understand the modeling of dependency between AEs and CLEs. Let us assume we are building a MSF with edges from a sorted edge list E. We have added some edges to the forest. The next edge to be considered from the list for addition to MSF is an edge between B and F. B and F are part of trees TREE-1 and TREE-2 respectively. Although there is no cannot link constraint between B and F there are other cannot links between AE, CG and DH. So we do not add edge BF to MSF and include BF in set CLE. Say BF has a time stamp t. Now, BF was not added because a subset of active edges $\{AB, BC, CD, EF, FG, GH\}$ were added before time t. We create lists that will contain those active edges. Since there are three cannot links between trees TREE-1 and TREE-2, we have three lists for BF corresponding to each of the cannot links; we call them $LIST_{AE}^{BF}$, $LIST_{CG}^{BF}$, $LIST_{DH}^{BF}$. In $LIST_{AE}^{BF}$, we have $\{BC, CD, FG, GH\}$. We also have to keep track of the multiplicity of dependency, e.g, edges BC and FG are present in both $LIST_{CG}^{BF}$ and $LIST_{DH}^{BF}$. So BC and FG have multiplicity of two. If we remove any one of BC or FG, we can delete two lists together. That implies that if either of BC or FG was not added when we built the MSF, we would not have had the cannot link restrictions due to cannot links between CG or DH. Now let

us assume we remove active edge BC. So we can get rid of two lists. Still there is no chance we can add BF and make it active unless either AB or EF is inactive, because the cannot link constraint AE will prevent addition of BF. If either of those edges become inactive we can also delete the list $LIST_{AE}^{BF}$ and can add BF to the MSF.

So for all CLEs we keep lists of active edges and their multiplicity. Whenever we delete one active edge we see which of the lists can be deleted. If a cannot link edge is free from all lists it can be considered for addition to the MSF. However if a CLE itself is also a cannot link constraint we can not ever add that. Above subsection describes how removal of an active edge makes way for other CLEs to be added.

Redundant edge and Cannot Link edge dependency: A redundant edge is not added to the MSF if the points connected by the RE is already connected by a set of active edges. So we also note which of the active edges are responsible for not adding an RE.

Let us take an example in Figure 4d, where we have a set of active edges already added to the MSF. Now if AF is the next edge from the sorted list E to be considered, we cannot add that edge as it will create a cycle. So AF is included in RE. Now when we simulate constraints if one of the active edges from $\{AB, BC, CD, DE, EF\}$ is removed, can we add AF to MSF? No, if all other edges between A and F in TREE-1 remain active. Let us assume active edge CD is removed to satisfy some cannot link constraint. Now we have two subtrees TREE-11 and TREE-12. If redundant edge EF is added in this scenario, the constraint for which CD was removed would be violated again. But if any other active edges from $\{AB, BC, DE, EF\}$ is removed AF comes under consideration for possible addition to MSF. Now we may wonder what happens if any other edge from

$\{BG, BH, DI, EJ\}$ is removed. If that gets rid of the constraint for which CD was removed, we would add CD back and will not add AF, because AF has a higher time stamp than CD. So we don't worry about any other edge other than the active edges on the simple path between A and F. If two or more active edges from that path are removed we start considering AF. So we keep a separate list of active edges for all redundant edges and see if two or more edges are removed from the list.

Now we have discussed how removal of active edges bring CLEs and REs into consideration. But when we add an edge to the MSF from CLE or RE, how does this change active edges? At a high level, whenever we are about to add a CLE or RE, we remove all active edges, with a later time stamp than that of the CLE/RE, from the two trees connected by that CLE/RE. However we reconsider them again for addition to the MSF. This is pretty fast in practice. The main reason is, addition or removal of edges is very local, so in reality we do not need to remove or consider adding huge number of edges.

Priority Queue (PQ): This is the most important structure we use while we simulate a SCC or a DCM. In this queue we store possible edges to be added in the MSF but sorted based on their time stamps. But there are some additional properties of the elements in this queue and we describe them below:

- Edges in the priority queue must not be present in the MSF.
- The MSF should always be consistent. That means it can have more or less than K clusters but all the must link and cannot link constraints must be satisfied.
- Once we start popping elements from the priority queue, if we have considered

adding an edge at time t, we never look at an edge with time stamp $T \leq t$. That implies any other edge with $T \leq t$ can not be added to the priority queue from now on.

In our framework we keep a priority queue with a set of edges, that can possibly be added to the MSF. We want to minimize the total number of elements(say $|PQ|$) added to PQ. The minimum set is the set of edges, which we can definitely add to MSF. Although it is easier to find which of the CLE/REs are affected when we remove an active edge, tracking which of active edges could be affected while we add a CLE/RE is complicated. It is hard to find a polynomial time precomputation process for this part, which can be computed once and used for all possible question simulations. The precomputation in that case involves taking care of all possible cycles of a complete graph, which could take exponential time. So we have a simple work around, that works well in practice. We try to minimize $|PQ|$ as much as possible but we do not try to get the exact minimum of $|PQ|$ using our algorithm. We make sure the algorithm is correct, but we may do some extra computation.

3.2.4.3 Primitive Operations

These are some set of basic operations which we perform while we simulate the same cluster cannot-links or different cluster must-links.

- **Insert with priority**: Add an edge to the priority queue with the priority based on time.

- **Pop highest priority element**: Pop out an edge from the priority queue which has the lowest time stamp.

- **Add an edge to the MSF**: Consider an edge to add to the MSF such that all constraints are satisfied and if added, relabel necessary nodes.

- **Remove an edge from MSF**: Remove an edge and relabel nodes.

3.2.4.4 Same Cluster Cannot Links (SCC)

In this section, we consider an additional optimization that speeds up the simulation of adding a cannot link edge between two nodes from the same cluster. In fact, we do not need to consider possible pairs within the same cluster separately. It turns out that many possible cannot link edges, if added, will have exactly the same effect. If we have n points within a cluster we should simulate the results of cannot link edges between $\binom{n}{2}$ possible pairs. But in reality we have to simulate only (n-1) pairs. Let us take an example in Figure 4b to explain this. The edge weights are shown on the edges. Let us assume we want to simulate the effects of a cannot link constraint between D and E. Now if there is a CL between D and E, all the active edges would have been added except CF, because CF is the highest weighted edge. When we would have tried to add CF, the cannot link constraint between D and E would stop addition of CF. Now let us assume we want to simulate a cannot link constraint between A and L. Again we would not have added the edge CF because it has the maximum weight on a path connecting A and L. So in both of these cases we have to remove CF to simulate a cannot link constraint. Now we can observe that if we want to simulate a cannot link constraint between any pair in

$\{A, B, C, D\}$ (T_1) $\times$ $\{E, F, G, H, J, L\}$ (T_2), we have to remove CF, because this is the maximum weighted edge on any simple path between a pair of points from T_1 and T_2 . For all of those pairs we would ultimately not add edge CF. So we can proceed like this: We find the maximum weighted edge (say E_M in a tree, find the subtrees (say T_1 and T_2) connected by this edge. For all possible pairs in $T_1 \times T_2$, we would remove edge E_M. Now we can proceed to the next maximum weighted edge and do the same thing, but we don't update simulation of cannot link constraints for pairs that we have already considered. We continue doing this till we reach the least weighted edge. In this way, by simulating the effects of removing n possible edges, we determine the effects of every possible cannot link edge that could be added to this cluster.

While we simulate a SCC or removal of an edge, we first remove that edge (say time stamp of this edge is t_M) and then run a module QUICK-SIMULATE to complete the simulation of SCC. We will provide a brief high level idea and detailed pseudo code of this part later. We will use the same module also in DCM simulation.

3.2.4.5 Different Cluster Must Links (DCM)

When we simulate the must links between different clusters finding the set of active edges to be removed is harder. However we have precomputed some data before the start of the simulation to make things easier. We will refer to Figure 4e to explain which of the active edges should be removed to add a DCM. Let us assume we would like to add an ML between B and F. There are three CLs between TREE-1 and TREE-2. So if the cannot

link constraint AE and simulated must link constraint BF both have to be satisfied one of the active edges on each simple path (in this particular example each simple path is just an edge) AB or EF must be removed. The higher weight edge will be removed for the same reasons we mentioned in our discussion of SCC. So while we simulate a DCM, we find the maximum weighted edges on the union of simple paths between the cannot link ends and simulated must link ends. So if there are three cannot links between TREE-1 and TREE-2 we will have three such paths and we have to find the maximum weighted edge in each path. Since we already have the simple paths calculated between any pair of points in a tree, this process is fast. In our example three paths are, $\{AB, EF\}$, $\{BC, FG\}$ and $\{BC, CD, EG, GH\}$. The set of edges to be removed are $RS = \{BC, CD, EF\}$. Now we start removing edges with the minimum time stamped edge being removed first. So we remove BC first. We also see the fact that BC has multiplicity two and if we remove BC we do not need to remove CD. Then we can remove EF. Also assume the minimum time stamp of RS is t_M. Unlike SCC, here we will get a set of edges to be removed that can contain more than one element. Now we would run the module QUICK-SIMULATE to complete the simulation of DCM.

3.2.4.6 QUICK-SIMULATE

This is a module which is used both for SCC and DCM. Once we remove the set of required edges, we can run this module. Now whatever happened with HACC before time stamp t_M will remain the same in the simulation. We know if an active edge is removed, which CLE/REs come under consideration. We insert them into the priority

queue. Now we pop the first element (say time stamp t) from the priority queue and try to add that into the forest. We should remember again any edge with time stamp $T \leq t$, will not be included in priority queue in future. Now, if the popped edge is a CLE or RE with time stamp t, we remove all the active edges (with time stamp $T \geq t$) in the two subtrees, which are going to be connected by that CLE/RE and put them into the priority queue. We also see if we can add any other CLE/RE due to removal of these edges. One of our observations is that if we don't remove the active edges with $T \geq t$, sometimes we may end up with a different clustering than HACC would obtain. As we mentioned earlier this does not take lot of time in real experiments because we remove a few edges very locally and the total number of edges to be removed in this process is very small. We continue doing this until we do not have any element in the priority queue.

Then, if we have more clusters than required, we pull the minimum time stamped element from OE and add that to the forest. If we have less than the required number of clusters we remove the maximum weighted edge from the forest.

We get the clustering but we are not done yet. To complete the simulation, we need to compute the change in the Relative Jaccard's coefficient. Calculation of Jaccard's coefficient is complex, i.e, we have to do $O(N^2)$ work. But we can easily keep track of which of the points change assignments or move from one cluster to another. We use that information to calculate Jaccard's coefficient and we have empirically seen on an average it does not take more than $O(N)$ time.

Once we have found the effect of simulation of all possible constraints, we select the pair with maximum expected change in the Relative Jaccard's Coefficient. We present

Algorithm 2 QUICK-SIMULATE

Given: Priority-Queue with edges to be removed
while $Priority - Queue \neq \phi$ **do**
  Pop the first element e from Priority-Queue
  e connects clusters C_1 and C_2 and has time stamp t
  if e is in CLE or RE **then**
    Remove all active edges with time stamp $T \geq t$ from C_1 and C_2
    Insert them into Priority-Queue
  end if
end while
If required remove the last added edge or add more edges from OE to maintain K clusters

that pair to some user, get feedback and continue doing this until our human budget is exhausted or we get reasonable clustering. Since FAST-Active-HACC is complex, code for this has been made publicly available for use at `http://www.umiacs.umd.edu/~arijit/projects/Active_clustering/active_clustering`.

3.3 Experimental Results

We have experimented in three different domains, leaf, face and scene images. For leaves, we create three subsets from a huge corpora of leaf images used for Leafsnap [48]. All leaf images are iPhone images of leaves on a white background. The leaf subsets are called Leaf-100 (containing 100 images from 10 different species), Leaf-250 (250 images from 25 different species) and Leaf-1042 (1042 images from 62 species). Leaf-100 and Leaf-250 have the same number of leaves from all species. But in Leaf-1042, the number of leaves in each species vary from 4 to 31. Similarly for faces, we have extracted three subsets of images from a face dataset called PubFig [49]. The PubFig database is a large,

real-world face dataset consisting of $58,797$ images of 200 people collected from the Internet. Unlike most other existing face datasets, these images are taken in completely uncontrolled situations with non-cooperative subjects. Thus there is large variation in pose, lighting, expression, scene, camera, and imaging conditions, etc.... The subsets of images are called Face-100 (100 images from 10 different people), Face-250 (250 images from 25 different people) and Face-500 (500 images from 50 different people). All of these face datasets have the same number of images in each class. For scenes, we have created a dataset called Scene-300 which has 300 images from 30 classes with each class having 10 images. Scene-300 is a subset of the Indoor Scene Recognition dataset [63]. This dataset has images from different kinds of places such as airports, hospitals, libraries, kitchens etc.... We first run experiments with perfect human responses, and then run a second experiment to evaluate the effect of real human responses.

The distance matrix for face images and leaf images are calculated based on algorithms described in [9] and [48] respectively. For scenes, 512 dimensional gist features are extracted from each image and Euclidean distances are calculated between all pairs of images to create the distance matrix. Once we get the distance matrix, we can run our proposed algorithm on all these datasets.

As described in Section 3.2.1 we have used a couple of heuristics named H1 and H2 to further speed up our algorithm. In H1, if the pairwise probability measure indicates that human response is going to be very similar to the current clustering, we do not bother to simulate a different response. In our experiments, if the probability that an image pair are from different classes is more than 0.9 and they are actually in different clusters, we do not simulate a must-link constraint for that pair. For larger datasets of size more than 1000

we initially use K-means [56] to group very close points together. For example for Leaf-1042, we cluster 1042 images to 700 clusters and then run our algorithm with centroids of those 700 clusters. In H2, instead of computing the change in overall clustering, we only compute local changes involving the image pair under consideration. Often when a new constraint is added, the effects are very local and involves clusters which contain the image pair. We can determine these changes much faster than the overall change in clustering.

The main objective of running experiments on smaller datasets of size 100 and 250 is to show that even if we use heuristics the performance of the algorithm is not affected that much. The algorithm without heuristics is too slow to run on larger datasets. For example we run our algorithm on Face/Leaf-100 without any heuristic, with only H1 and then with H1 and H2 both. For Face/Leaf-250 we run our experiment with only H1 and then H1 and H2 both. For Face-500/Leaf-1042/Scene-300, we run a set of experiments using both heuristics H1 and H2.

3.3.1 Empirical Observations

We have run our algorithm on all of the datasets described above. All the algorithms to which we compare are described in Table 3.1. Figures 5 and 6 (where Jaccard's Coefficient corresponding to one misclassification per cluster is displayed using green squares) show performance evaluations of all the algorithms on Leaf-100, Face-100, Leaf-250, Face-250, Leaf-1042, Face-500 and Scene-300. We run K-means clustering 100 ($S = 100$) times on each dataset with different random initial set of centroids to find the pairwise probability distributions. We see how Jaccard's Coefficient changes with the

Table 3.1: Summary of the algorithms that we compare.

Active-HACC	Proposed active learning version for Image Clustering
FAST-Active-HACC	Faster version of our proposed algorithm without any heuristic
FAST-Active-HACC-H1	Faster version of our proposed algorithm with H1 only
FAST-Active-HACC-H1_H2	Faster version of our proposed algorithm with H1 and H2 both
Random Constraints	Seek pairwise constraints randomly and use HACC
K-means without questions [56]	Simple K-means without any human intervention
CAC1 [13]	A heuristic to find the best pair
Active-PCKMeans [6]	An active constrained clustering algorithm
COP-Kmeans [74]	A semi-supervised clustering algorithm (with pairwise constraints)
[78]	A spectral active clustering approach based on k nearest neighbor graph purification

number of questions. Since we have the ground truth for these datasets we were able to calculate the Jaccard's Coefficient with respect to the ground truth. In real world situations we will not have the ground truth and will have to decide when we have reached a good clustering. One possible way to stop asking questions is when clustering assignments do not change even with additional constraints. Also, one of the advantages of FAST-Active-HACC is that we do not need to set any parameters. One of our main interests is in problems that grow big because the number of clusters becomes large. We make the following observations from the experiments:

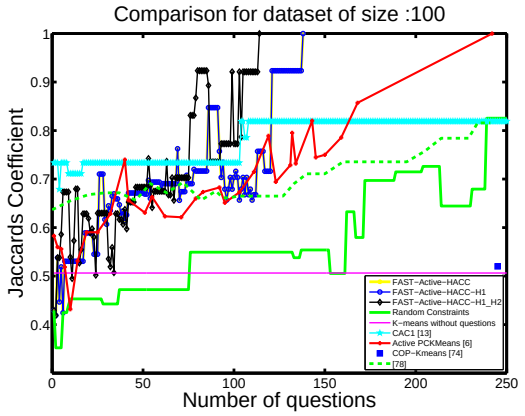
- In all of these experiments our algorithm significantly outperforms all other algorithms. We were able to greatly reduce the number of constraints required for a reasonable clustering.
- We run both Active-HACC and FAST-Active-HACC for the Leaf-100 and Face-100 datasets. We see that FAST-Active-HACC is $25-30$ times faster than Active-HACC for a dataset of size 100. Overall we expect FAST-Active-HACC to be $O(N)$ faster

than Active-HACC. Since Active-HACC is slow, we could not experiment with it in larger datasets. We also observe that FAST-Active-HACC-H1 produces the exact same results as FAST-Active-HACC for Leaf-100/Face-100. For a dataset of size 1042, FAST-Active-HACC-H1_H2 takes around a minute per question. We believe this could be further sped up with more optimized code (our current implementation is in MATLAB) or parallel processing as our algorithm is highly parallelizable.

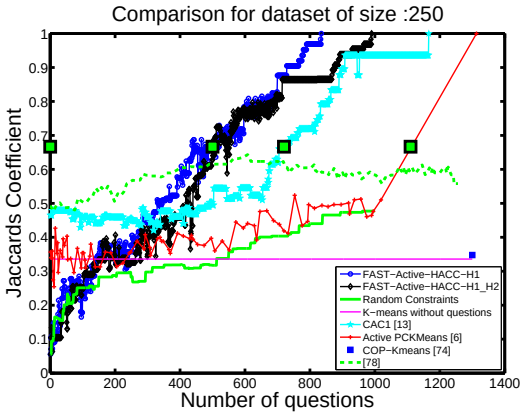
- In Figure 5c, we compare the results for different algorithms for the Leaf-1042 dataset. For this dataset only we use K-means initially as part of H1, to reduce the number of points to 700. Even with that, we get Jaccard's Coefficient of 0.8152 (one misclassification per cluster on average) by asking just 3782 questions.
- We wanted to compare our algorithm with [31] and [38], but a direct comparison on our image datasets is not possible due to the complexity of their algorithm and the lack of publicly available code. However we also run experiments using the iris dataset to compare with [31], on which they have reported results. This is a relatively easy dataset with 150 points from 3 clusters in 4 dimensions. They have used the ratio of well categorized points to the total number of points as an evaluation metric (let us call it RWCP). Our algorithm reaches RWCP of 0.97 within three questions, where they take ten questions to reach the same RWCP.
- One of the major differences in these domains is the distance matrix. The leaf image distances are more accurate than the face images. Also the face image distances are more accurate than the scene images. So even if we have three similar sized datasets from three different domains, we need different amount of human interaction in each of these domains.

3.3.2 Experiments with Real Humans

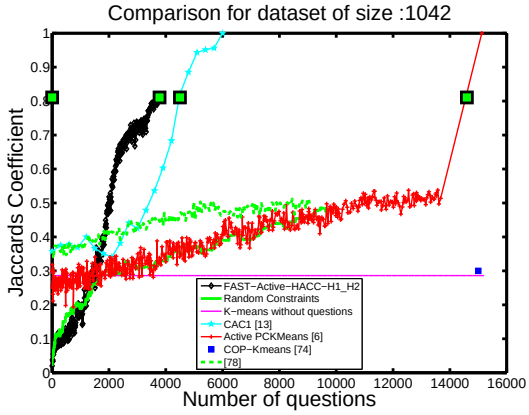
In previously described experiments we assumed that humans are always correct when they provide pairwise constraints. We note that all previous active clustering algorithms [6, 37, 38, 57, 79] have explicitly made this assumption. Authors in [49] demonstrate that this assumption is reasonable for face images, as humans are 99.2% accurate in judging similarity of two faces. However we wanted to verify how robust our algorithm can be when we relax this assumption and allow real humans to judge similarity of images using Amazon Mechanical Turk. To allow for extensive quantitative evaluations while still using feedback from real users, we collected exhaustive human feedback for all possible pairs of images in our datasets Leaf-250 and Face-250 using Mechanical Turk. We ask about each image pair to a mechanical turk user to decide if a pair of images belong to the same category or not. In general Mturk workers found this task to be fun and interesting; their comments were "nice","good", "again provide these types of work" etc.... However some workers provided random/noisy responses throughout a HIT, which we had to filter out. Overall we find that humans are wrong around 1.2% of the time in judging the similarity of face images and around 1.9% of the time in judging the similarity of leaf images. We plot the experimental results with real human input in Figure 7. We find that with real human feedback we never achieve perfect clustering (JCC=1) because of the errors introduced by humans. Also note that after asking some number of questions all algorithms' performance start to become worse. Our experiments were run until our proposed algorithm stopped improving. Determining a good stopping point automatically remains an interesting research problem. We find that our algorithm is still better than



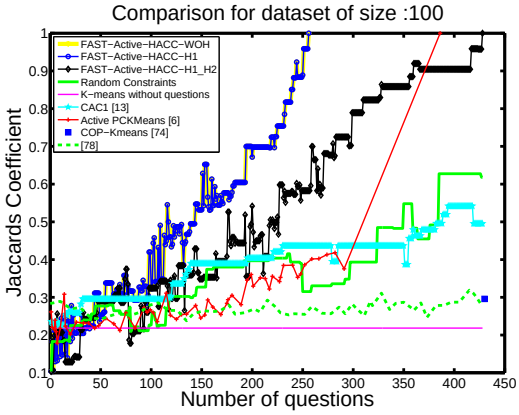
(a) Leaf-100 dataset



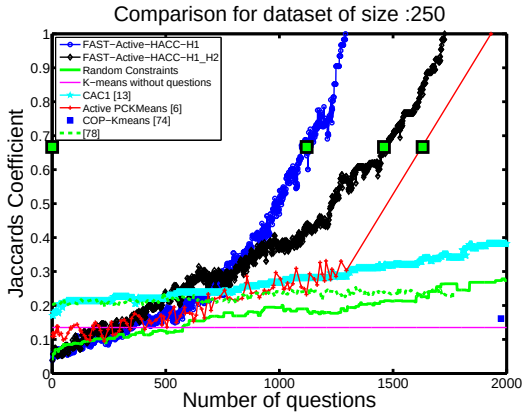
(b) Leaf-250 dataset



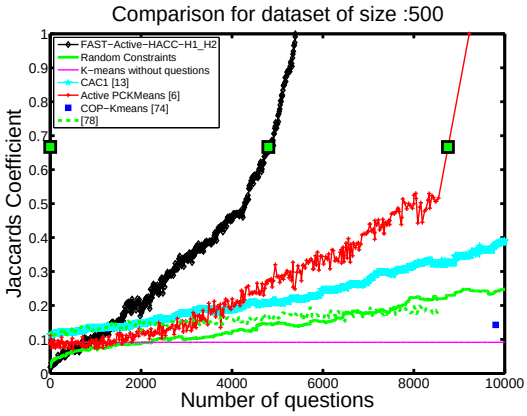
(c) Leaf-1042 dataset



(d) Face-100 dataset



(e) Face-250 dataset



(f) Face-500 dataset

Figure 5: Performance plot showing how the Jaccard's Coefficient increase with the number of questions. Our proposed algorithm Active-HACC significantly outperforms all other algorithms we compare.

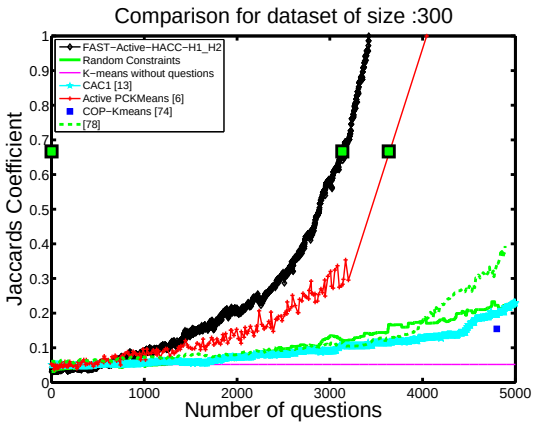
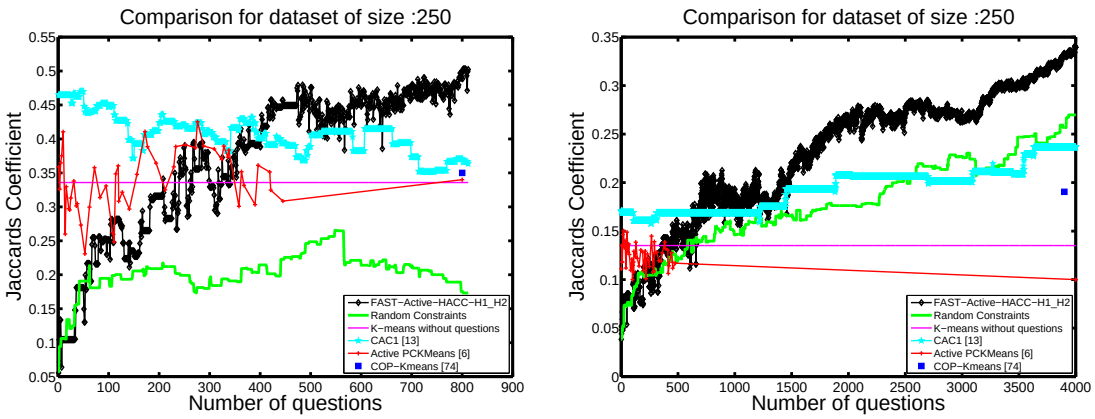


Figure 6: Scene-300 dataset

other active and passive algorithms. Although the heuristic proposed by [13] has a good starting point for Leaf-250, it's performance quickly becomes worse as real human input is introduced. However for Face-250, [13] does not perform well even in the beginning. Our proposed algorithm achieves maximum Jaccard's Coefficient (JCC) of 0.5 for Leaf-250 and 0.35 for Face-250. Note that JCC $= 0.5$ implies 20 clusters with 2 mistakes and 5 clusters with only one mistake on average (cluster size is 10). Similarly JCC $= 0.35$ implies 20 clusters with 3 mistakes and 5 clusters with 2 mistakes on average (cluster size is 10).

3.4 Conclusions

We have presented an approach for image clustering that incorporates pairwise constraints from humans. An algorithm is developed for choosing data pairs to use when querying a person for additional constraints. Since a brute force version of our algorithm is time consuming we also formulate a more complex but faster version in this chapter. Our



(a) Leaf-250 dataset

(b) Face-250 dataset

Figure 7: Performance plot showing how the Jaccard's Coefficient increases with the number of questions when we get pairwise constraints from real humans in Mechanical Turk.

algorithm outperforms all state-of-the-art results in image clustering. Although this work was focused on solving image clustering, this idea could be extended to any clustering domain in general.

Chapter 4: **Active Subclustering**

In this chapter we talk about clustering a subset of a dataset only. Although there are many excellent clustering algorithms, clustering images with many classes is a hard problem. We address these challenges in two ways. First, we propose a passive algorithm to cluster a subset of the data only (which we call subclustering). Subclustering will produce smaller but purer clusters. Second, we get pairwise constraints from humans to correct errors in a subclustering solution. We run experiments on a face image dataset (having $51,418$ images from 200 classes) and a leaf image dataset and show that our proposed algorithms perform better than baseline methods.

4.1 Related Work

As far as we know there is no previous work on the proposed subclustering approach. Perhaps the most similar work is Bregman Bubble Clustering proposed by Gupta *et al.* [33]. Their approach tries to find a robust and scalable clustering framework to find K dense clusters in the dataset, removing outliers. The authors in [80] propose a robust clustering algorithm that maximizes a total similarity objective function related to an approximate density shape estimation. However, these approaches show experimental results for a very

small number of clusters. We implemented the BBC-S algorithm [33] but modified to our subclustering framework for comparison. The authors in [69] proposed a face annotation framework that involves partial clustering and interactive labeling. The partial clustering approach tries to model images that are not grouped tightly enough, using a uniform background noise distribution. However it is often hard to model these images, with only one uniform background distribution, especially if there are many classes. Note that our experiments are with $51,418$ images from 200 clusters, where [69] uses only $1,147$ images from 34 clusters. The authors in [50] proposed a method for visual category discovery that focuses on the easiest examples first and then progressively expands its repertoire by including more complex objects. However this approach is initialized with a familiar set of category classifiers.

There are also a couple of recent works which try to enforce diversity in mixture models [61, 83], such that mixture components do not represent redundant information. The authors in [34, 81] proposed methods for discriminant analysis, where data from each class is represented by a mixture of Gaussians. These methods are useful when data from a single category belong to multiple clusters instead of a single cluster. Although these methods have shown to improve the classification results, finding the actual number of mixture components in each class still remains a difficult problem. Similar to these methods our approach also assumes that data from different categories do not form well-separated clusters, such that any single cluster will contain all the data points from a single category. However we note that our approach is developed for clustering problems, where no class labels are available. In subclustering instead of representing data from each category with multiple clusters, we find one small but dense group of points from

each category.

Since we have already discussed the past work on active learning for computer vision and clustering in detail in the last chapter (Chapter 3) we do not repeat the discussion here.

4.2 Passive Subclustering

Standard clustering algorithms usually perform a complete partition of the dataset. However, state of the art image features do not always map images to a space in which images from the same class remain close to each other and far from images from other classes. Consequently, standard clustering approaches [44, 56, 66], which optimize a cost function that takes into account all images, often end up with many impure clusters. In subclustering the cost function takes into account only a few examples from each class.

We formulate subclustering as a problem of maximizing the probability that all images in a cluster come from the same class, and that every pair of cluster centroids comes from a different class. Intuitively, this means that we want cluster centroids to be as far from each other as possible, while images assigned to each centroid should be as close to the centroid as possible in the feature space.

4.2.1 Notation

We now introduce some notation. We are given a set of N unlabeled images $\mathcal{U}$ to be clustered into K clusters. N_k denotes the number of images in the k-th cluster ($\sum_{k=1}^{K} N_k = N$). T denotes the number of iterations an algorithm is run. We want to produce sub-

clusters of size n (provided by the user) and we assume $n \leq N_k$ for any k. $\mathcal{C}$ ($c \in \mathcal{C}$) represents the set of K cluster centroids and $\mathcal{C} \subset \mathcal{U}$, i.e., centroids are members of the dataset. $\mathcal{R}$ ($|\mathcal{R}| = R$) denotes a restricted set of images (randomly chosen from $\mathcal{U}$), which is larger than the number of clusters but smaller than the total number of images ($\mathcal{R} \subset \mathcal{U}$ and $K < R < N$). $P(I_i = I_j)$ denotes the probability that images I_i and I_j are from the same class ($I_i, I_j \in \mathcal{U}$).

In all our experiments we assume that we have access to pairwise distances between any pair of images. Any kind of image features and distance measures can be used to find a pairwise distance. We convert each pairwise distance to a pairwise probability by using a simple exponential function $P(I_i = I_j) = exp(-d(I_i, I_j)/\lambda)$ [66], where $d(I_i, I_j)$ denotes the distance between images I_i and I_j and λ is a positive scaling parameter.

4.2.2 Our Algorithm

In subclustering we want to choose a set of centroids $\hat{\mathcal{C}}$ such that the following product of probabilities is maximized:

$$\mathbf{P} = \underbrace{\prod_{k=1}^{K} \prod_{j=2}^{n} P(c_k = I_k^j)}_{P_1} \times \underbrace{\prod_{\substack{k_1,k_2 \\ k_1 \neq k_2}} (1 - P(c_{k_1} = c_{k_2}))}_{P_2} \tag{4.1}$$

where I_k^j denotes the j-th image assigned to cluster k. Note that the first image assigned to the k-th cluster is c_k itself. The first part (P_1) ensures that $n-1$ images $\{I_k^j\}$ assigned to centroid c_k should be close to c_k. The second part (P_2) makes sure that the probability of dissimilarity between all pairs of centroids in $\hat{\mathcal{C}}$ should be as high as possible.

Next we convert the maximization problem to a minimization problem by taking the negative log of (4.1), i.e., $\mathbf{Cost} = -log(\mathbf{P})$ is:

$$-\sum_{k=1}^{K}\sum_{j=2}^{n}\log P(c_k = I_k^j) - \sum_{\substack{k_1,k_2 \\ k_1 \neq k_2}} \log(1 - P(c_{k_1} = c_{k_2})) \tag{4.2}$$

The optimal centroid set is $\hat{\mathcal{C}} = \arg\min_{\mathcal{C}} \mathbf{Cost}$.

The cost function in equation 4.1 is formulated in a way such that images in the same subcluster are tightly grouped and are far from the images in other subclusters. Ideally we want to extract features from images such that images from the same category stay close to each other in the feature space and images from different categories are far from each other. However state-of-the-art computer vision image features are not accurate enough that all images will remain in the feature space in this way. However we expect that features from some of the images will follow the ideal structure in the feature space. Our cost function for subclustering is formulated in a way such that if optimized will output only those images. Although there are other possible ways to formulate cost functions to perform subclustering, they may not be as good as our approach. For example, one such possible cost function is the traditional complete clustering cost, where we assign each point to the nearest cluster center and then minimize the average distance between the images and the centroid in each cluster. From the complete clustering solution, we can obtain n images (nearest to the center) from each cluster to get a subclustering output. We note that although such an approach can be used to create subclusters from complete clusters, cost functions solely built to solve subclustering create purer subclusters. Intuitively it is clear that points that are difficult to cluster can affect the results of clustering

algorithms in a way that degrades the clustering of easier points. That is, it makes more sense to solve the problem we want to solve directly (subclustering), rather than trying to solve a harder problem (clustering) as an intermediate step. This is demonstrated in the experimental results in Section 4.6.

For each cluster centroid we have to find $n-1$ images $\{I_k^j\}$ to be assigned to centroid c_k. This is achieved by simply assigning the $n-1$ closest images to c_k in cluster k. Since this can lead to overlapping subclusters we add an extra step in our algorithm to minimize the overlap between clusters, discussed later in this section. We denote a set of $n-1$ images assigned to a centroid c along with c itself as $\mathcal{S}_c$ (we use c_k or c both as elements of $\mathcal{C}$, c_k additionally implies that it is the k-th centroid).

We optimize $\mathbf{Cost}$ with an iterative algorithm. We initialize our algorithm with random cluster centers, assigning the $n-1$ closest images to each of these cluster centers. Then, at each iteration, we swap one of these cluster centers with a new cluster center, chosen from a pool of possible replacements in the restricted set, $\mathcal{R}$, choosing the replacement that most reduces $\mathbf{Cost}$. Now we explain our algorithm in detail:

(a) First, we precompute the change in the first part of $\mathbf{Cost}$ if any $r \in \mathcal{R}$ is added to or removed from a centroid set. We can precompute that as we know which images will be assigned to each centroid due to our simple assignment procedure. We take advantage of this fact and significantly cut down computational cost.

(b) Next, we randomly choose an initial centroid set $\mathcal{C} \subset \mathcal{R}$. We maintain an additional vector of size R, which stores the changes in the second part of $\mathbf{Cost}$ if a centroid $r \in \mathcal{R}$ is added to the current centroid set $\mathcal{C}$. This vector needs to be updated at the

start of each iteration.

(c) We calculate the present subclustering cost using (4.2) and denote that as $\mathbf{Cost}_t$ at the t-th iteration.

(d) We consider replacing each centroid, $c \in \mathcal{C}$, with each centroid from the restricted set, $r \in \mathcal{R}$.

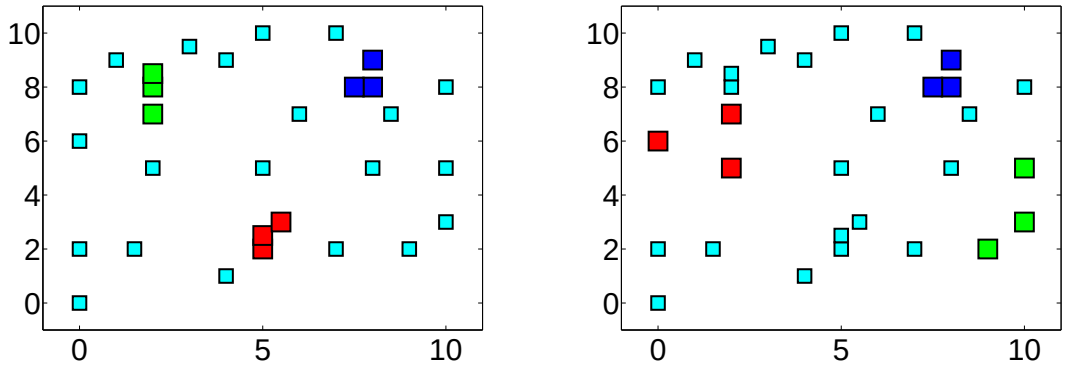
(i) A replacement of c with r is valid only if $\mathcal{S}_r$ has no more overlap with existing subclusters than $\mathcal{S}_c$ does. We say there is an overlap between two subclusters if one or more images in those two subclusters are the same. We define an indicator function $\mathbf{1}(\mathcal{S}_{c_i}, \mathcal{S}_{c_j})$, which is 1 if there is an overlap between subclusters $\mathcal{S}_{c_i}$ and $\mathcal{S}_{c_j}$ and 0 otherwise. A valid centroid replacement must follow: $\sum_{j=1}^{K-1} \mathbf{1}(\mathcal{S}_c, \mathcal{S}_{c_j}) \geq \sum_{j=1}^{K-1} \mathbf{1}(\mathcal{S}_r, \mathcal{S}_{c_j})$, where $c_j \in (\mathcal{C} - c)$.

(ii) We calculate the changed cost for all of these valid replacements ($\mathbf{Cost}_{t+1}(r, c)$) and see if any of them is less than the present cost $\mathbf{Cost}_t$. If yes, we make the replacement for which the decrease in cost is maximum, i.e., replace the corresponding centroid $\hat{c}$ with $\hat{r}$ in $\mathcal{C}$.

(e) Go to step (c) until the algorithm converges, i.e., we do not find any replacement which decreases $\mathbf{Cost}_t$. The centroid set at the final iteration along with the $n-1$ closest images to each centroid is the output of our subclustering algorithm.

Optimizing the subclustering cost function in equation 4.2 is a NP-hard problem[1] and obtaining a global solution is not possible in polynomial time. We use an iterative algorithm that can find a locally optimal solution in polynomial time. We note that the it-

[1]This is similar to a subset selection problem and they are known to be NP-hard.



(a) Our Subclustering Algorithm (b) K-medoids for Subclustering

Figure 1: Comparison of our passive subclustering algorithm with K-medoids (medoid and closest 2 points from the medoid are displayed in each cluster) in a small scale 2-d clustering problem. We see that subclusters produced by our proposed algorithm are more meaningful (30 points from 3 clusters with subcluster size of 3).

erative optimization method is motivated by the optimization used in Partition Around Around Medoid clustering algorithm [44]. Also note that our cost function is lower bounded by zero and at each iteration of our algorithm cost either decreases or remains the same, thus our proposed algorithm is guaranteed to converge.

We have made the code for this algorithm available at `http://www.umiacs.umd.edu/~arijit/projects/Active_subclustering/active_subclustering`. We demonstrate the proposed subclustering algorithm's performance on a simple 2-d example in Figure 1.

Complexity: The total computational cost of our proposed algorithm is $O(RN + KRT)$ (see Appendix 6.2 for proof), where K is the number of clusters, T is the number of iterations the algorithm is run, R is the restricted set size and N is the total number of images. In our experiments we find that for any given K, our algorithm converges within a maximum 100 iterations, i.e., $T = 200$. That is why in general KT remains

typically similar to N; so our runtime is proportional to KRT. This is $O(N)$ faster than PAM [44]($O(KRNT)$) with the same sized restricted set. However, we note that PAM in general does not use a restricted set.

4.3 Active Subclustering

In active subclustering we improve our passive subclustering solution by asking humans if a pair of images is from the same class. In a subclustering solution we generally have two kinds of errors:

- **Between-class errors:** One of our objectives in subclustering is to choose centroids covering as many different image classes as possible, so we want to eliminate centroids that are from the same class, using constraints provided by humans. We refer to these errors as between-class errors and a centroid pair of the form (c_i, c_j) as a between-class pair.
- **Within-class errors:** In subclustering we also want images in the same cluster to be from the same class. So we look for images with classes that are different from the classes of the centroids they are assigned to and correct them. They are called within-class errors and a centroid-image pair of the form (c_k, I_k^j) is called a within-class pair.

One of the major challenges in active subclustering is that we want to correct both between-class and within-class errors simultaneously. In a subclustering solution we have K centroids with n images assigned to each of these centroids. At any iteration, we can ask about any of the $O(K^2)$ between-class centroid pairs or $O(Kn)$ within-class centroid-

image pairs. We choose the image pair that is most useful (formally defined later as a utility function) for subclustering. One factor in determining the usefulness of an image pair is the likelihood that it reveals an error. So we ask about centroid pairs that are likely to be from the same class to correct between-class errors.

However the usefulness of within-class pairs not only depends on the similarity between image I_k^j and the centroid c_k, but also depends on the likelihood that c_k will remain in the centroid set $\mathcal{C}$ even after we correct the between-class errors. If c_k is very unlikely to remain in $\mathcal{C}$ after between-class errors have been corrected, corresponding within-class pairs (c_k, I_k^j) are considered less useful (even if they are likely to be erroneous) for subclustering and we should not spend a good amount of human budget in correcting them. At each iteration we have to estimate the probability that a centroid c_k will remain in $\mathcal{C}$. This is hard to compute as it depends on other centroids, their pairwise similarity and also changes made to $\mathcal{C}$ during between-class error correction. To estimate such probabilities we make an assumption that for each c_k, its probability of remaining in $\mathcal{C}$ will be determined by another centroid $c_k{}' \in (\mathcal{C} - c_k)$, which is most likely to be similar to c_k. We assume that high similarity between c_k and $c_k{}'$ implies that c_k is unlikely to remain in set $\mathcal{C}$ after between-class error correction. We note that there are other possible ways we can estimate this probability, however this method is very efficient and works well in practice.

Now, let us look at a toy example in Figure 2 to understand intuitively what our estimation means. In this subclustering example, it is evident that I_i^1 is less likely to belong to the same class as c_i than I_j^1's likelihood of being in the same class as c_j. However since c_i is very close to c_k, there is a good chance that c_i will be taken out of $\mathcal{C}$ due to it's similarity with c_k. On the other hand c_j is unlikely to be from the same class as c_k

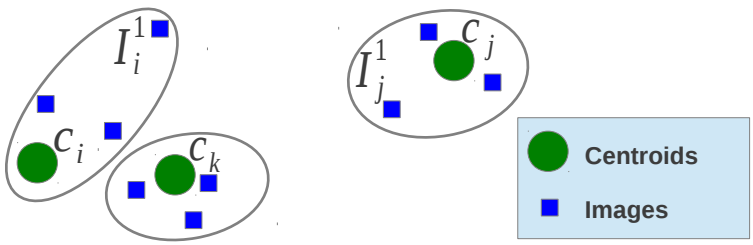


Figure 2: This is a subclustering example with 3 subclusters and with 4 images in each subcluster. We ask humans about within-class pairs like (c_j, I_j^1) before (c_i, I_i^1). (see text)

and should remain in $\mathcal{C}$. So in our approach we prefer to ask users about pairs like (c_j, I_j^1) before (c_i, I_i^1).

Based on our estimates as described above, we show either a between-class or a within-class pair to humans and update our subclustering solution with their response. We continue as long as the human budget permits.

As we get human input, pairwise probabilities $P(I_i = I_j)$ also change, i.e., same-class image pairs are assigned a probability of 1 and different-class image pairs are assigned 0. We also update the subclustering solution using our passive algorithm and these new probabilities (details are provided later). We note that we want to build algorithms that depend only on pairwise image distances and not on image vectors. For this reason we do not learn a new distance metric as we obtain additional constraints. Also we believe the amount of pairwise supervision we obtain from users in active subclustering would not be enough to alter the distances significantly. For example, for a dataset of 51418 images, we eventually get supervision for only 0.0108% of all possible pairs to reach almost perfect subclustering.

Now, we define a utility function $\boldsymbol{U}(I_i, I_j)$ for all within-class and between-class image pairs, where I_i and I_j are any pair of images. At each iteration we choose the

image pair for which this functional value is maximum. For a between-class image pair the utility function is defined as, $\boldsymbol{U}(c_i, c_j) = P(c_i, c_j)$, where c_i and c_j are centroids in $\mathcal{C}$. For a within-class pair (c_k, I_k^j):

$$\boldsymbol{U}(c_k, I_k^j) = (1 - P(c_k = I_k^j))(1 - P(c_k = {c_k}')) \tag{4.3}$$

where ${c_k}' = \underset{c}{\operatorname{argmax}} P(c_k = c)$ and $c \in (\mathcal{C} - c_k)$.

A priority queue $\mathbf{PQ}$ of size $O(K^2 + Kn)$ is created that contains utility function ($\boldsymbol{U}$) values for $O(K^2)$ within-class and $O(Kn)$ between-class pairs. Highest priority is given to the maximum utility pair.

The pseudocode of our active algorithm is described in Algorithm 3 and the details are provided in Appendix 6.3. The steps of our active approach are designed such that we can correct both between-class and within-class errors simultaneously, with minimum human interaction. The image pair selection for active input is based on a utility function for image pairs. This utility function indicates the likelihood that an error will be revealed by asking about that image pair and also whether detecting that error is at all required, as the clustering solution changes while we get human feedback.

4.4 Subclustering Jaccard's Coeff. (SJC)

Many methods exist for evaluating the quality of a clustering of data relative to ground truth. We feel that these are not ideal for evaluating subclustering, however. We propose a novel metric to evaluate subclustering called Subclustering Jaccard's Coefficient

Algorithm 3 Active Subclustering

Given: $\mathbf{PQ}$ (Priority Queue), Max_questions
while num_questions $\leq$ Max_questions **do**
  Pop from $\mathbf{PQ}$ and ask about the corresponding image pair
  if the selected pair is a between-class pair (c_i, c_j) **then**
    if User says "no" **then**
      Update the corresponding probability.
    else
      Update the corresponding probability.
      The current solution is no longer locally optimal.
      we return to the optimization procedure used in the passive algorithm until we find another optimal centroid set.
    end if
  else
    The selected pair is a within-class pair (c_k, I_k^j)
    if User says "yes" **then**
      Update the corresponding probability.
    else
      Update the corresponding probability.
      Replace I_k^j with another image in the database $I_k^{j'}$ that is the next closest to c_k and has not already been assigned to any other centroid.
    end if
  end if
  Update the utility functions and the priority queue $\mathbf{PQ}$.
end while
Output: An improved subclustering solution.

(SJC), which is motivated by Jaccard's coefficient [68] (detail in Chapter 2) for complete clustering. SJC mainly captures two properties of a subclustering solution:

- **Coverage:** We want K subclusters to represent K different ground truth classes.
- **Purity:** When a subcluster represents a ground truth class, it should be as pure as possible.

We first compute a simple mapping between the subclusters and the ground truth classes. For each class, we match it with the subcluster that contains the most elements from this class. This selects at most one subcluster for each class. Then we evaluate the quality of these subclusters using a metric similar to the Jaccard's coefficient. Specifically, subclustering S is evaluated with respect to the ground truth G

$$\text{SJC}_G(S) = \frac{\sum_{k=1}^{K} ind_k * \frac{SS_k}{{}^nC_2}}{K} \tag{4.4}$$

For each class, k, SS_k is the number of pairs of elements from class k in the subcluster associated with k, and ${}^nC_2$ is the total number of pairs of elements present in that subcluster. For example, if for class k there is a subcluster with six out of ten elements from class k, the fraction in this sum will be $\frac{{}^6C_2}{{}^{10}C_2}$, i.e.,$\frac{15}{45}$ (Figure 3). Additionally, we require that a subcluster can only represent a class if it contains a sufficient number of elements from that class. In our experiments, we require a subcluster to have greater than half of its elements from a single class; subclusters without this minimal level of purity will not be very useful for browsing. If a subcluster of size ten contains less than six elements of class k it is considered not to be useful. ind_k is an indicator variable that will be zero if there is no subcluster that contains a sufficient number of elements from class k. This

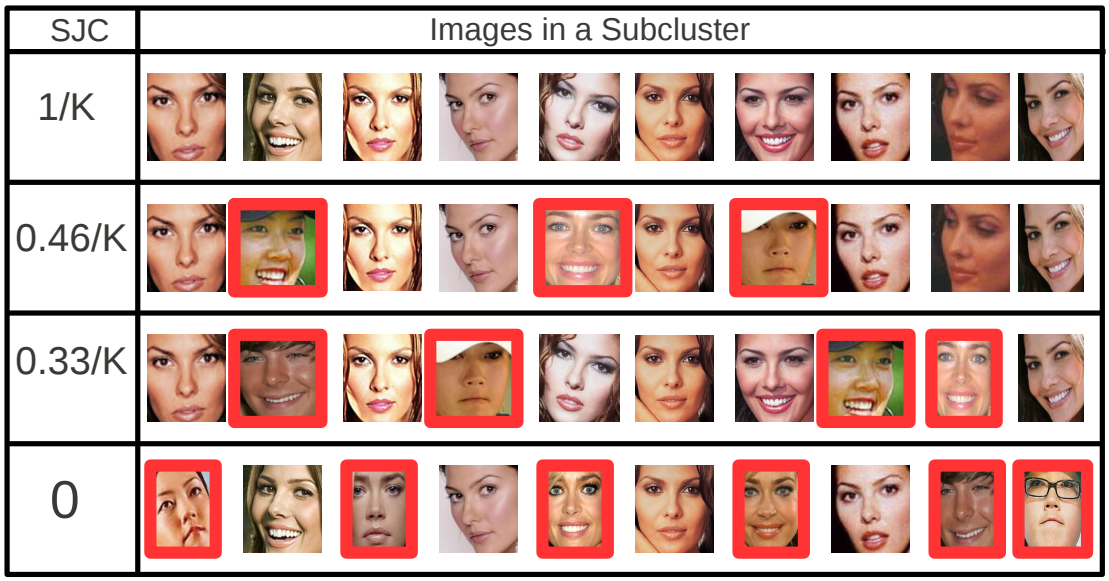


Figure 3: Some subclusters ($n = 10$) and the fraction they contribute to a Subclustering Jaccard's Coefficient calculation (K clusters). Red boxes denote errors in a subcluster.

also ensures that a subcluster can only represent one class.

4.5 Fast-PAM

Because there are no prior algorithms for subclustering, we will adapt other clustering algorithms to the problem so that we may use them as baselines. K-medoids algorithms are easy to adapt to subclustering, by selecting each cluster's medoid, and the $n-1$ closest other points in the cluster as a subcluster. We see that PAM [44] is much more effective on large image sets than a baseline K-medoids algorithm[2] and we provide some experimental evidence regarding that in the Appendix 6.4. However PAM is slow and not scalable to medium or large datasets. A contribution of this work is to provide a faster algorithm that optimizes the same cost as PAM. This can be used to cluster datasets of size over

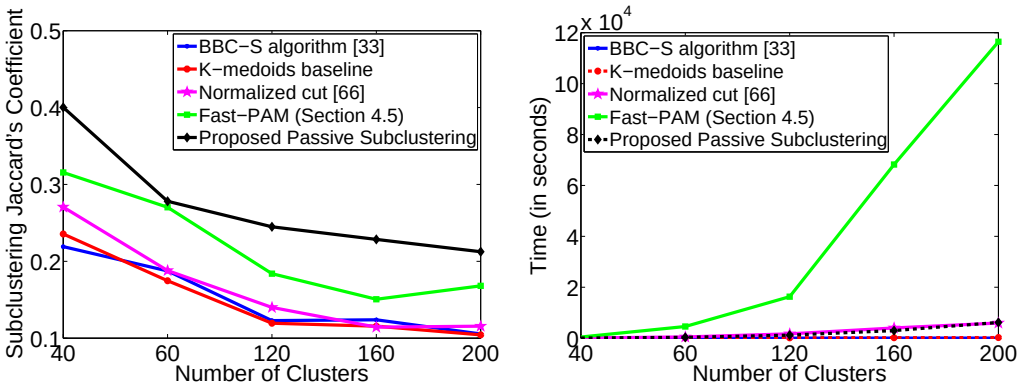
[2]A baseline K-medoids is similar to K-means [56] but the medoid is calculated instead of the mean in the update step.

$10,000$ in around 7 minutes (Matlab implementation), and could be further sped up with an optimized C implementation. We provide the implementation details of Fast-PAM in Appendix 6.4.

4.6 Experimental Results

We run experiments in two different domains: face images from Pubfig [49] and leaf images from Leafsnap [47]. Unlike most other face datasets, images in Pubfig are taken in completely uncontrolled settings with non-cooperative subjects. Thus, there is a large variation in pose, lighting, expression, scene, camera, imaging conditions and parameters, etc.... For the passive algorithm, we create several subsets of Pubfig [49] containing 40, 80, 120, 160 and 200 clusters. These face image sets are called Face-40, Face-80, Face-120, Face-160 and Face-200 respectively. Face-200 is the complete Pubfig dataset containing 51418 images from 200 classes[3]. The number of images in each class in Face-200 vary from 56 to 1434 and we use subcluster size of $n = 10$. We also create a leaf image dataset containing 1535 images from 27 classes to test our algorithms in a different image domain and we call this dataset Leaf-27. Each class in this case has at least 50 or more images and we use a subcluster size of 5. This is a subset of the leaf image data used in Leafsnap [47]. We also create a small face dataset containing 250 images from 10 classes (called Face-10) and a leaf dataset containing 250 images from 25 classes (called Leaf-25). We use a subcluster size of 5 and 3 for Face-10 and Leaf-25 respectively. In our experiments we required pairwise distances between all images. The distance

[3]Note that the total number of images we use is less than the actual number of images (58797) in Pubfig because of inactive links to images.



(a) SJC comparison (b) Time comparison

Figure 4: Comparison of our proposed passive subclustering algorithm with other baselines for Face-40, 80, 120, 160, 200 ($|\mathcal{R}| = 20K$).

matrix for face images and leaf images are calculated based on algorithms in [9] and [47] respectively. Scaling parameter λ is chosen based on a small holdout training dataset and we use $\lambda = 5$ for the face images and $\lambda = 0.2$ for leaf images.

4.6.1 Subclustering

Our passive algorithm is compared with four other algorithms, Fast-PAM (Section 4.5), K-medoids baseline, Normalized Cut [66] and BBC-S [33], where the first three are traditional complete clustering algorithms. From complete clustering results, the centroid of each cluster along with the $n - 1$ nearest images from the centroid create a subcluster. This seems to be a natural way of creating subclusters from clusters and provides useful baselines for comparisons. We use SJC as the evaluation metric. Each algorithm is run 10 times for each dataset and mean SJC is reported. From Figure 4 and Table 4.1, we can see that our proposed algorithm significantly outperforms other baselines. For example, our algorithm achieves 26% higher SJC than that of Fast-PAM and is 30 times faster than Fast-PAM (even with the fast implementation) for Face-200.

Table 4.1: Subclustering comparison for Leaf-27.

Algorithms	SJC	time (s)
BBC-S algorithm [33]	0.5023	0.4114
K-medoids baseline	0.5778	0.4244
Normalized cut [66]	0.6037	0.7709
Fast-PAM (Section 4.5)	0.6103	77.32
Proposed Passive Subclustering Algorithm	**0.6159**	9.7857

4.6.2 Restricted Set Size and Subcluster Size Variation

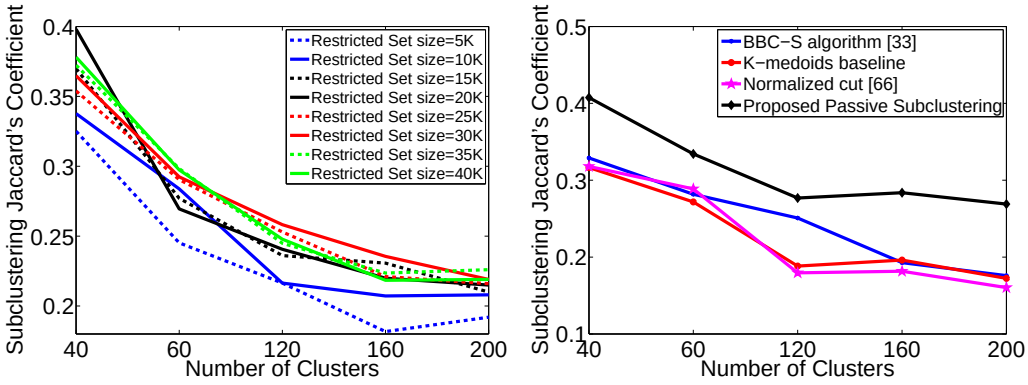
One of the important innovations that allows our algorithm to run efficiently is to select the set of centroids from a restricted set only. If the restricted set size is smaller, our algorithm will be efficient but results may not be accurate. On the other hand with a larger restricted set we are likely to get a much better solution but at the cost of efficiency. We empirically find that selecting restricted sets of size $5K$-$25K$ (K is the number of clusters) gives reasonable trade-off between efficiency and accuracy. In Figure 5a we show how the clustering performance varies (after the passive algorithm) for all the face datasets as we change the restricted set size. The runtime increases as we increase the restricted set size, e.g., for Face-40 runtime increases from 19 seconds to 95 seconds, as we increase the restricted set size from $5K$ to $40K$.

We expect that a user will provide the subcluster size based on her requirements. Subcluster size of 10 seems reasonable for most of the possible applications of subclustering such as browsing, categorization, summarization. This size is not too large that the subclusters will be erroneous and not too small that there will be no intra-class variation in each subcluster. However we performed some experiments where the subcluster size is varied to see how sensitive our approach is with respect to the subcluster size.

In Figures 5b-5f, we compare the proposed approach with other approaches, where the subcluster size varies from 5 to 25. As we increase the subcluster size, all algorithms' performance become a little worse, but our algorithm still outperforms other approaches by a significant margin.

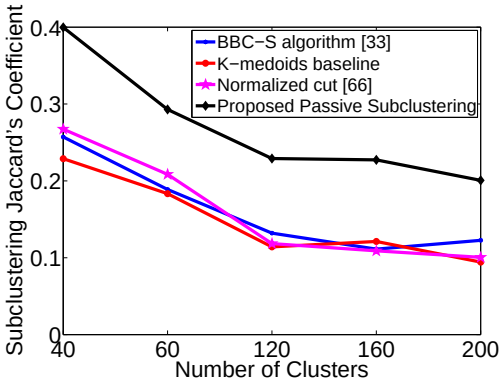
4.6.3 Active Subclustering

- We compare our active algorithm to three previous active clustering algorithms [6, 14, 78], adapted to subclustering by selecting n images from each cluster to use as subclusters.
- **Face-200:** Our approach requires around $143,000$ questions to reach a SJC of 0.92 (equivalent to 120 perfect subclusters and 80 subclusters with only one mistake) for Face-200. We note that asking $143,000$ questions in Mechanical Turk costs only around $358. Basu *et al.*'s algorithm [6] achieves SJC of just 0.44 after $143,000$ questions. The "Explore" stage itself takes more than $200,000$ questions and complete clustering of the dataset takes as many as 2.3 million questions (Mturk cost: $5800). Xiong *et al.*'s approach [78] also falls short producing SJC of only 0.2 after $143,000$ questions. In Figure 6a we plot SJC against the number of questions for Face-200. We could not run [14] on Face-200, because their approach is not fast enough for such a large dataset.
- **Leaf-25 and Face-10:** Although our active algorithm is aimed at problems with a large number of clusters, we run our approach in datasets (Leaf-25 and Face-10 in Figure 6b and 6c respectively) with a small number of clusters for comparison and

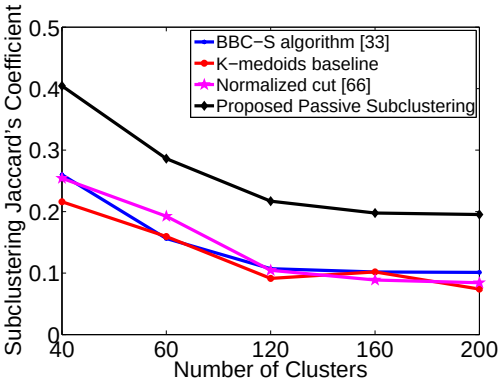


(a) Restricted Set Size Variation

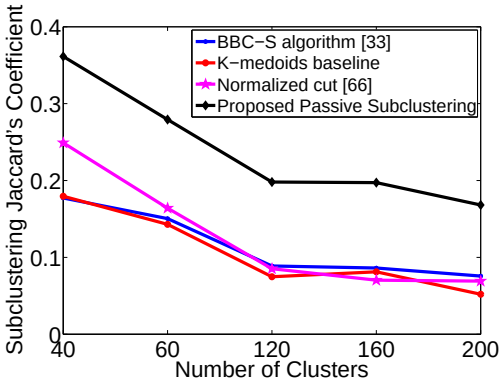
(b) Subcluster Size is 5



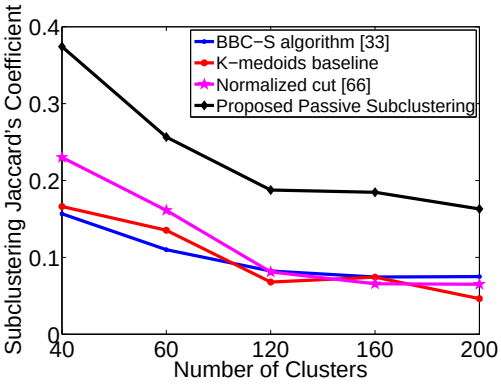
(c) Subcluster Size is 10



(d) Subcluster Size is 15

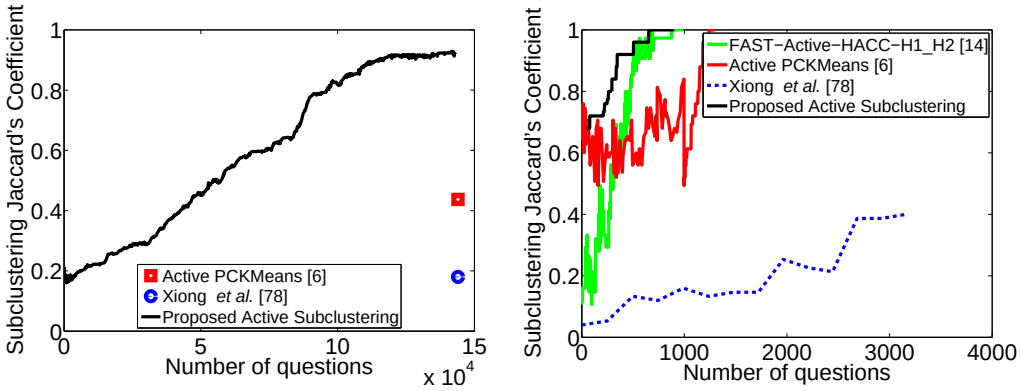


(e) Subcluster Size is 20



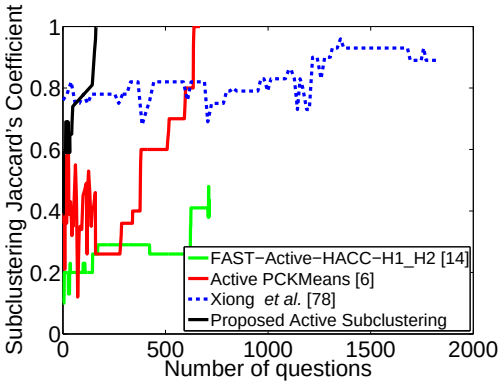
(f) Subcluster Size is 25

Figure 5: Varying restricted set size and subcluster size. (a) The restricted set size $|\mathcal{R}|$ is varied from $5K$ to $40K$. (b)-(f) The subcluster size is varied from 5 to 25 and performance for all algorithms were reported. Restricted set size was set to a fixed value of $20K$. Since Fast-PAM is slow, we do not report that algorithm for this part of the experimental evaluation.



(a) Face-200

(b) Leaf-25



(c) Face-10

Figure 6: Proposed active algorithm's performance for several datasets.

we still outperform all approaches. For example, we reach perfect subclustering by asking just 654 and 161 questions for Leaf-25 and Face-10 respectively, where the next best approaches take 876 and 639 questions respectively.

- While we feel that these comparisons with previous active clustering algorithms provide a useful baseline, we stress that these prior approaches were not designed for subclustering, and poor performance on these tasks should not be taken as a criticism of the overall algorithms.

4.7 Conclusion

We propose a novel clustering problem called subclustering and show its application in large image datasets. We presented a passive and an active algorithm for subclustering. We compare our passive algorithm with several baselines and show that our proposed algorithm is better and scalable to large datasets. We compare our active subclustering algorithm with active complete clustering algorithms and demonstrate that the proposed algorithm is much better. We also present a metric for subclustering evaluation. Although experimental results are shown only for images, this idea can be extended to any clustering domain.

Chapter 5: **Conclusion and Future Works**

In this thesis we have proposed three approaches to improve image clustering with pairwise constraints that indicate whether a pair of images belong to the same category or not. We have shown that these approaches can improve clustering significantly. However there is still a plenty of scope to improve clustering of images. Although extracting better features may improve clustering, perfect unsupervised clustering still seems unlikely in the near future. That strongly motivates further research on semi-supervised clustering to improve the state-of-the-art. There are many interesting open problems in the area of semi-supervised clustering. We point out only a few of them which are directly related to the research carried out in this thesis:

- In chapter 2 we have proposed a method for learning better image distances using the pairwise constraints. This is a general approach that can be used to learn distances using any kinds of pairwise distances between all images, and does not require any vector representation of images. Our approach was developed for clustering images, in which all distances and constraints are available when we learn the new distances. The out-of-sample problem is also useful, in which a query image retrieves its nearest neighbors using the learned distances. Extending our distance learning approach to an out-of-sample scenario is a possible future work

which will enable us to compute distances between new images that were not part of the training data.

- Our proposed approach for distance learning is a general but non-parametric approach that can work with any kinds of pairwise distances. It is not clear how a parametric method can be developed for distance learning which will work with all kinds of distances. Perhaps it is possible to develop parametric methods catered to each type of distance individually. These approaches, if developed in future, might produce better distances between images than a general framework such as ours that applies to all kinds of distances.

- In chapter 3 we have proposed an active learning technique for image clustering where useful pairwise constraints are obtained from humans in an interactive manner. In most of the experiments we assumed that humans are always correct in judging the similarity of two images. We also performed some experiments using real humans in Amazon Mechanical Turk. We found that it is not possible to reach perfect clustering if any of the inputs from humans is erroneous. Although our approach performs better than the state-of-the-art with erroneous human inputs it is far from perfect. That is why it is useful to build active models that can incorporate erroneous pairwise constraints in a more robust way.

- In chapter 3 and 4 we have proposed active learning methods for clustering and subclustering. In these approaches we assume that at each iteration only one pairwise constraint can be obtained. However in real platforms such as Amazon Mechanical Turk we can avail ourselves of multiple annotators providing constraints simultane-

ously. It is important to propose active methods where multiple image pairs can be chosen at each iteration. That will require developing fast methods for evaluating all sets of possible pairs and then choosing the best set of pairs for obtaining constraints. This could also be an interesting future work which will enable us to collect a large number of constraints at every iteration and use platforms like Mechanical Turk in a more effective way.

- As another future work we envision a large scale interactive system involving humans and images. The proposed system will contain a very large collection of images; multiple humans will interact with those images in different ways. Humans can upload, label, provide pairwise constraints, retrieve, play simple image based games or browse the image collection depending on their expertise or interest. The system should have multiple algorithmic components such as clustering with constraints, retrieval, active selection of tasks, assigning tasks based on human expertise, learning from human interactions etc.... Building such a system will require expertise in computer vision, machine learning, algorithms, database management, programming and many other related areas. The system should evolve with time as more images are uploaded and as more human interactions happen. In this thesis we have looked only at a small part of this system which involves clustering with constraints. However there are many fascinating problems that must be solved to develop the system as proposed and should be pursued in the future.

Chapter 6: **Appendix**

6.1 Detailed Proofs for Distance Learning

Here we note some initial triangle inequality constraints, which we will use to prove the Lemmas:

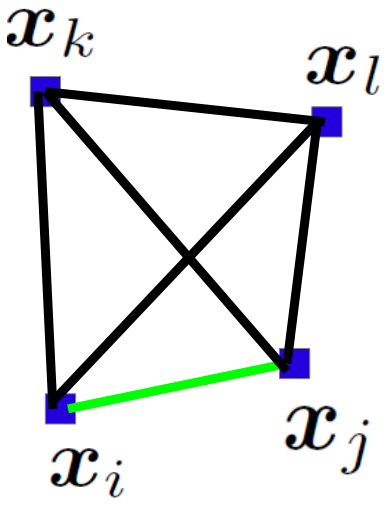
$$d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_i, \boldsymbol{x}_l) \tag{6.1a}$$

$$d_I(\boldsymbol{x}_j, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_j, \boldsymbol{x}_l) \tag{6.1b}$$

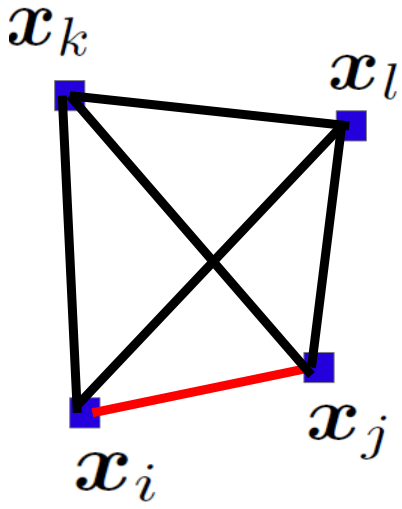
$$d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_i, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \tag{6.1c}$$

$$d_I(\boldsymbol{x}_j, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \tag{6.1d}$$

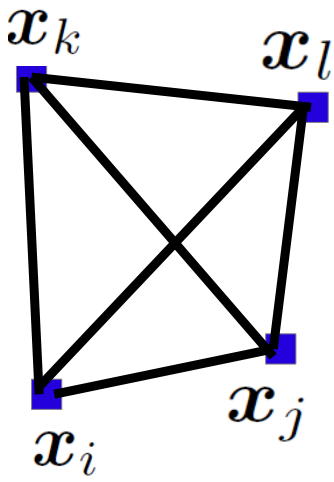
Proof for Lemma 1. We will refer to Figure 1a and the QP in 2.1 for this proof. We find the minimum cost solutions for the graph (Figure 1a) considering triangles $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$ and $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_l$ and show that this solution does not violate the triangle inequality for $\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_l$ and $\Delta \boldsymbol{x}_j \boldsymbol{x}_k \boldsymbol{x}_l$. When an optimal solution obtained using a subset of the constraints ($\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$ and $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_l$) also satisfies other constraints ($\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_l$ and $\Delta \boldsymbol{x}_j \boldsymbol{x}_k \boldsymbol{x}_l$) which were not enforced, it is not required to change the optimal solution. Thus the solution which we obtain in this proof is optimal.



(a) Must-link constraint.



(b) Can't-link constraint.



(c) Four points in 2-d.

Figure 1: Simple examples in 2-d Euclidean space (green denotes must-link edge and red denotes can't-link edge) to explain our proposed approach.

There is a must-link constraint between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$. To minimize the objective function in 2.1, and to avoid violation of the triangle inequality for triangle $\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_j$, we will have $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = d_F(\boldsymbol{x}_j, \boldsymbol{x}_k) = \frac{d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_k)}{2}$. Similarly for $\Delta \boldsymbol{x}_i \boldsymbol{x}_l \boldsymbol{x}_j$, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) = d_F(\boldsymbol{x}_j, \boldsymbol{x}_l) = \frac{d_I(\boldsymbol{x}_i, \boldsymbol{x}_l) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_l)}{2}$.

Now we show that the triangle inequality holds for $\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_l$ and $\Delta \boldsymbol{x}_j \boldsymbol{x}_k \boldsymbol{x}_l$ with $d_F(\boldsymbol{x}_k, \boldsymbol{x}_l) = d_I(\boldsymbol{x}_k, \boldsymbol{x}_l)$. Adding equations 6.1a and 6.1b, we get $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_F(\boldsymbol{x}_i, \boldsymbol{x}_l)$. Similarly adding equations 6.1c and 6.1d, we get $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_k, \boldsymbol{x}_l)$. By symmetry $d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_F(\boldsymbol{x}_i, \boldsymbol{x}_k)$. So the triangle inequality holds for $\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_l$. Similarly, we can prove that triangle inequality holds for $\Delta \boldsymbol{x}_j \boldsymbol{x}_k \boldsymbol{x}_l$. So $d_F(\boldsymbol{x}_k, \boldsymbol{x}_l) = d_I(\boldsymbol{x}_k, \boldsymbol{x}_l)$ is not required to change. Thus $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) \geq min(d_I(\boldsymbol{x}_i, \boldsymbol{x}_k), d_I(\boldsymbol{x}_j, \boldsymbol{x}_k))$ because $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = \frac{d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_k)}{2}$. □

Proof for Lemma 2. Follows from the proof for Lemma 1. □

Proof for Lemma 3. We will refer to Figure 1b and the QP in 2.1 for this proof. Note that the can't-link threshold distance was L. Since there is a can't-link constraint between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ the corresponding distance is set to be L. We find the minimum cost solutions for the graph (Figure 1b) considering triangles $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$ and $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_l$ and show that this solution does not violate the triangle inequality for $\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_l$ and $\Delta \boldsymbol{x}_j \boldsymbol{x}_k \boldsymbol{x}_l$. Now there are three possible cases:

Case 1: Triangle inequality is violated for both $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_l$ and $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$. To avoid any violation and to minimize cost, the new distances will be: $d_F(\boldsymbol{x}_j, \boldsymbol{x}_k) = \frac{L}{2} + \frac{d_I(\boldsymbol{x}_j, \boldsymbol{x}_k) - d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)}{2}$, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = \frac{L}{2} + \frac{d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) - d_I(\boldsymbol{x}_j, \boldsymbol{x}_k)}{2}$, $d_F(\boldsymbol{x}_j, \boldsymbol{x}_l) = \frac{L}{2} +$ $\frac{d_I(\boldsymbol{x}_j, \boldsymbol{x}_l) - d_I(\boldsymbol{x}_i, \boldsymbol{x}_l)}{2}$, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) = \frac{L}{2} + \frac{d_I(\boldsymbol{x}_i, \boldsymbol{x}_l) - d_I(\boldsymbol{x}_j, \boldsymbol{x}_l)}{2}$. Now we have to prove that with

$d_F(\boldsymbol{x}_k, \boldsymbol{x}_l) = d_I(\boldsymbol{x}_k, \boldsymbol{x}_l)$ and the new set of distances, triangle inequalities hold for both $\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_l$ and $\Delta \boldsymbol{x}_j \boldsymbol{x}_k \boldsymbol{x}_l$. We know that $L \geq d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_k)$, $L \geq d_I(\boldsymbol{x}_i, \boldsymbol{x}_l) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_l)$ (that is why the triangle inequality was violated for $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$ and $\Delta \boldsymbol{x}_i \, bmx_j \boldsymbol{x}_l$). We divide these two inequalities by 2 and add them together. If the resulting equation is added with another equation $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_i, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_l, \boldsymbol{x}_k)$, we get $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_k, \boldsymbol{x}_l)$. We also have $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_i, \boldsymbol{x}_l)$ and $d_I(\boldsymbol{x}_j, \boldsymbol{x}_l) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_j, \boldsymbol{x}_k)$. Adding this two equations we can show that, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_F(\boldsymbol{x}_i, \boldsymbol{x}_l)$. By symmetry we can prove that, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_F(\boldsymbol{x}_i, \boldsymbol{x}_k)$. Thus the triangle inequality is proved for $\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_l$. Similarly we can prove the triangle inequality for $\Delta \boldsymbol{x}_j \boldsymbol{x}_k \boldsymbol{x}_l$. Now we see that $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) - d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) = \frac{L}{2} - \frac{d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_k)}{2} \geq 0$, because $L \geq d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_k)$. That implies $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) \geq d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$. Similarly it can be proved that $d_F(\boldsymbol{x}_j, \boldsymbol{x}_k) \geq d_I(\boldsymbol{x}_j, \boldsymbol{x}_k)$, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_i, \boldsymbol{x}_l)$ and $d_F(\boldsymbol{x}_j, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_j, \boldsymbol{x}_l)$.

Case 2: The triangle inequality is violated for either $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$ or $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_l$. Without loss of generality we assume the triangle inequality is violated for $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$. In this case the new distances will be: $d_F(\boldsymbol{x}_j, \boldsymbol{x}_k) = \frac{L}{2} + \frac{d_I(\boldsymbol{x}_j, \boldsymbol{x}_k) - d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)}{2}$, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = \frac{L}{2} + \frac{d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) - d_I(\boldsymbol{x}_j, \boldsymbol{x}_k)}{2}$. However, $d_F(\boldsymbol{x}_j, \boldsymbol{x}_l) = d_I(\boldsymbol{x}_j, \boldsymbol{x}_l)$ and $d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) = d_I(\boldsymbol{x}_i, \boldsymbol{x}_l)$, because $d_I(\boldsymbol{x}_i, \boldsymbol{x}_l) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_l) \geq L$. Now we have to prove that with $d_F(\boldsymbol{x}_k, \boldsymbol{x}_l) = d_I(\boldsymbol{x}_k, \boldsymbol{x}_l)$ and the new set of distances, triangle inequalities hold for both $\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_l$ and $\Delta \boldsymbol{x}_j \boldsymbol{x}_k \boldsymbol{x}_l$. Now since $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) \geq d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$, it can easily be proved that $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_k, \boldsymbol{x}_l)$ and $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_F(\boldsymbol{x}_i, \boldsymbol{x}_l)$. Now we have, $d_I(\boldsymbol{x}_i, \boldsymbol{x}_l) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_l) \geq L$, $d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) + d_I(\boldsymbol{x}_j, \boldsymbol{x}_k) \geq d_I(\boldsymbol{x}_j, \boldsymbol{x}_l)$ and $d_I(\boldsymbol{x}_i, \boldsymbol{x}_l) +$

$d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$. Adding all three equations, we can show that $d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) + d_I(\boldsymbol{x}_k, \boldsymbol{x}_l) \geq d_F(\boldsymbol{x}_i, \boldsymbol{x}_k)$. Thus the triangle inequality is proved for $\Delta \boldsymbol{x}_i \boldsymbol{x}_k \boldsymbol{x}_l$. Similarly we can prove the triangle inequality for $\Delta \boldsymbol{x}_j \boldsymbol{x}_k \boldsymbol{x}_l$. Then as in Case 1, it can be proved that $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) \geq d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$ and $d_F(\boldsymbol{x}_j, \boldsymbol{x}_k) \geq d_I(\boldsymbol{x}_j, \boldsymbol{x}_k)$. However, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_l) = d_I(\boldsymbol{x}_i, \boldsymbol{x}_l)$ and $d_F(\boldsymbol{x}_j, \boldsymbol{x}_l) = d_I(\boldsymbol{x}_j, \boldsymbol{x}_l)$.

Case 3: No triangle inequality is violated. In this case all distances remain the same as the initial distances.

Thus $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) \geq d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$. □

Proof for Lemma 4. Follows from proof for Lemma 3. □

Now with the following theorem we show that even with the triangle inequality constraints that we enforce in the QP in equation 2.2, distances change in a similar way as they would have, had we enforced all the triangle inequality constraints (as in equation 2.1). We note that distances that are smaller than or equal to a fixed threshold t_h are called nearest neighbors or "small" and distances larger than t_h are called "large" for all d_I, d_F and d'_F.

Let us refer to Figure 1c in the following Theorems. In the QP in 2.1, the enforced triangle inequalities are for triangles $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$, $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_l$, $\Delta \boldsymbol{x}_k \boldsymbol{x}_j \boldsymbol{x}_l$, $\Delta \boldsymbol{x}_k \boldsymbol{x}_i \boldsymbol{x}_l$. In the QP in 2.2, triangle inequalities for $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$ and $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_l$ might be enforced. Note that when a must-link/can't-link constraint is present between $\boldsymbol{x}_j$ and $\boldsymbol{x}_l$, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$, even when all the triangle inequality constraints are enforced (from Lemma 2 and 4). Without loss of generality we consider the case when there is a must-link/can't-

link constraint between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and we study how $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$ is modified when we run the QPs in equation 2.1 and 2.2.

Proof for Theorem 5. I. **Case 1:** There is a must-link between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and $d_I(\boldsymbol{x}_j, \boldsymbol{x}_k) > t_h$. By Lemma 1, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) > t_h$, when QP in 2.1 is used. Now with QP in 2.2, we do not consider the triangle $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$, which was responsible for change in $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$, because $\boldsymbol{x}_k$ is not a nearest neighbor to either $\boldsymbol{x}_i$ or $\boldsymbol{x}_j$. Thus $d'_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) > t_h$. **Case 2:** There is a must-link between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and $d_I(\boldsymbol{x}_j, \boldsymbol{x}_k) \leq t_h$. Triangle $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$, which is responsible for change in $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$ is considered in both QP in 2.1 and 2.2. Thus $d'_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = d_F(\boldsymbol{x}_i, \boldsymbol{x}_k)$. **Case 3:** There is a can't-link between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and $d_I(\boldsymbol{x}_j, \boldsymbol{x}_k) > t_h$. By Lemma 3, $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) > t_h$, when QP in 2.1 is used. Now with QP in 2.2, we do not consider the triangle $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$, which was responsible for change in $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$, because $\boldsymbol{x}_k$ is not a nearest neighbor to either $\boldsymbol{x}_i$ or $\boldsymbol{x}_j$. Thus $d'_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) > t_h$. **Case 4:** There is a can't-link between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and $d_I(\boldsymbol{x}_j, \boldsymbol{x}_k) \leq t_h$. Triangle $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$, which is responsible for change in $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$ is considered in both QP in 2.1 and 2.2. Thus $d'_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = d_F(\boldsymbol{x}_i, \boldsymbol{x}_k)$. Thus, $d'_F(\boldsymbol{x}_i, \boldsymbol{x}_k) > t_h$ if $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k) > t_h$ and $d_F(\boldsymbol{x}_i, \boldsymbol{x}_k) > t_h$.

II. $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$ can become small ($\leq t_h$) only when there is a must-link between $\boldsymbol{x}_i$ and $\boldsymbol{x}_j$ and $d_I(\boldsymbol{x}_j, \boldsymbol{x}_k) \leq t_h$. Triangle $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$, which is responsible for change in $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$ is considered in both QP in 2.1 and 2.2 because $\boldsymbol{x}_i \boldsymbol{x}_j$ is a must-link edge and $\boldsymbol{x}_j \boldsymbol{x}_k$ is a nearest neighbor edge. Thus $d'_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = d_F(\boldsymbol{x}_i, \boldsymbol{x}_k)$.

III. Triangle $\Delta \boldsymbol{x}_i \boldsymbol{x}_j \boldsymbol{x}_k$, which is responsible for change in $d_I(\boldsymbol{x}_i, \boldsymbol{x}_k)$ is considered in

both QP in 2.1 and 2.2 because $\boldsymbol{x}_i\boldsymbol{x}_j$ is a must/can't-link edge and $\boldsymbol{x}_i\boldsymbol{x}_k$ is a nearest neighbor edge. Thus $d'_F(\boldsymbol{x}_i, \boldsymbol{x}_k) = d_F(\boldsymbol{x}_i, \boldsymbol{x}_k)$.

IV. The same proof as part III applies.

□

6.2 Computational Complexity of Passive Subclustering

In this section we calculate the computational complexity of our passive algorithm. Notations used in this section are the same as used in Chapter 4. We assume that we are given the distance between all pairs of images in $\mathcal{U}$. Actually, our algorithm only requires distances between images in $\mathcal{U}$ and images in $\mathcal{R}$; if these are not available a priori they may be computed in $O(RND)$ time (where D is the dimension of feature vectors).

For step (a) (Section 4.2.2 in Chapter 4) in our algorithm we have to find the $n-1$ closest points to each centroid in the restricted set. Finding the $n-1$ smallest elements in an array of size N can be done in $O(N)$ time using a linear time selection algorithm [21] (subcluster size n is assumed to be constant). So, this takes $O(RN)$ time for R centroids. For step (b), we maintain a vector of size R and update at every iteration, which takes $O(R)$; total cost is $O(RT)$. Another major computational cost comes from tracking the overlap between clusters in step d(i). We can do this by maintaining a matrix of size K-by-R and updating it at every iteration. This requires $O(KR)$ time in each iteration; total $O(KRT)$. For each replacement overlapping can be checked in constant time. The final major computational cost comes from considering KR replacements (step d(ii)); this is carried out T times and takes $O(KRT)$ time. Each replacement is done in constant

time with the precomputed cost of addition or removal of a centroid (see steps (a) and (b)). Other steps in our algorithm are not significant with respect to computational cost. So the total computational cost for the proposed algorithm is $O(RN + KRT)$. In our experiments we find that KT is typically similar to N; so our runtime is proportional to KRT. This is $O(N)$ faster than PAM [44]($O(KRNT)$) with a same sized restricted set. However, we note that PAM in general does not use a restricted set.

6.3 Details of the Active Subclustering Method

Here we formally describe our active algorithm for correction of between-class and within-class errors:

(a) A priority queue $\mathbf{PQ}$ of size $O(K^2 + Kn)$ is created that contains utility function ($\boldsymbol{U}$) values for $O(K^2)$ within-class and $O(Kn)$ between-class pairs. Highest priority is given to the maximum utility pair.

(b) We pop from the priority queue and ask about the corresponding image pair.

(c) If the selected pair is a between-class pair (c_i, c_j):

- User says "no": Cost of the present centroid set decreases (because a positive cost component contributed by $-log(1 - P(c_i = c_j))$ will be zero after we know c_i and c_j are from different classes), so it still remains locally optimal. We update the corresponding pairwise probability.
- User says "yes": The cost of the current centroid set blows up (as $-log(1 - P(c_i = c_j))$ becomes infinite) and the current solution is no longer locally optimal. At that point we return to the optimization procedure used in the

passive algorithm until we find another optimal centroid set.

(d) If the selected pair is a within-class pair (c_k, I_k^j):

- User says "yes": Update the pairwise probability.
- User says "no": Replace I_k^j with another image in the database $I_k^{j'}$ that is the next closest to c_k and has not already been assigned to any other centroid (to avoid any overlapping induced by our active algorithm).

(e) Within-class thresholding: we keep a threshold (q_{th}) on the maximum number of questions we ask to correct one particular image assignment for a centroid in a within-class pair. Once we decide to replace a cluster element this is always the highest priority in the queue, until we find a good replacement. However, we don't want to get stuck repeatedly asking questions about one cluster when finding a replacement is difficult. So after the q_{th} questions, we do not ask about a $(c_k, I_k^{j'})$ pair and adjust the corresponding probability to ask about it later. q_{th} is updated at every iteration such that when we have corrected most of the between-class errors, we will gradually shift our emphasis to within-class errors and start asking more questions about them (see Appendix $6.3.1$ for details).

(f) Update the utility functions and the priority queue $\mathbf{PQ}$. Goto step (b).

(g) Continue asking about image pairs until the human budget is exhausted.

6.3.1 Thresholding (q_{th}) in Active Learning

We define variables H_{bc} and H_{wc}, which keep track of how often humans agree with the algorithm regarding between-class pairs and within-class pairs respectively. If more

questions are required to get a disagreement (a same-class centroid pair for between-class and a different-class centroid-image pair for within-class) from users, that implies humans are agreeing more with the present subclustering. H_{bc} is defined as the average number of questions asked for between-class pairs to obtain a between-class disagreement. H_{wc} is defined in similar way but corresponds to questions asked for within-class pairs only. These variables are updated after every iteration.

Within-class thresholding (step (e) in active subclustering): we keep a threshold (q_{th}) on the maximum number of questions we ask to correct one particular image assignment for a centroid in a within-class pair. Replacing a cluster element is always the highest priority in the queue, until we find a good replacement. However, we don't want to get stuck repeatedly asking questions about one cluster when finding a replacement is difficult. So after the q_{th} questions, we do not ask about a $(c_k, I_k^{j'})$ pair and adjust the corresponding probability to ask about it later. q_{th} is set to be $\frac{H_{bc}}{H_{wc}}$ and is updated at every iteration. This intuitively means, as we keep correcting between-class errors, we will gradually shift our emphasis to within-class errors and start asking more questions about them.

6.4 Fast Partition Around Medoids (Fast-PAM)

6.4.1 Related Work

We use PAM [44] proposed by Kaufman and Rousseeuw in 1987 as one of our major baselines. Although PAM is slow for large scale datasets, in general K-medoids algorithms like PAM are known to be more robust to outliers and are invariant to translations

and orthogonal transformations to data points [59]. There have been several approaches to make PAM faster like CLARA [44], CLARANS [59] and [60]. We provide a faster implementation of PAM which we call Fast-PAM.

6.4.2 Algorithm Description

First, we introduce some notation to be used in this section. We are given a set of N unlabeled images $\mathcal{U}$ to be clustered into K clusters. N_k denotes the number of images in the k-th cluster ($\sum_{k=1}^{K} N_k = N$). T denotes the number of time-steps or iterations an algorithm is run. $\mathcal{C}$ ($c \in \mathcal{C}$) represents the set of K cluster centroids and $\mathcal{C} \subset \mathcal{U}$, i.e., centroids are members of the dataset only. $\mathcal{R}$ ($|\mathcal{R}| = R$) denotes a restricted set of images (randomly chosen from $\mathcal{U}$), which is larger than the number of clusters but smaller than the total number of images ($\mathcal{R} \subset \mathcal{U}$ and $K < R < N$). $d(I_i, I_j)$ is the distance between images I_i and I_j ($I_i, I_j \in \mathcal{U}$).

The state of the art implementation of PAM has computational cost $O(TKN^2)$ [59], where our implementation takes only $O(TRN)$. Our approach uses some additional data structures and precomputation which helps to carry out each centroid replacement of PAM (c with r) in only $O(n_{avg})$ time, where n_{avg} is the average number of images in each cluster; this helps to reduce the run time of PAM by $O(K)$. For simplicity, we will assume $N = Kn_{avg}$. Also in Fast-PAM we use a restricted set of images to choose the replacements from unlike using all non-centroids in PAM; this gives additional speed up of $O(\frac{N}{R})$. In experiments we see that as the number of clusters increase, Fast-PAM is more efficient than PAM.

The cost function in clustering is of the form $\mathbf{Cost} = \sum_{k=1}^{K} \sum_{I_k^j \in S_k} d(c_k, I_k^j)$, where d measures the distance between two images, c_k is the k-th centroid, I_k^j is the j-th image assigned to the k-th cluster and S_k is the set of images in cluster k.

In PAM, at each iteration, a cluster center is replaced by the data point that reduces the $\mathbf{Cost}$ by the maximum amount. This continues until convergence, that is until no such replacement is found that decreases the $\mathbf{Cost}$.

In the original implementation of PAM [59] each replacement (c with r) takes $O(N)$ (since $N >> k$) time. We reduce this replacement cost from $O(N)$ to $O(n_{avg})$. When we take out a centroid $c \in \mathcal{C}$ from the present set of centroids and add $r \in \mathcal{R}$ to $\mathcal{C}$, the cost changes mainly because of two sets of points. First, the points that were earlier assigned to c as the closest centroid now need to be reassigned to some other centroid. We call these set of points $\mathcal{I}_1$. Second, there is a set of points which will change their present assignment and will be assigned to r, because they are closer to r than their present assignments. We call these set of points $\mathcal{I}_2$. As we mentioned earlier, for simplicity we assume $|\mathcal{I}_1| = |\mathcal{I}_2| = n_{avg}$ to calculate the complexity of our implementation in a simpler way. Now we discuss the additional matrices, which we maintain to reduce the cost of replacements. Given a set of centroids $\mathcal{C}$ we maintain couple of matrices $CL_1^{R \times N}$ and $CL_2^{R \times N}$. $CL_1(i,j)$ and $CL_2(i,j)$ hold the first and second closest centroid for an extended centroid set $\mathcal{C}' = \mathcal{C} \cup r$ for the j-th point in the dataset (say r is the i-th element in $\mathcal{R}$). Before we start each iteration of the total KR replacements, we also create a list for each $r \in \mathcal{R}$, which holds the points closest to r, given centroid set C'. We call the list set RL. Now we look at how we reassign the points in $\mathcal{I}_1$ and $\mathcal{I}_2$ below when a centroid replacement is made.

- **For $\mathcal{I}_1$:** For each $I \in \mathcal{I}_1$, we have to see what is the second closest assignment given r and the other centroids except c. This can be done using matrices CL_1 and CL_2 in constant time for each image. So total computational cost for these set of points is $O(n_{avg})$.

- **For $\mathcal{I}_2$:** We already have set $\mathcal{I}_2$ from lists RL, which we compute before the replacements stage starts in each iteration. Although we have to remove some points from $\mathcal{I}_2$, which were in $\mathcal{I}_1$ as they have already been taken care of. Now for each $I \in \mathcal{I}_2$, we have to update corresponding old cost $d(I, c_{old})$ (c_{old} is the earlier assignment of I) with its new cost $d(I, r)$. Total computational cost for $\mathcal{I}_2$ is $O(n_{avg})$.

At the end of each iteration centroid set $\mathcal{C}$ will be updated. So we have to update CL_1, CL_2 and RL. But the computational cost to update these is $O(RN)$ because each element in CL_1 and CL_2 can be updated in constant time. List RL can be obtained from CL_1 in $O(RN)$ time. So the total computational cost for each iteration becomes $O(RN + KRn_{avg}) = O(RN)$ because we assume $N = Kn_{avg}$. So total computational cost for the complete algorithm becomes $O(TRN)$, which is $O(\frac{KN}{R})$ faster than the original PAM implementation.

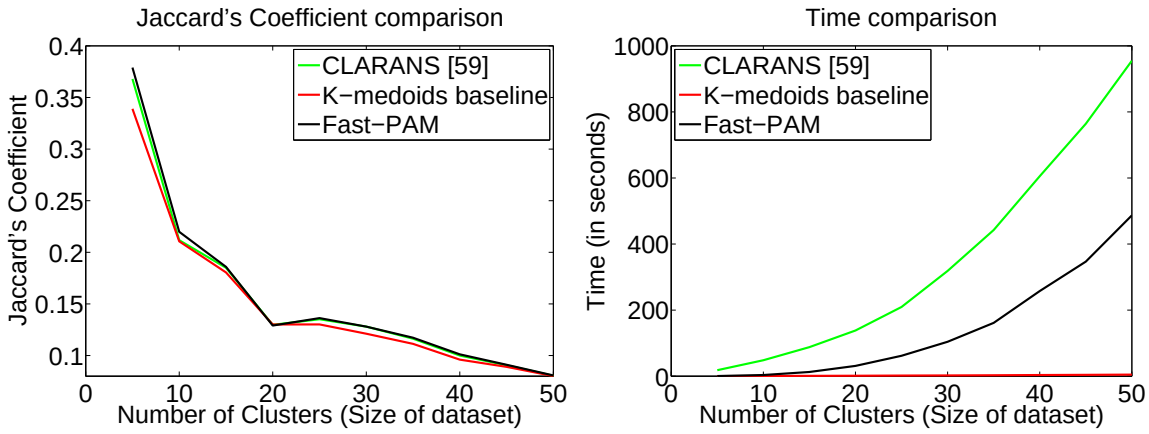
6.4.3 Experimental Results

We compare Fast-PAM with two other algorithms CLARANS [59] and a K-medoids baseline [56] (similar to K-means but in the update step we calculate the medoid instead of the mean) for performance and running time. We note that we can also use a restricted set in original PAM; complexity is $O(TKRN)$. If we use the same restricted set (used in

Fast-PAM) in PAM, Fast-PAM produces the exact same result as that of PAM but $O(K)$ faster. So we do not run PAM with a restricted set in our experiments. Also PAM without a restricted set (complexity is $O(TKN^2)$) is too slow to run for these datasets. Fast-PAM is $O(\frac{KN}{R})$ faster than PAM without a restricted set, i.e., the original PAM implementation [44].

We use several subsets of the face image dataset Pubfig [49]. We vary the number of clusters from 5 to 50 with each cluster containing 200 images. Jaccard's Coefficient (JCC) [68] is used as the evaluation metric. Each algorithm is run 20 times and mean JCC and time are plotted in Figure 2.

Although CLARANS produces results comparable to Fast-PAM, it requires a couple of parameters ($numlocal$ and $maxneighbour$) to be set. We used $numlocal = 100$ and $maxneighbour = 1000$ (selected after trying several sets) for CLARANS. Although the K-medoids baseline algorithm is fast its performance is not on a par with Fast-PAM. The difference between Fast-PAM and K-medoids baseline is more pronounced towards the beginning of the performance curve (Figure 2a). As the size of the dataset increases, clustering performance degrades for all the algorithms. So the difference in JCC between Fast-PAM and K-medoids is not very significant. However we see that as the number of clusters increases Fast-PAM tends to produce better ($2 - 3\%$ lower) local minimum of $\mathbf{Cost}$ than produced by the K-medoids baseline. For example, in the dataset with 50 clusters and 10000 images, the value of local minimum achieved by Fast-PAM is 34007 compared to 34873 achieved by K-medoids baseline. Overall, we feel that Fast-PAM is comparable with CLARANS but better than the K-medoids baseline in terms of performance for complete clustering.



(a) JCC Comparison

(b) Time Comparison

Figure 2: Performance and time comparison of Fast-PAM with other algorithms.

Publications

- **Journals:**

  1. **"Active Image Clustering with Pairwise Constraints from Humans";** Arijit Biswas, David Jacobs; *International Journal of Computer Vision (IJCV), 2014.*
  2. **"Active Subclustering";** Arijit Biswas, David Jacobs; *Computer Vision and Image Understanding (CVIU), Elsevier Press, 2014.*

- **Conferences:**

  1. **"Distance Learning Using the Triangle Inequality for Semi-supervised Clustering";** Arijit Biswas, David Jacobs; *European Conference on Computer Vision (ECCV), 2014, Submitted.*
  2. **"Simultaneous Active Learning of Classifiers & Attributes via Relative Feedback";** Arijit Biswas, Devi Parikh; *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2013.*
  3. **"Active Image Clustering: Seeking Constraints from Humans to Complement Algorithms";** Arijit Biswas, David Jacobs; *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2012.*
  4. **"Leafsnap: A Computer Vision System for Automatic Plant Species Identification";** Neeraj Kumar, Peter Belhumeur, Arijit Biswas, David Jacobs, W. John Kress, Ida Lopez, Joao Soares; *European Conference in Computer Vision (ECCV), 2012. (Oral presentation)*
  5. **"Odd Leaf Out: Improving visual recognition with games ";** Derek Hansen, David Jacobs, Darcy Lewis, Arijit Biswas, Jennifer Preece, Dana Rotman, Eric Stevens; *Third IEEE International Conference on Social Computing (Socialcom), 2011.*

- **Workshops:**

  1. **"Large Scale Image Clustering with Active Pairwise Constraints";** Arijit Biswas, David Jacobs; *Combining Learning Strategies to Reduce Label Cost: International Conference in Machine Learning (ICML) Workshop, 2011.*

- **Demos:**

  1. **"Relative Attributes for Enhanced Human-Machine Communication";** Naman Agarwal, Arijit Biswas, Adriana Kovashka, Kristen Grauman, Devi Parikh; *IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 2013.*

Bibliography

[1] http://vis-www.cs.umass.edu/lfw/.

[2] http://www.dabi.temple.edu/ shape/mpeg7/results.html.

[3] D. Angluin. Queries and concept learning. *Machine Learning*, 2(4):319–342, 1987.

[4] E. Bair. Semi-supervised clustering methods. *CoRR*, abs/1307.0252, 2013.

[5] S. Basu, A. Banerjee, and R. J. Mooney. Semi-supervised clustering by seeding. In C. Sammut and A. G. Hoffmann, editors, *ICML*, pages 27–34. Morgan Kaufmann, 2002.

[6] S. Basu, A. Banerjee, and R. J. Mooney. Active semi-supervision for pairwise constrained clustering. In *Fourth SIAM ICDM*, 2004.

[7] S. Basu, M. Bilenko, and R. J. Mooney. A probabilistic framework for semi-supervised clustering. In *ACM SIGKDD*, 2004.

[8] S. Basu, I. Davidson, and K. Wagstaff. *Constrained Clustering: Advances in Algorithms*. Data Mining and Knowledge Discovery Series. IEEE Computer Society, 2008.

[9] P. Belhumeur. Personal communication with author. 2011.

[10] A. Bellet, A. Habrard, and M. Sebban. A survey on metric learning for feature vectors and structured data. *CoRR*, abs/1306.6709, 2013.

[11] S. Belongie, J. Malik, and J. Puzicha. Shape context: A new descriptor for shape matching

and object recognition. In T. K. Leen, T. G. Dietterich, and V. Tresp, editors, *NIPS*, pages 831–837. MIT Press, 2000.

[12] T. Berg and P. N. Belhumeur. How do you tell a blackbird from a crow? In *ICCV*, pages 9–16. IEEE, 2013.

[13] A. Biswas and D. Jacobs. Large scale image clustering with active pairwise constraints. *International Conference in Machine Learning 2011 Workshop on Combining Learning Strategies to Reduce Label Cost*.

[14] A. Biswas and D. W. Jacobs. Active image clustering: Seeking constraints from humans to complement algorithms. In *CVPR*. IEEE, 2012.

[15] I. Borg and P. Groenen. *Modern multidimensional scaling, theory and applications*. Springer - Verlag , New York, 1997.

[16] S. Branson, P. Perona, and S. Belongie. Strong supervision from weak annotation: Interactive training of deformable part models. In D. N. Metaxas, L. Quan, A. Sanfeliu, and L. J. V. Gool, editors, *ICCV*, pages 1832–1839. IEEE, 2011.

[17] S. Branson, C. Wah, F. Schroff, B. Babenko, P. Welinder, P. Perona, and S. Belongie. Visual recognition with humans in the loop. In K. Daniilidis, P. Maragos, and N. Paragios, editors, *ECCV (4)*, volume 6314 of *Lecture Notes in Computer Science*, pages 438–451. Springer, 2010.

[18] G. Celeux, F. Forbes, and N. Peyrard. EM procedures using mean field-like approximations for markov model-based image segmentation. *Pattern Recognition*, 2003.

[19] R. Chellappa and A. K. Jain. *Markov Random Fields: Theory and Applications*. Academic Press, 1993.

[20] R. G. Cinbis, J. Verbeek, and C. Schmid. Unsupervised metric learning for face identification in TV video. 2011.

[21] T. Cormen, C. Leiserson, and R. Rivest. *Introduction to Algorithms*. MIT Press, 1990.

[22] C. K. Dagli, S. Rajaram, and T. S. Huang. Utilizing information theoretic diversity for SVM active learn. In *ICPR*, pages II: 506–511, 2006.

[23] I. Davidson and S. S. Ravi. Using instance-level constraints in agglomerative hierarchical clustering: theoretical and empirical results. *Data Min. Knowl. Discov*, 18(2):257–282, 2009.

[24] J. V. Davis, B. Kulis, P. J. 0002, S. Sra, and I. S. Dhillon. Information-theoretic metric learning. In *ICML*, 2007.

[25] B. Erol and F. Kossentini. A robust distance measure for the retrieval of video objects. In *SSIAI*, pages 40–44, 2002.

[26] M. R. Everingham, J. Sivic, and A. Zisserman. Hello! my name is buffy: Automatic naming of characters in TV video. In *BMVC*, 2006.

[27] R. Farrell, O. Oza, N. Zhang, V. I. Morariu, T. Darrell, and L. S. Davis. Birdlets: Subordinate categorization using volumetric primitives and pose-normalized appearance. In D. N. Metaxas, L. Quan, A. Sanfeliu, and L. J. V. Gool, editors, *ICCV*, pages 161–168. IEEE, 2011.

[28] P. F. Felzenszwalb, D. McAllester, and D. Ramanan. A discriminatively trained, multiscale, deformable part model. In *CVPR*, pages 1–8, 2008.

[29] O. Freifeld and M. J. Black. Lie bodies: A manifold representation of 3D human shape. In A. W. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, editors, *ECCV (1)*, volume 7572 of *Lecture Notes in Computer Science*, pages 1–14. Springer, 2012.

[30] R. Gomes, P. Welinder, A. Krause, and P. Perona. Crowdclustering. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. C. N. Pereira, and K. Q. Weinberger, editors, *NIPS*, pages 558–566, 2011.

[31] N. Grira, M. Crucianu, and N. Boujemaa. Active semi-supervised fuzzy clustering for image database categorization. In *Multimedia Information Retrieval*, pages 9–16. ACM, 2005.

[32] Y. Guo and R. Greiner. Optimistic active-learning using mutual information. In M. M. Veloso, editor, *IJCAI*, pages 823–829, 2007.

[33] G. Gupta and J. Ghosh. Bregman bubble clustering: A robust framework for mining dense clusters. *TKDD*, 2008.

[34] T. Hastie and R. Tibshirani. Discriminant analysis by Gaussian mixtures. *Journal of the Royal Statistical Society series B*, 58:158–176, 1996.

[35] S. C. H. Hoi, R. Jin, and M. R. Lyu. Learning nonparametric kernel matrices from pairwise constraints. In *ICML*, 2007.

[36] A. D. Holub, P. Perona, and M. C. Burl. Entropy-based active learning for object recognition. In *Online Learning for Classification Workshop*, pages 1–8, 2008.

[37] R. Huang and W. Lam. An active learning framework for semi-supervised document clustering with language modeling. *Data Knowl. Eng*, 68(1), 2009.

[38] R. Huang, W. Lam, and Z. Zhang. Active learning of constraints for semi-supervised text clustering. In *SDM*. SIAM, 2007.

[39] D. W. Jacobs, D. Weinshall, and Y. Gdalyahu. Classification with nonmetric distances: Image retrieval and class representation. *IEEE Trans. Pattern Anal. Mach. Intell*, 2000.

[40] A. K. Jain. Data clustering: 50 years beyond K-means. *Pattern Recognition Letters*, 31(8), 2010.

[41] P. Jain and A. Kapoor. Active learning for large multi-class problems. In *CVPR*, pages 762–769, 2009.

[42] A. J. Joshi, F. Porikli, and N. Papanikolopoulos. Breaking the interactive bottleneck in multi-class classification with active selection and binary feedback. In *CVPR*, pages 2995–3002.

IEEE, 2010.

[43] A. Kapoor, E. Horvitz, and S. Basu. Selective supervision: Guiding supervised learning with decision-theoretic active learning. In M. M. Veloso, editor, *IJCAI*, pages 877–882, 2007.

[44] L. Kaufman. Finding groups in data: an introduction to cluster analysis. In *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley, New York, 1990.

[45] D. Klein, S. D. Kamvar, and C. D. Manning. From instance-level constraints to space-level constraints: Making the most of prior knowledge in data clustering. In *ICML*, 2002.

[46] Kruskal, J. B. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proc. of the AMS*, 2:48–50, 1956.

[47] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. C. Lopez, and J. V. B. Soares. Leafsnap: A computer vision system for automatic plant species identification. In *ECCV, 2012*.

[48] N. Kumar, P. N. Belhumeur, A. Biswas, D. W. Jacobs, W. J. Kress, I. C. Lopez, and J. V. B. Soares. Leafsnap: A computer vision system for automatic plant species identification. *ECCV (2)*, 7573:502–516, 2012.

[49] N. Kumar, A. C. Berg, P. N. Belhumeur, and S. K. Nayar. Attribute and simile classifiers for face verification. In *ICCV*. IEEE, 2009.

[50] Y. J. Lee and K. Grauman. Learning the easy things first: Self-paced visual category discovery. In *CVPR*, pages 1721–1728. IEEE, 2011.

[51] K. Levi, M. Fink, and Y. Weiss. Learning from a small number of training examples by exploiting object categories. In *IEEE Workshop on Learning in Computer Vision and Pattern Recognition*, page 96, 2004.

[52] H. B. Ling and D. W. Jacobs. Shape classification using the inner-distance. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 29(2):286–299, Feb. 2007.

[53] Z. Lu and H. H.-S. Ip. Constrained spectral clustering via exhaustive and efficient constraint propagation. In *ECCV*. Springer, 2010.

[54] Z. D. Lu and M. A. C. Perpinan. Constrained spectral clustering through affinity propagation. In *CVPR*, 2008.

[55] Y. M. Lui and J. R. Beveridge. Grassmann registration manifolds for face recognition. In *ECCV*, pages II: 44–57, 2008.

[56] J. MacQueen. Some methods for classification and analysis of multivariate observations. In *5th Berkeley Symp. on Mathematics Statistics and Probability*, 1967.

[57] P. K. Mallapragada, R. Jin, and A. K. Jain. Active query selection for semi-supervised clustering. In *ICPR*, 2008.

[58] A. Martinez and R. Benavente. The AR Face Database. *CVC Technical Report #24*, 1998.

[59] R. T. Ng and J. Han. CLARANS: A method for clustering objects for spatial data mining. *IEEE Trans. Knowl. Data Eng*, 14(5):1003–1016, 2002.

[60] H.-S. Park and C.-H. Jun. A simple and fast algorithm for K-medoids clustering. *Expert Syst. Appl*, 36(2):3336–3341, 2009.

[61] F. Petralia, V. Rao, and D. B. Dunson. Repulsive mixtures. In P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *NIPS*, pages 1898–1906, 2012.

[62] K. Punera and J. Ghosh. Consensus-based ensembles of soft clusterings. *Applied Artificial Intelligence*, 22(7&8):780–810, 2008.

[63] A. Quattoni and A. Torralba. Recognizing indoor scenes. In *CVPR*, pages 413–420, 2009.

[64] F. R. Schmidt, M. Clausen, and D. Cremers. Shape matching by variational computation of geodesics on a manifold. In *DAGM*, pages 142–151, 2006.

[65] B. Settles. Active learning literature survey. Technical report, 2010.

[66] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transactions on Pattern*

Analysis and Machine Intelligence, 22:888–905, 1997.

[67] B. Siddiquie and A. Gupta. Beyond active noun tagging: Modeling contextual interactions for multi-class active learning. In *CVPR*, pages 2979–2986. IEEE, 2010.

[68] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining*. Addison-Wesley, 2005.

[69] Y. D. Tian, W. Liu, R. Xiao, F. Wen, and X. Tang. A face annotation framework with partial clustering and interactive labeling. In *CVPR*, pages 1–8, 2007.

[70] S. Vijayanarasimhan and K. Grauman. Large-scale live active learning: Training object detectors with crawled data and crowds. In *CVPR*, pages 1449–1456. IEEE, 2011.

[71] S. Vijayanarasimhan and K. Grauman. Active frame selection for label propagation in videos. In A. W. Fitzgibbon, S. Lazebnik, P. Perona, Y. Sato, and C. Schmid, editors, *ECCV (5)*, volume 7576 of *Lecture Notes in Computer Science*, pages 496–509. Springer, 2012.

[72] S. Vijayanarasimhan, P. Jain, and K. Grauman. Far-sighted active learning on a budget for image and video recognition. In *CVPR*, pages 3035–3042. IEEE, 2010.

[73] N. Vretos, V. Solachidis, and I. Pitas. A mutual information based face clustering algorithm for movie content analysis. *Image Vision Comput*, 2011.

[74] K. Wagstaff, C. Cardie, S. Rogers, and S. Schroedl. Constrained K-means clustering with background knowledge. In *ICML*, pages 577–584, 2001.

[75] X. Wang and I. Davidson. Active spectral clustering. In *ICDM*. IEEE Computer Society, 2010.

[76] B. Wu, Y. Zhang, B.-G. Hu, and Q. Ji. Constrained clustering and its application to face clustering in videos. In *CVPR*, 2013.

[77] E. P. Xing, A. Y. Ng, M. I. Jordan, and S. J. Russell. Distance metric learning with application to clustering with side-information. In *NIPS*, 2002.

[78] C. Xiong, D. Johnson, and J. J. Corso. Spectral active clustering via purification of the k-nearest neighbor graph. In *ECDM*, 2012.

[79] Q. Xu, M. desJardins, and K. Wagstaff. Active constrained clustering by examining spectral eigenvectors. In *Discovery Science*, volume 3735. Springer, 2005.

[80] M.-S. Yang and K.-L. Wu. A similarity-based robust clustering method. *IEEE Trans. Pattern Anal. Mach. Intell*, 26(4), 2004.

[81] M. L. Zhu and A. M. Martinez. Subclass discriminant analysis. *IEEE Trans. Pattern Analysis and Machine Intelligence*, 28(8):1274–1286, Aug. 2006.

[82] J. Zhuang, I. W. Tsang, and S. C. H. Hoi. SimpleNPKL: simple non-parametric kernel learning. In *ICML*, 2009.

[83] J. Y. Zou and R. P. Adams. Priors for diversity in generative latent variable models. In P. L. Bartlett, F. C. N. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *NIPS*, pages 3005–3013, 2012.