LABELING HYPERGRAPH-STRUCTURED DATA USING MARKOV NETWORK

BY TOUFIQ U PARAG

A dissertation submitted to the Graduate School—New Brunswick
Rutgers, The State University of New Jersey
in partial fulfillment of the requirements for the degree of Doctor of Philosophy
Graduate Program in Computer Science
Written under the direction of Dr Ahmed Elgammal
and approved by

----------------------------------

----------------------------------

----------------------------------

----------------------------------

New Brunswick, New Jersey
January, 2011
ABSTRACT OF THE DISSERTATION

Labeling Hypergraph-structured Data using Markov Network

by Toufiq U Parag

Dissertation Director: Dr Ahmed Elgammal

The goal of this dissertation is to label datapoints into two groups utilizing higher order information among them. More specifically, given likelihood (or error) measures that subsets of data are generated by a pattern (or belong to a class), we wish to label the individual datapoints into two classes. Several computer vision problems deal with data in this format. In model estimation, small subsets of data are randomly sampled to produce an error measure. Groups of object parts are often used to provide useful geometrical information for object recognition/localization.

We propose a novel labeling algorithm by modeling the datapoints as nodes of a higher order Markov Network. A new higher order clique function plays the central role in our method. The behavior of this clique function is analyzed to explain how it affects the inference and to predict when it is theoretically guaranteed to produce an optimal solution. We also describe how a parametric form of the proposed clique function can be learned from labeled data. Results on several different computer vision problems will be presented to demonstrate the effectiveness of the proposed algorithm.
I gratefully acknowledge the guidance and generous support of my supervisor, Professor Ahmed Elgammal, throughout my PhD study. His advice and support helped me overcome various obstacles during this period. I am thankful for his contribution to this dissertation as well as all my other studies.

I would like to thank Professor Vladimir Pavlovic for the fruitful discussion we had about parameter learning. I also highly appreciate his suggestions regarding the key components of the proposed framework.

I thankfully acknowledge the assistance from Professor Michael Grigoriadis to apply linear programming relaxation for general inference problems.

I also thank Professor Dimitris Metaxas and Dr Maneesh Singh of Siemens Corporate Research for very useful comments to improve the quality of this thesis.
Dedication

This work is dedicated to my parents, Nazrul Islam and Selina Begum, and to my wife Tasmina Ahmed.

It is because of your support and inspiration I could come this far.
# Table of Contents

Abstract ................................................................. ii
Acknowledgements ....................................................... iii
Dedication ............................................................... iv

List of Tables ........................................................... x
List of Figures .......................................................... xi

1. Introduction ......................................................... 1

2. Background and Related Work ................................. 7  
   2.1. Hypergraph model ............................................. 7  
       2.1.1. Definition .............................................. 7  
       2.1.2. Application in Vision ................................. 8  
       2.1.3. Spectral Graph Clustering ............................ 9  
       2.1.4. Hypergraph Clustering ............................... 10  
   2.2. Markov Network models .................................... 11  
       2.2.1. Markov Random Field (MRF) ......................... 11  
       2.2.2. Conditional Random Field (CRF) .................... 13  
       2.2.3. Inference for Pairwise Markov Models .......... 14  
       2.2.4. Inference for Higher Order Markov Models .... 14  
       2.2.5. Markov Models in Vision ............................ 14  
   2.3. Factor Graphs ................................................ 16  
   2.4. Model Estimation Problems .............................. 17  

3. Overall Framework ............................................. 19
3.1. Problem Description ................................................. 19
3.2. Graphical Model Framework ........................................ 20
3.3. Proposed Clique Function .......................................... 22
3.4. Analysis of the Clique Function ................................. 24
   3.4.1. Submodularity and Tractability .............................. 24
       Definition of Submodular Function ............................. 25
   3.4.2. Energy function as submodular function: .................. 25
   3.4.3. Submodularity of $E^k(X^k; V^k)$ ......................... 26
3.5. Effect of Penalty Functions on Inference ...................... 27
   3.5.1. Linear $g_c$ .................................................. 27
   3.5.2. Nonlinear $g_c$ ............................................. 29
4. Inference and Learning ............................................ 31
   4.1. Inference ....................................................... 31
       4.1.1. Inference for Linear $g_c$ ............................... 31
       4.1.2. Inference with Nonlinear $g_c$ ......................... 32
       Linear Programming Relaxation ............................... 33
       Sum-Product Belief Propagation ............................... 34
   4.1.3. Parameter Selection ....................................... 36
   4.2. Learning ...................................................... 36
       4.2.1. Learning Penalty Functions ............................... 36
       Discrete $g_c$ .................................................. 37
       Second order polynomial $g_c$ ............................... 37
       4.2.2. Learning algorithms ..................................... 38
       Estimation by likelihood maximization ....................... 38
       Estimation by Margin maximization .......................... 39
5. Difference and Similarities with Relevant works ............ 40
   5.1. Difference with Hypergraph Clustering ...................... 40
       5.1.1. Objective Function ...................................... 41
Appendix A. Boosting Adaptive Linear Weak Classifiers for Online learning and Tracking

A.1. Introduction ................................................. 85
A.2. Background and Related Work .................................. 88
  A.2.1. Offline Boosting ............................................. 88
  A.2.2. Online boosting .............................................. 89
A.3. Adaptive Linear Weak Classifier ................................ 89
  A.3.1. Weighted Linear Regressor ...................................... 89
  A.3.2. Adaptive Linear Regressor ......................................... 90
  A.3.3. Weak Classifier Memory ........................................... 91
  A.3.4. Temporal weighting ............................................. 92
A.4. Combining adaptive linear regressors for online boosting ......... 92
A.5. Application to tracking ........................................... 95
A.6. Experiments and Results ......................................... 95
  A.6.1. Synthetic data .................................................. 95
  A.6.2. UCI datasets .................................................. 97
  A.6.3. Tracking examples .............................................. 97
A.7. Conclusion ..................................................... 99

Appendix B. Unsupervised Learning of Boosted Tree Classifier using Graph Cuts for Hand Pose Recognition

B.1. Introduction .................................................. 102
  B.1.1. Related Work ................................................ 103
B.2. Overview of the approach ........................................ 104
B.3. Unsupervised Learning of Tree Classifier ........................... 105
  B.3.1. Normalized Cut Clustering ...................................... 105
  B.3.2. One-pass Learning of weak classifiers ......................... 106
  B.3.3. Soft Margin AdaBoost ............................................ 108
B.4. Experiment and Results ......................................... 109
# List of Tables

5.1. Pointwise costs for toy problem for which existing higher order MRF fails. . . 46
5.2. Groupwise weights for toy problem for which existing higher order MRF fails. 47
6.1. Performance comparison for Model Estimation. . . . . . . . . . . . . . . . . . 56
6.2. Performance comparison for minimizing false positives. . . . . . . . . . . . . . 60
7.1. Matching performance comparison: predefined and learned penalty . . . . . . . 71
8.1. Number of accurately detected edges and FA for object localization . . . . . . 79
A.1. Algorithm: Boosting with Linear Adaptive Classifier. . . . . . . . . . . . . . . 94
A.2. Classification rates on synthetic and UCI data . . . . . . . . . . . . . . . . . . 97
A.3. Frames exactly tracked by the proposed & other methods . . . . . . . . . . . . 99
B.1. Recognition accuracies of the decision tree with different classifiers . . . . . . 112
### List of Figures

<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Example Hypergraph</td>
<td>8</td>
</tr>
<tr>
<td>2.2</td>
<td>Hypergraph matching example</td>
<td>9</td>
</tr>
<tr>
<td>2.3</td>
<td>Example Pairwise MRF</td>
<td>12</td>
</tr>
<tr>
<td>2.4</td>
<td>MRF model for binary image segmentation</td>
<td>15</td>
</tr>
<tr>
<td>2.5</td>
<td>Factor Graph</td>
<td>17</td>
</tr>
<tr>
<td>2.6</td>
<td>Line estimation from noisy data</td>
<td>18</td>
</tr>
<tr>
<td>3.1</td>
<td>Problem description</td>
<td>20</td>
</tr>
<tr>
<td>3.2</td>
<td>Neighborhood definition</td>
<td>21</td>
</tr>
<tr>
<td>3.3</td>
<td>Visual illustration of clique function: Linear penalty</td>
<td>28</td>
</tr>
<tr>
<td>3.4</td>
<td>Visual illustration of clique function: Non-linear penalty</td>
<td>29</td>
</tr>
<tr>
<td>5.1</td>
<td>A toy example for two line estimation</td>
<td>42</td>
</tr>
<tr>
<td>5.2</td>
<td>MRF applications emphasizes on pointwise information</td>
<td>44</td>
</tr>
<tr>
<td>5.3</td>
<td>Clique potential comparison</td>
<td>45</td>
</tr>
<tr>
<td>5.4</td>
<td>Difference in RANSAC type algorithm.</td>
<td>51</td>
</tr>
<tr>
<td>6.1</td>
<td>Line estimation</td>
<td>54</td>
</tr>
<tr>
<td>6.2</td>
<td>Error distribution for fundamental matrix approximation</td>
<td>57</td>
</tr>
<tr>
<td>6.3</td>
<td>Worst case qualitative performance comparison-1</td>
<td>58</td>
</tr>
<tr>
<td>6.4</td>
<td>Worst case qualitative performance comparison-2</td>
<td>59</td>
</tr>
<tr>
<td>6.5</td>
<td>Circle/Ellipse detection</td>
<td>61</td>
</tr>
<tr>
<td>7.1</td>
<td>Weight computation for matching</td>
<td>64</td>
</tr>
<tr>
<td>7.2</td>
<td>Matching performance comparison: proposed and other methods</td>
<td>67</td>
</tr>
<tr>
<td>7.3</td>
<td>Qualitative matching result-1</td>
<td>68</td>
</tr>
<tr>
<td>7.4</td>
<td>Qualitative matching result-2</td>
<td>69</td>
</tr>
<tr>
<td>7.5</td>
<td>Learned penalty function</td>
<td>70</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

The objective of this dissertation is to label datapoints into two classes when the information about the datapoints is available in terms of small groups\(^1\) of data. The information encode the likelihood that all the members of the corresponding group should fall into the same class. These likelihood measures are generated utilizing a model, either analytical or implicit, of the entity or pattern we are trying to decide about. Given the subsets and their corresponding likelihood measures, we wish to find the labels of data having maximal agreement with these measures.

It should be noted here that we are interested in scenarios where higher order knowledge about data subsets of sizes \(k > 2\) is the primary source of information for labeling; we may or may not have any information about the individual data sample or pairs of samples. Furthermore, we may have data subsets of different sizes, i.e., \(k = 3, 4, \ldots\), each with corresponding likelihood measure, to decide the individual label from.

It is necessary to understand what is meant by models that generate the weights for the subsets of each class. In this study, a model implies a formula, either closed form or algorithmic, that quantifies any property that a group of datapoints of certain class should possess. For example, if it is known (or believed) that subset of examples should obey a certain geometric structure, then a model is the analytical formula that captures the geometric property (area, perimeter etc.) On the other hand, if the subset should match a specific predefined template, a model corresponds to the probabilistic technique to determine the similarity between the template and the subset.

An interesting instance of the problem discussed here is when two classes are defined based

\(^1\)Group simply implies a collection of datapoints. The mathematical definition of group is not used in this paper. Similarly, likelihood simply implies a numerical weight.
on a single model. With a single model, one of the two classes can be defined to comprise all datapoints that satisfy the model and the other class to comprise all the examples that do not. In this case, the weight for the former class is computed from the model fitting error and assumed to be normalized. As a consequence, the likelihood for the other class can be computed by subtracting the weight from 1.

For the single model case, there is a direct connection between the problem we are dealing with and the hypergraph node labeling problem. A Hypergraph is a generalization of a graph where each hyperedge connects more than two vertices, i.e., a hyperedge is a subset of nodes. Weighted hypergraphs, a hypergraph whose hyperedges are associated with real valued weights, have been recently gained much popularity in computer vision [56]. Each small group of data, as discussed above, can be considered as a hyperedge of some hypergraph. The likelihood weight for this group of data is analogous to the weight of corresponding hyperedge. Given such a hypergraph, a hypergraph node labeling algorithm produces a label configuration for its nodes to optimally conform with the weights computed given the model.

There are many examples of such problem in machine learning and vision literature. In part-based object recognition for computer vision, we may learn an implicit statistical model among groups of detected parts to decide which of the detected parts actually belong to the object we are trying to recognize. It is well known that larger groups of parts capture more geometrical information than pairs of parts. This is a necessary step in recognition due to the fact that object part-detectors often generate many false alarms. In website classification problems, a subset comprising more than two words could be more informative about the category than that comprising one or two words.

Subset-wise information is also utilized in model estimation algorithms. The problem of model estimation is to determine the parameters of a model given a set of (noisy) data and an analytic form of a model. A standard algorithm, namely RANSAC [33], for this sort of problem randomly samples smaller subsets of data, computes target model parameters and calculates an error value for each of the subsets. RANSAC is used for fundamental matrix computation, affine motion estimation etc in computer vision [131] literature.

We propose a novel approach to solve this labeling problem in a principled fashion. Each
datapoint is associated with a binary label variable. The label variables (and their dependencies) are modeled by an Undirected Graphical Model or Markov Network with appropriate neighborhood system and clique definitions (to be explained later). To compute the optimal configuration of the label variables, it is necessary to minimize an overall cost function which is a summation of all cost functions defined on the cliques.

We propose a class of potential function for each clique which plays the central role in deciding labels of data samples. The clique function is defined as a weighted combination of penalty functions, one for each class, that attempt to label the data subset to its corresponding classes. As a result, the overall cost function becomes a summation of two types of competing penalty functions for each subsets, each trying to label the samples into its corresponding class. We analyze what characteristics of this clique function enforce the output label configuration to optimally conform with the likelihood measure. The behavior of the clique function with different forms of penalty functions is explained to assist designing a problem specific one.

From a theoretical point of view, we investigate what makes the proposed potential function submodular and therefore is guaranteed to produce an accurate solution for two-class problems [20, 125, 74]. For a specific form of this potential function, the inference problem can be solved with a simple algorithm that runs in linear time w.r.t. the number of data samples and subsets. Standard methods for inference, e.g., linear programming relaxation or sum product belief propagation algorithms, can be used for other forms of potential functions. This Markov Network formulation is able to cope with the situation when the likelihood values are available for different size subsets.

This dissertation also investigates learning the clique (more specifically, penalty) functions from labeled data. It is not difficult to realize that designing a clique function to work successfully for all possible instances of the problem is not an easy task (in fact, may even not be feasible). We show how existing methods for learning log-linear models can be utilized to learn problem specific clique function for labeling.

Finally, it is worth mentioning here we do not exploit any property of the model itself for labeling. The sole purpose a model (for each class) in this framework is to compute the weights; the only assumption we make about the model is that it can correctly capture the property that a data subset should possess for each class. Once the likelihood measures are computed, our
The proposed method is a much more general technique than model estimation algorithms, e.g., RANSAC and its variants. RANSAC family algorithms require a (global) model for the whole entity or object to be estimated from data. These algorithms are not applicable where there exists only models defined over local parts. The proposed algorithms can work with both global and local models, in the latter case the dependence among the datapoints is utilized to result in an optimal label configuration. Our algorithm can solve any model estimation problem that RANSAC can handle and can also solve other problems where RANSAC can not be applied.

The type of problems that proposed method addresses is different from what hypergraph clustering methods are designed for. In hypergraph clustering, each hyperedge is usually associated with a weight computed from the similarity of the member datapoints in some feature space. On the other hand, in our framework, the hyperedges are associated with a likelihood value, calculated given a model. Conceptually, the most significant difference lies in the objective function the proposed method and the clustering methods optimize. For example, spectral graph clustering methods [1, 48, 140] minimize a product of cut function and a restraint function defined over the associativity or size of each cluster. As a result, these techniques are prone to find ‘balanced’ clusters, i.e., the partitioning will be substantially influenced by the associativity of each cluster. In contrast, our method minimizes a summation of weighted combinations of competing penalty functions.

Higher order Markov Network models also aim to label datapoints using higher order information in a fundamentally different manner. Once again, existing higher order Markov models are different from our method in terms of what they aim to achieve and how they achieve it. Most of the existing higher order models rely primarily on pointwise information (information about each datapoint) for labeling. The pairwise and higher order clique potential functions are treated as smoothness functions in order to enforce spatial consistency over the regions. In the absence of pointwise information, these models will produce trivial solution (all 0 or all 1). Hence, despite being very successful in the problems they are designed for, they are not suitable for situations where we only have higher order information about the data, e.g. model estimation from noisy data. On the other hand, the proposed higher-order clique function is
not a smoothing function and is able to produce useful labeling without pointwise and pairwise information. The proposed method is designed to work primarily in scenarios where we only have higher order information available to decide from. These differences will be illustrated in more technical details in Chapters 2 and 5.

The proposed method can be considered as an instance of Factor graphs proposed in [77] (readers unfamiliar with Factor graphs are referred to next chapter) with compatible definition of factors. This dissertation shows how can factor graphs, with the appropriately defined components, can be applied for a large number of vision problems that have not been shown before.

There is an interesting connection between the proposed framework and probabilistic grammars. In contrast to traditional logic, where a rule can be either true or false, probabilistic grammars associate a weight or confidence to each rule. In a recent work [116], the authors have proposed a probabilistic logical framework to represent the relationship among the parts of an object. This study uses two numerical weights for each rule: the confidence that the rule is true (evidence for) and that the rule is false (evidence against). These weights are similar to the likelihood measures for each model in the proposed framework. The inference in [116] is carried out by combining the outcome of rule evaluation probabilistically using a bi-lattice based framework for all possible combinations of data sample values. Because one needs to evaluate each rule of bi-lattice based framework for all combinations of data, the method of [116] becomes impractical when the number of observations becomes large. It can not decide based on a restricted number of data subsets, e.g., as we are given in model estimation scenario. That is why [116] is only applicable with a few candidate detections for a small number of parts in object detection/recognition.

The proposed method has been applied to a wide range of problems in vision. We show promising results of our algorithms in model estimation, object localization and feature correspondence problems. None of the aforementioned methods can address all these problems. Thus, this study also enhances the scope of problems Markov Network based models can tackle and will motivate researchers in this community to work towards more diversified set of problems.

This thesis is organized as follows. The next chapter will provide brief descriptions of
some previous works relevant to our algorithm. Chapter 3 explains the overall model of our
algorithm. This chapter also analyzes how the proposed clique function contribute to labeling
and discusses its properties in Section 3.4. We suggest what inference algorithm can be used
to solve the labeling problem in Section 4.1 of Chapter 4. Techniques for learning application
dependent penalty functions are also discussed in Section 4.2 of the same chapter. Chapter 5
illustrates the differences between the proposed method and existing higher order Markov mod-
els, hypergraph clustering algorithm and model estimation in a more conceptual and technical
level. The next three chapters, Chapters 6, 7, 8 describes three applications of this method on
model estimation, feature correspondence and object localization. Finally we summarize our
findings and conclude in Chapter 9.
Chapter 2

Background and Related Work

This chapter describes several methods that also utilize higher order information for data labeling or other purposes. Specifically, we familiarize the readers to Hypergraph applications, higher order Markov Network models, Factor graphs and model estimation techniques in order to assist them to connect the relevant aspects to the proposed method and also to realize the difference among them. For descriptions of previous works in different applications of the proposed method, e.g., model estimation, feature matching, object localization, we refer the readers to the chapters that describe these applications.

2.1 Hypergraph model

2.1.1 Definition

A hypergraph is an abstract mathematical model to represent the dependence among (small) subsets of data samples. As a generalization of graph structure, each observation is considered to be node to hypergraph. But, contrary to the graph definition where an edge embodies the relationship between a pair of nodes, the hyperedges of a hypergraph connect a small subset (usually of size larger than two) of nodes. A hyperedge weight quantifies the strength of this relationship. Formally, given $n$ datapoints $V = \{v_i \mid 1 \leq i \leq n\}$, a hyperedge is defined by a subset like $V^k = \{v_{i_1}, \ldots, v_{i_k}\}$ of size $k$, where $1 \leq i_l \leq n$, $1 \leq l \leq k$, and their corresponding weights $\omega(V^k)$. The size $k$ could be different for different subsets. In Figure 2.1, we show an illustrative example of a hypergraph with eight nodes $v_1, \ldots, v_8$ where the nodes are connected by hyperedges of sizes two, $\{v_3, v_4\}$, three, $\{v_1, v_2, v_3\}$ and four, $\{v_4, v_5, v_6, v_7\}$. 
An example hypergraph: $H = (V, E)$, where $V = \{v_1, \ldots, v_8\}$ are the nodes shown as cyan circles, and $E = \{\{v_1, v_2, v_3\}, \{v_3, v_4\}, \{v_4, v_5, v_6, v_7\}, \{v_1, v_7, v_8\}\}$ are the hyperedges shown in dotted blue polygons.

Figure 2.1: A sample hypergraph.

### 2.1.2 Application in Vision

Such a representation of the relationship among datapoints is utilized in several vision application to attain different goals. Examples of problems utilizing hypergraph structure include (partial) matching a hypergraph to another; labeling the nodes into two or more classes (e.g., clustering). The work of [86] builds a hypergraph representation among the object parts and relies on matching hypergraph representations for recognition. More recently, [137, 27] proposed two algorithms for hypergraph matching and applied them for establishing feature correspondences between two images. Figure 2.2 visually illustrates an example of hypergraph matching problem. The goal is to establish the correspondences between keypoint locations on the left image and those on the right image, as indicated in red lines in Figure 2.2. To attain this goal, hypergraph matching techniques assume a hypergraph structure among the keypoints on each of the images, indicated by a set of yellow triangles on both left and right images, and attempts of match the nodes of these two hypergraphs that are maximally consistent with the underlying spatial arrangement imposed by hyperedges.

A relatively larger number of studies focus on clustering hypergraphs that we describe in Section 2.1.4. Since most of these methods rely on spectral graph clustering methods, we will first introduce them briefly in Section 2.1.3. One very important instance of the problems that the proposed algorithm tackles is labeling the nodes of a hypergraph, but our method is fundamentally different from the traditional clustering and matching algorithms.
2.1.3 Spectral Graph Clustering

The problem of clustering is similar to the one we described in the sense that, given a dataset, clustering algorithms also aim at labeling the datapoints. Among many clustering algorithms, we emphasize on spectral clustering methods. Given pairwise similarities between pairs of data samples, a spectral graph clustering algorithm considers a graph among the samples where the similarity values are treated as weights of the edges connecting the two examples. This type of algorithm seeks to obtain the solution by minimizing product a cut and restraint function, where restraint function reflects the associativity or size of the corresponding cluster.

More formally, with a graph defined on examples \( V = \{ v_i \mid 1 \leq i \leq n \} \) and edges \( E^2 \subset V \times V \), let \( \omega_{ij} \) denote the weight of edge \( \{ v_i, v_j \} \). The goal of a clustering algorithm is to partition the datapoints into two (or more) clusters, i.e., assign the label \( x_i \) of example \( v_i \) to 1 or 0 depending on which cluster it belongs to. The spectral clustering method of [117] defines a cut and a restraint function as follows.

Cut Function: \[ f_{\text{cut}}(x) = \sum_{i,j} \delta(x_i, x_j) \omega_{ij} \] (2.1)

Restraint Function: \[ f_{\text{res}}(x) = \frac{1}{\sum_i x_i d_i} + \frac{1}{\sum_i (1 - x_i) d_i} \] (2.2)

where \( \delta(x_i, x_j) = 1 \) only when \( x_i = x_j \) and zero otherwise and \( d_i \) is defined as the degree of \( v_i : d_i = \sum_i \omega_{ij} \). The objective function that [117] minimizes is a product of cut
and constraint functions, \( \arg \min_x f_{\text{cut}}(x)f_{\text{res}}(x) \). While the cut function tries to minimize the combined weight of connections across the two partitions, the restraint function tries to increase the connectivity within each partition. As a result, these techniques are largely influenced by the associativity factor embedded in the objective function of spectral clustering and tend to produce ‘balanced’ clusters. Spectral clustering methods have used for image segmentation [117], object recognition [109] [21], image clustering [95] and also, in other domains such as text clustering [26].

### 2.1.4 Hypergraph Clustering

Several hypergraph clustering methods attempt to apply spectral graph clustering [117, 90] by either a transforming the hypergraph to a graph so that graph spectral clustering method can be applied or incorporating new quantities into the clustering methods to handle hypergraphs. For example, the studies of [1, 48] transforms hyper-connectivity into pairwise ones and applies the spectral clustering method. These studies do not address the issue of potential information loss due to this projection. On the other hand, Zhou et.al. [140] augment the spectral clustering algorithm with necessary quantities required to handle hypergraphs. Despite the extensions (or projections), the objective functions that these algorithms minimize, retain the characteristics of spectral clustering method, i.e., they also minimize the product of a cut and a restraint function. Hence, these methods are also strongly biased towards resulting in ‘balanced’ clusters.

Shashua et.al. [115] assumed independence among the labels of the datapoints, a contradiction to our notion that they are dependent given a neighborhood (to be explained in the next chapter). In [115], the authors use hyper-stochasticity as a balancing factor for hypergraphs. In VLSI design, the hmetis algorithm [64] for hypergraph partitioning performs a shrinking and expanding of hypergraph size so that a partitioning results in coarse level is projected back onto the original level. Hypergraph clustering have been used in vision for image segmentation [105] and video object segmentation [57].

Clustering methods were not designed for model dependent hyperedge weights. The proportion of observations satisfying a model can not be assumed to be balanced either. We will elaborate why clustering algorithms are not suitable for the task we are addressing using a motivating example in Section 5.1.
2.2 Markov Network models

Markov Network, also known as Undirected Graphical model, is a probabilistic graphical model designed to capture the relationship among datapoints to decide their hidden states (or labels). Markov Networks assume that the state or label of a datapoint is probabilistically dependent only to its ‘neighbors’. This assumption facilitates a factorization of joint probability of any label configuration to those of smaller subsets of labels. The definition of an Markov Network requires a neighborhood structure and a function that quantifies the joint probabilities of a subset of states. Out of several forms of Markov Networks models, two have particularly gained much importance in vision, namely Markov Random Field (MRF) and Conditional Random Field (CRF). Although these two terms are used synonymously in vision community, they are different in how they model the joint probability of label configuration.

2.2.1 Markov Random Field (MRF)

A Markov Random Field (MRF) is defined over the states or labels \(X = \{x_1, \ldots, x_n\}\) of \(n\) datapoints \(V = \{v_1, \ldots, v_n\}\). State \(x_i\) of each datapoint \(v_i\) are assumed to be dependent only those of \(v_i’s\) neighbors \(N_i\). Any data sample \(v_j\) can be defined as a neighbor of \(v_i\), i.e., \(v_j \in N_i\), as long as \(i \neq j\) and \(v_j \in N_i \Leftrightarrow v_i \in N_j\). Once the neighbors of all the datapoints are defined, the conditional probability \(p(x_i \mid X \setminus \{x_i\})\) given all other labels only depends on the conditional probability \(p(x_i \mid x_j \text{ where } v_j \in N_i)\) given only labels of its neighbors. Furthermore, we define a clique as subsets of neighboring datapoints that represent an ‘active’ interaction, that is, it is possible to select only some of the relationships among the neighboring datapoints to be effective. The joint probability of the label configuration depends on functions defined over these cliques.

Figure 2.3 shows a schematic diagram an example MRF structure defined on a grid. The smaller cyan circles on the right hand side of this diagram represent the datapoints and the relatively larger black (and yellow) circles are their corresponding labels. We assume a four-connectivity neighborhood over the spatial location of datapoints, as shown in dotted blue cross. This definition of neighborhood imposes a certain dependence among the labels of these datapoints. For example, the label drawn in yello circle depends on the four neighboring labels.
drawn in black circles with red boundaries. Among these neighboring datapoints, we define the cliques with only the left, up and right connectivity. The resulting label dependence is shown on the left side of the diagram. In this MRF structure, only the pairwise dependence among the labels are modeled and the neighborhood is defined by spatial proximity. MRFs of this type is popular in vision for many applications (see Section 2.2.5). Modeling higher order interaction are becoming increasingly popular as well. The proposed method deal with higher order interactions using a Markov Network with a general hypothetical neighborhood definition (unlike the grid structure in Figure 2.3).

With suitably defined neighborhood and cliques, MRF requires a clique potential function to be defined on each clique and datapoint (also regraded as a clique of size 1). These potential functions quantify the costs each of the datapoints and cliques incur for a label assignment. The optimal configuration of the labels is the one that minimizes the joint probability function defined on the summation of all these clique functions [82]. Let $\mathcal{V}^k$ denote the set of all possible cliques of size $k$ and $\mathcal{E}^k$ denote the potential function defined on these cliques. The joint probability $p(X)$ of any label configuration $X$ is given by

\begin{equation}
    p(X) = \frac{1}{Z} \exp\{-E(X)\},
\end{equation}

where

\begin{equation}
    E(X) = \sum_{k=1}^{K} \sum_{\mathcal{V}^k \in \mathcal{V}^k} \mathcal{E}^k(X^k)
\end{equation}
where $Z$ is a normalizing factor. The optimal label configuration maximizes the joint probability $p(X)$, or equivalently, minimizes the Gibbs energy function $E(X)$, as defined in Equation 2.5. Computing the optimal label configurations is commonly called inference procedure for MRF. Many researchers have worked on developing efficient and accurate inference algorithms for MRF, we briefly discuss some of them below.

### 2.2.2 Conditional Random Field (CRF)

An important subclass of Markov Network is Conditional Random Field (CRF). CRF is a conditional distribution with an associated graph structure [46]. More precisely, MRF models compute the posterior probability $p(X|V)$ of a label configuration by combining the conditional likelihood $p(V|X)$ given labels and prior $p(X)$ over labels as in the Bayes rule. On the other hand, CRF models the conditional distribution $p(X|V)$ directly and therefore is discriminative. The Markovian constraint on the conditional probability of the labels becomes $p(x_i \mid V, X \setminus \{x_i\}) = p(x_i \mid V, x_j \text{ where } v_j \in \mathcal{N}_i)$, notice that the probabilities are globally conditioned on observation. Consequently, The quantities $p(X), Z$ and $E^k(X^k)$ in Equations 2.4 and 2.5 are conditioned on the data cliques and are replaced by $p(X|V), Z(V)$ and $E^k(X^k, V^k)$ respectively.

$$
p(X|V) = \frac{1}{Z(V)} \exp\{-E(X; V)\}, \quad (2.6)
$$

where

$$
E(X; V) = \sum_{k=1}^{K} \sum_{V^k \in V^k} E^k(X^k; V^k) \quad (2.7)
$$

Generally, in CRFs, the clique potential functions include a weight or function defined over the datapoints (not labels) to incorporate dependency on the data. The discriminative nature of CRFs allow learning the necessary parameters from labeled data. Almost all vision problems with a Markov model that we will discuss (and/or differentiate our method from) in this dissertation effectively use Conditional Random Field and use general Markov network inference algorithms as well to compute the optimal labeling. These applications of Markov networks in vision attempt to solve a problem (e.g., segmentation, denoising) given an image, i.e., they attempt to optimize probability of a configuration given an image. The underlying framework and inference methods these method utilize are same as MRFs and the difference between these
two methods (CRF and MRF) does not play any significant role in these vision models. It has become common to address Markov models for vision as MRF [120] models. We will also call these models as MRFs and CRFs interchangeably in this dissertation.

A common form of CRF is the log-linear model where the clique potential functions can be expressed as linear combinations of factors (or features). Factors are some functions defined on each clique. The clique potential function is defined as follows.

\[
E_k(x_k, v_k) = \sum_l w_l \phi_l(x_k, v_k)
\]

In this definition, \( \phi_l(x_k, v_k) \) are the factors defined on each clique and \( w_l \) are the mixing weights.

### 2.2.3 Inference for Pairwise Markov Models

Efficient algorithms for solving MRF inference on graphs include Graph-Cut method [11], primal-dual method for Linear Programming (LP) relaxation [76], efficient belief propagation [29, 98]. Wainwright et.al. [126] points out an important connection between LP relaxation methods and belief propagation. This connection was utilized to design several efficient message passing algorithms that were also shown to work very well [69, 75] (see the survey paper [120] for a comparison among these methods).

### 2.2.4 Inference for Higher Order Markov Models

For higher order MRF inference, [65, 66] propose efficient techniques to reduce a sub-class of submodular higher order interactions to pairwise ones so that Graph-cut methods can be applied. Komodakis [74] et.al. extends the dual decomposition based message passing algorithm for higher order relations. A third type of algorithms resort to pseudo-boolean optimization methods for solving higher order Markov models [60].

### 2.2.5 Markov Models in Vision

Since its introduction in [45], Markov Random Field (MRF) has been used in numerous applications in vision. We will discuss more recent models and applications, especially those handle
higher order information among the data. Most Markov models in vision consider each pixel as a datapoint. These models offer a principled approach to enforce dependency among the datapoints in a neighborhood, where the neighborhood is commonly defined based on the spatial proximity of the data [79], [11]. Due to large number of datapoints, a large part of research in MRF community deals with the design of efficient algorithms for inference.

We show an example Markov model for binary image segmentation in Figure 2.4. As before, the cyan and black circles represent the datapoints and their labels respectively and the random field is defined over a grid structure on the labels. Here the datapoints are the pixels of an input image containing the digit 5 in heavy noise. We wish to label each of the datapoints using a binary label, 1 if it falls on the digit, 0 otherwise. The label image is the output image of the segmentation with the digit clearly separated from background. For each individual datapoint a cost function is designed to penalize a label assignment. For example, if the pixel is relatively darker than others, it is more likely to belong to background than to be a digit pixel. Therefore, it should incur a high penalty for label 1 (digit class) and low penalty for label 0 (background class). In addition, each pairwise clique is also associated with a potential function that penalizes a disagreement between the two label assignments of the member of this pair. Once the clique functions are defined, an inference algorithm is supposed to produce the optimal label configuration as close as possible to the ideal output.

![Figure 2.4: An illustration of using MRF for binary image segmentation.](image)

Common applications of Markov models defined on graph are image restoration, stereo
matching, binary image segmentation, photomontage, optical flow estimation etc, see [120, 29]. Other notable applications of pairwise Markov models for images are interactive image segmentation [107], image denoising and inpainting using field of experts model [106], object recognition [78]. Several other works incorporate higher order information for same and different objectives, e.g., image segmentation [67], fields of expert [79], stereo reconstruction [130].

Once again, these applications are actually modeling the the problems with CRFs. But, as stated in Section 2.2.2, it is common in vision community to refer them as MRFs since these methods are not sensitive to the difference between MRFs and CRFs. Most of the Markov models have two common properties, 1. they consider the pixels of an image as datapoints and define neighborhood based on spatial proximity ([78] being an example of notable exception), and 2. they incorporates information from higher order interactions as smoothness terms mainly to enforce spatial consistency in the labels. We will show how Markov networks can be applied on a larger set of problems by considering a general definition of the neighborhood and a different clique function not designed to act as a smoothness factor.

### 2.3 Factor Graphs

Factor graphs [77, 39] was proposed to capture the (local) dependence among arbitrary sized data subsets. This dependence is quantified by a function, defined on the local neighborhood, called factor. A factor graph is a bipartite graph with data samples and the associated factors as the nodes and edges establishing the relationship between samples and factors.

Let us suppose, $f_q$ is the factor defined on the $q$-th of many subsets (possibly of different size) of labels of datapoints. A factor graph assumes some function $f(X)$, defined over all the labels, factors into product of such factors: $f(X) = \prod_q f_q(X_q)$. The bipartitie graph that emobodies the dependence imposed examples of such subsets are shown in Figure 2.5.

It is not difficult to realize that factors and clique potential functions are essentially equivalent. In fact any MRF structure can be modeled as a factor graph. Factor graphs are more general than MRF since it can handle different size subsets with different factors for each of

---

1 Work of [78] uses a second order cone programming relaxation to solve the inference.
A factor graph defined by factors: \( f_1(x_1, x_2) \), \( f_2(x_1, x_2, x_3) \), \( f_3(x_3, x_4) \), \( f_4(x_3, x_5, x_6, x_7) \).

Figure 2.5: Example of a Factor Graph. Grey squares and cyan circles represent factors and datapoints respectively; the arrows indicate dependences.

these subsets. Factor graphs present a better understanding and organization of belief propagation algorithms. In a recent study, Lan et.al. [79] used a factor graph representation of a higher order MRF to apply belief propagation. We will revisit belief propagation with Factor graph in Chapter 4.

Our framework uses a factor graph model to address data labeling. We propose a suitable definition of data subsets and a specific type of factor to solve this problem. Our method shows how factor graphs can be employed in various applications that have not been shown before.

2.4 Model Estimation Problems

Traditional model estimation problems in vision deals with estimating a model from noisy dataset. Given a dataset, the goal of model estimation is to identify the samples that satisfy a certain analytical model. A standard method for this type of model estimation repeatedly samples small sets of examples large enough to estimate a candidate model. This estimated model is then evaluated against all the examples to compute a goodness of fit measure. Methods of RANSAC [33] family attempt to greedily find the best model to fit the data, each time replacing an old estimate by a better one. After a certain determined statistically pre-computed number of iterations RANSAC outputs the best model estimated.

We elaborate this procedure by an example of line fitting on a 2D dataset as shown in Figure 2.6. To identify which of the points, shown in blue squares, on this 2D plot fall on a line, we sample three points (minimum number of samples required is two) randomly from the
data (shown in black circles). A line is estimated from this subset of data and the ratio of points lying close to this line is computed. The points lying close to the line are often termed as inliers. Naturally, the estimated line, shown in red, on the left plot (Figure 2.6(a)) would have a larger inlier ration than that computed using the line of the right plot (Figure 2.6(b)). Algorithms of RANSAC family iterate this estimation stage and always select the model with higher inlier ratio (e.g., the line on the left image in Figure 2.6).

An important aspect of these problems is, we only have higher order information to label individual datapoints. No information about individual sample is available. This is because the analytical model always comprises more than one parameters and therefore requires more than one samples to estimate it. The information available in these problems can be modeled by a hypergraph. Each sampled subset can be considered as a hyperedge and the associated inlier ratio as the hyperedge weight. The strategy of RANSAC-type algorithms can then be reduced to a search to find the hyperedge with largest inlier ratio. On the contrary, the proposed method is intended to work on the full hypergraph to find the optimal labeling of the nodes. Prior works on model estimation algorithms and their differences with the proposed method will be studied in details in Chapters 6 and 5 respectively.
Chapter 3

Overall Framework

We model the label interactions among the members of the subsets by an Undirected Graphical Model or Markov Network. This chapter discusses the framework and all its necessary components. The problem of data labeling with subset wise information is first elaborated in Section 3.1. The neighborhood structure and a suitable (class of) function for each clique are defined in next two sections. This chapter explains how modeling the framework as Markov network enables us to pose the labeling problem as an inference problem and to use the existing works on Markov inference and learning for our method.

3.1 Problem Description

Suppose there are \( n \) datapoints \( V = \{v_i| 1 \leq i \leq n\} \) that we wish to separate into two subsets or categories, \( A \) and \( B \) comprising. We wish to label these samples using binary variables \( \{x_i \in \{0, 1\}| 1 \leq i \leq n\} \) where \( x_i = 1 \) implies \( v_i \in A \) and \( x_i = 0 \) implies \( v_i \in B \).

Throughout the dissertation, we use the term ‘group’ to imply (interchangeably) a subset \( V^k \) of \( \{v_{i_1}, \ldots, v_{i_k}\} \) of size \( k \) where \( 1 \leq i_l \leq n \) and \( 1 \leq l \leq k \). The likelihoods (also called weights) that all members of \( \{v_{i_1}, \ldots, v_{i_k}\} \) belong to \( A \) and \( B \) are denoted by \( \lambda_1(v_{i_1}, \ldots, v_{i_k}) \) and \( \lambda_0(v_{i_1}, \ldots, v_{i_k}) \) respectively. For computation of these likelihood values, we assume the existence of two models, one for each class \( A \) and \( B \). This models can be either analytical (closed form) or implicit (realized by a procedure). In case there is only one model available for category \( A \) only, we assume \( \lambda_1(V^k) \) to be normalized between \([0, 1]\) and define \( \lambda_0(V^k) = 1 - \lambda_1(V^k) \). We describe the problem visually in Figure 3.1 for four datapoints.

It is worth mentioning here that, the set of subsets \( V^k \) is considered as input to the problem. It is not required to generate all possible subsets of a certain size, we assume that sufficient number of subsets are available to capture the dependence among datapoints. The number...
Two subsets of input data \( \{v_1, v_2, v_3\}, \{v_2, v_3, v_4\} \) to be labeled into class \( A \) and \( B \). Two models (or procedures) \( M_A \) and \( M_B \) for class \( A \) and \( B \) are assumed to exist.

\[
\begin{align*}
\lambda_1(v_1, v_2, v_3) & \quad \lambda_0(v_1, v_2) \\
\lambda_4(v_2, v_3, v_4) & \quad \lambda_0(v_2, v_3, v_4)
\end{align*}
\]

Compute likelihood values given model: e.g., compute \( \lambda_1(\{v_1, v_2, v_3\}) \) given model \( M_A \) and \( \lambda_0(\{v_1, v_2, v_3\}) \) given model \( M_B \).

Intended output. Given likelihood values for many such subsets, group the data into two classes.

Figure 3.1: Visual description of data subsets, models and input data generation.

and procedures of these subsets are application dependent and they may be determined either empirically or using the standard formula used in the problem domain.

### 3.2 Graphical Model Framework

As already stated, the objective of this work is to propose an algorithm to label the datapoints \( v_i, 1 \leq i \leq n \), mainly based on the information provided in terms of small groups of them. We may or may not have any information about individual sample \( v_i \) or pair of samples, but the focus is on how to utilize the knowledge we have for the subsets. To this end, the input to the algorithm is a set of small groups or subsets of data samples along with their likelihood values.

Let \( \mathcal{V}^k \) be the set of all groups \( \{v_1, \ldots, v_k\} \) that satisfy a certain condition\(^1\).

To establish a neighborhood \( \mathcal{N}_i^k \) for any datapoint \( v_i \), we define \( v_j \) as a neighbor of \( v_i \) if both \( v_i \) and \( v_j \) are members of any group \( V^k \in \mathcal{V}^k \). Any subset \( V^k \in \mathcal{V}^k \) also defines a clique

\(^1\)For example, a condition check retains only groups with weights larger (or errors less) than a threshold \( \delta^k \). This threshold \( \delta^k \) is application dependent and we mostly set its value empirically.
of size $k$ in this neighborhood system.

$$
\mathcal{N}_i^k = \{ v_j \mid j \neq i \ \text{and} \ \exists V^k \{ v_i, v_j \} \subseteq V^k \}. \quad (3.1)
$$

For inputs with different size groups, $k_1, k_2, \ldots$, we can similarly define a neighborhood system for each subset size.

We wish draw the attention to the fact that this definition of neighborhood is rather hypothetical or abstract unlike the ones that are traditionally defined on pixel grids of images. As shown in Figure 3.2, the two dotted polygons indicate two subsets of examples represented by cyan circles. All the members of of the same subset, i.e., samples in the same polygon, are neighbors to each other. In our framework, these polygons also define a clique. The relationship among the labels (drawn as black circles) of these datapoints are shown in solid polygons.

![Figure 3.2: Hypothetical neighborhood assumed among members of data subsets.](image)

Now, we can define a Markov Network over the label variables $x_i$ assuming the Markov property that the value of $x_i$ depends on $x_j$ only if $v_j$ is a neighbor of $v_i$. Any subset $V^k \in \mathcal{V}^k$ defines a clique of size $k$ in the neighborhood system. Let $X^k$ denotes the labels of members of $V^k$, i.e., $X^k = \{x_{i_1} \ldots x_{i_k}\}$ and $X$ denotes labels of all $n$ datapoints $\{x_1, \ldots, x_n\}$.

It is well known that the probability of any assignment $p(X|V)$, given the datapoints, depends on what is known as the Gibbs energy function $E(X; V)$ [82]. The energy function $E(X; V)$ is the summation of potential functions defined on cliques. Let $E^k(X^k; V^k)$ be an appropriately defined clique potential function for the clique $V^k = \{v_{i_1}, \ldots, v_{i_k}\}$ of size $k$. Then the Gibbs energy functions is defined as follows.
\[ E(X; V) = \sum_{k=1}^{K} \sum_{V^k \in \mathcal{V}^k} E^k(X^k; V^k) \]

With different size cliques, the Gibbs energy function equals to sum over all energy function of different sizes [82]. The optimal assignment \( X = \{x_1, \ldots, x_n\} \) should minimize this Gibbs energy function. To complete the definition of the overall energy function, we define the clique potential functions \( E^k(X^k; V^k) \) for each \( k \) in Section 3.3.

Before describing the clique function in details, we would like to point out that the graphical model we use here is a general definition of Markov Network models. Typically, MRF or CRF studies in computer vision assumes a spatial neighborhood among the neighboring pixels or their labels [79, 120]. The relation among the labels are modeled using a tree or graph structure, where each label corresponds to a node in the graph. One popular example of this type of neighborhood is the grid structure assumed among the pixels. We use a hypothetical neighborhood definition induced by the subsets and is different from those assumed in traditional applications of Markov models.

### 3.3 Proposed Clique Function

The clique potential function determines the cost of assigning the labels within the clique into different classes. A ‘generative’ higher order order clique function, a type frequently used in most Markov Random Field applications, is not useful for problems we address. For example, a higher order Potts model [65], assigns a larger cost when at least one of the labels in the clique is different from the others, and a smaller cost when all labels are same. Without information about assignment costs for each individual datapoint, a trivial assignment will always produce the lowest overall cost for higher order Potts cost. Hypergraph clustering methods work without pointwise information, but they usually combine the cut function with a restraint function to avoid trivial solution (Chapter 5 discusses these issues in more details).

We wish to generate a non-trivial solution without incorporating any additional functionals. To achieve this, we introduce two penalty functions \( g_c(\cdot), \ c \in \{1, 0\} \), for the two categories \( A \) and \( B \). Penalty function corresponding for each class penalizes number of label variables assuming the other class label. That is, when combined together, these penalty functions act as
competing functions, each trying to decrease the number of opposite labels within the clique (and increase the number of labels of its own class). The likelihood values act as weights for these penalty functions so that for a large likelihood value $\lambda_1(V^k)$ for class $A$, there will be higher cost to label the clique variables to 0 (class $B$).

To be more specific, let $\eta_c$ denote the number of variables $x_{i_1} \in X^k$ in the clique $V^k$ to be assigned to class $c$. A penalty function $g_c(\cdot)$ is defined on the number $\eta_{1-c}$ of variables for clique $V^k$ to be assigned to the opposite class $1-c$. Ideally, the penalty function $g_c(\cdot)$ should be non-decreasing with the increase in $\eta_{1-c}$. We define a clique potential function $E^k(X^k; V^k)$ for clique $V^k$ as a weighted linear combination of competing penalty functions $g_1(\eta_0)$ and $g_0(\eta_1)$, weighted by the likelihood value $\lambda_1(V^k)$ and $\lambda_0(V^k)$ respectively.

$$E^k(X^k; V^k) = \beta_1 \lambda(V^k) g_1(\eta_0) + \beta_0 \lambda_0(V^k) g_0(\eta_1). \quad (3.3)$$

In this definition, $\beta_1$ and $\beta_0$ are two nonnegative balancing parameters. Recall, $\lambda_1(V^k)$ and $\lambda_0(V^k)$ quantify the likelihood that all $v_{i_1} \in V^k$ belong to $A$ and $B$ respectively, and that $\eta_1 + \eta_0 = k$. With high likelihood $\lambda_1(V^k)$ value, the clique potential would be prone to decrease $\eta_0$, the number of variables $x_{i_1} \in X^k$ to assume label 0 and tolerate a small penalty $\lambda_0(V^k) g_0(\eta_1)$ as $\lambda_0(V^k)$ is small. The behavior will reverse for low $\lambda_1(V^k)$ when it will try to decrease $\eta_1$. Thus, this potential function becomes a weighted combination of two competing penalty functions, each trying to increase number of $x_{i_1} \in X^k$ to be labeled to its corresponding class.

In cases where there is only one model available for category $A$, $\lambda_0(V^k)$ is replaced by $1 - \lambda_1(V^k)$ in Equation 3.3 (recall our assumption that $\lambda_1(V^k)$ is normalized). Example of such likelihood values could be found model estimation problems where we have only one model description.

Our potential function in Equation 3.3 is a clique potential function for a Conditional Random Field (CRF) model. It is indeed a weighted summation of two factors (in our definition, the penalty functions). We will restate the clique potential function as a linear combination of factors in the next section for training. In fact, the penalty functions themselves will be decomposed into smaller components and these components will act as the factors in this definition.
by whether or not the clique function is defined for a CRF or an MRF as we will see in the following sections (refer to Section 2.2.2).

With the definition of clique potential function, we complete the definition of the overall Gibbs energy function in Equation 3.2. Any standard algorithm for Markov Network inference can be applied to determine the optimal labels $x_i$, as discussed in Chapter 4. In fact, for a linear form of penalty functions, it is possible to compute the optimal label using a very simple algorithm (Chapter 4). However, it remains to investigate (1) which types of penalty functions would be useful for the labeling problem, (2) how different forms of $g_c$ functions will affect the inference, and (3) which forms of $g_c$ are theoretically guaranteed to produce an exact solution. All these issues are discussed in Section 3.4.

3.4 Analysis of the Clique Function

At this point, we are interested to learn the behavior of the clique potential function. In Section 3.4.1, it is investigated which forms of the proposed clique function makes the energy function submodular and therefore is theoretically guaranteed to generate an optimal solution. The following section, Section 3.5, visually inspects different forms of the clique function and describes how they affect inference. Analysis of clique function would assist to design suitable potential functions for different instances of hypergraph labeling problem. For the situations where a learned potential is more advantageous than pre-defined ones, Section 4.2 describes methods to learn parametric form of the potentials specific to problem instances.

3.4.1 Submodularity and Tractability

Submodularity is an important property to investigate for solving inference in undirected graphical models. In past several years, many researchers investigated what properties of higher order potential functions are guaranteed to produce a globally optimal solution by various inference methods, we refer interested readers to [60, 128] and references therein to learn more on this topic. In summary, it has been shown that for a submodular higher order clique potential function, there are algorithms to produce a globally optimal solution efficiently [20, 74] for two class problems. This section analyzes the proposed clique function of Equation 3.3 to identify
what properties make it submodular and therefore can be solved efficiently and accurately with existing algorithms.

**Definition of Submodular Function**

Let $Y$ be a subset of some set $W$ and $\mathcal{P}(W)$ denotes set of all subsets of $W$. A function $f(\cdot)$, defined over $\mathcal{P}(W)$, is called submodular if for any $u, z \in W \setminus Y$ the increase $f(Y \cup \{u\}) - f(Y)$ is larger than $f(Y \cup \{u, z\}) - f(Y \cup \{u\})$ [41].

$$f(Y \cup \{u\}) - f(Y) \geq f(Y \cup \{u, z\}) - f(Y \cup \{u\}).$$

(3.4)

This implies the rate of change of a submodular function $f(Y)$ decreases as we keep augmenting the set $Y$ with new samples from $W \setminus Y$. An example submodular function is the Cut function of a graph. Submodular functions are widely regarded to be analogous to convex functions in set domain [85]. A general submodular function is theoretically guaranteed to be minimized in strongly polynomial time, the lowest known bound is $O(n^6)$. However, in MRF community there exists algorithm (e.g., Graph-cut method) that can minimize a particular submodular function in $O(n^3)$ time.

**3.4.2 Energy function as submodular function:**

Now, our goal is to show that the energy function defined in Equation 3.2 is a submodular function. Observe that $E(X; V)$ in Equation 3.2 is expressed as a summation of clique potentials $E^k$ defined over labels of $\mathcal{P}(V^k)$. Form the study of Lovasz [85], we can define a function $\tilde{E}^k(X; V)$ which enables $E^k(X^k; V^k)$ to handle the inputs from domain $\mathcal{P}(V)$, where $V = \{v_1, \ldots, v_n\}$. If we can show that $E^k(X^k; V^k)$ is submodular and replace them by $\tilde{E}^k(X; V)$ in Equation 3.2, the Gibbs energy function becomes a summation of submodular functions. The nonnegative summation of submodular functions is also submodular [41]. Hence, if we could the submodularity of clique potential $E^k(X^k; V^k)$, the energy function $E(X; V)$ becomes submodular.
3.4.3 Submodularity of $E^k(X^k; V^k)$

As we know, the clique potential function $E^k(X^k; V^k)$ is a weighted summation of penalty functions $g_c(\cdot)$, $c \in \{0, 1\}$.

$$E^k(X^k; V^k) = \beta_1 \lambda(V^k) g_1(\eta_0) + \beta_0 \lambda_0(V^k) g_0(\eta_1).$$  \hspace{1cm} (3.5)

We first express the proposed clique function of Equation 3.5 as a set function to prove its submodularity. Let $A^k \in V^k$ be a subset of data samples in $V^k$ that belong to class $A$. Then $X^k$ becomes an indicator vector for set $A^k \in V^k$. Now, the potential function $E^k(X^k; V^k)$ can be restated as $f(A^k) = C_1 g_1(\eta_0) + C_0 g_0(\eta_1)$ where $C_1 = \beta_1 \lambda_1(V^k)$, $C_0 = \beta_0 \lambda_0(V^k)$ and $\eta_0 + \eta_1 = k$. If we add any $v_i \in V^k \setminus A^k$ to $A^k$, the clique function becomes $f(A^k \cup \{v_i\}) = C_1 g_1(\eta_0 - 1) + C_0 g_0(\eta_1 + 1)$. We next show what properties of the proposed clique potential makes it submodular.

**Proposition:** The proposed clique function is submodular if

$$C_1 \Delta_1(\eta_0) + C_0 \Delta_0(\eta_1 + 2) \leq 0.$$ \hspace{1cm} (3.6)

where $\Delta_c(n) = [g_c(n) - g_c(n-1)] - [g_c(n-1) - g_c(n-2)]$ is the second order difference for $c \in \{0, 1\}$.

**Proof:** If we augment $A^k$ further by $v_{ij} \in V^k \setminus (A^k \cup \{v_i\})$, the clique function becomes $f(A^k \cup \{v_i, v_{ij}\}) = C_1 g_1(\eta_0 - 2) + C_0 g_0(\eta_1 + 2)$. To prove the submodularity of $f(A^k)$, we have to prove that the successive increase in $f(A^k)$ diminishes.

$$f(A^k \cup \{v_i\}) - f(A^k) \geq f(A^k \cup \{v_i, v_{ij}\}) - f(A^k \cup \{v_i\})$$

$$\Rightarrow -C_0 g_0(\eta_1 + 2) + 2C_0 g_0(\eta_1 + 1) - C_0 g_0(\eta_1) \geq$$

$$C_1 g_1(\eta_0 - 2) - 2C_1 g_1(\eta_0 - 1) + C_1 g_1(\eta_0 - 2)$$

$$\Rightarrow C_1 \left\{ [g_0(\eta_1 + 2) - g_0(\eta_1 + 1)] - [g_0(\eta_1 + 1) - g_0(\eta_1)] \right\}$$

$$+ C_1 \left\{ [g_1(\eta_0) - g_1(\eta_0 - 1)] - [g_1(\eta_0 - 1) - g_1(\eta_0 - 2)] \right\} \leq 0$$

The proof follows by replacing the second order difference as $\Delta_c(n) = [g_c(n) - g_c(n-1)] - [g_c(n-1) - g_c(n-2)]$ where $c \in \{0, 1\}$. \Box
There are several options for $g_c$ functions to render the clique potential to be submodular.

**Linear $g_c$:** It is obvious that for linear $g_c$, the second order differences is 0 and the clique function is submodular. In fact, for linear $g_c$, the clique potential becomes modular.

**Concave $g_c$:** As the second derivative of any concave function is negative, the condition in Equation 3.6 holds.

**Constant $\Delta_c$:** For $g_c$ functions with constant $\Delta_c(n) = \bar{\Delta}_c$, one can design a submodular clique function with a concave $g_1$ and convex (or linear) $g_0$ whenever $C_1 > C_0$ and $|\bar{\Delta}_1| > |\bar{\Delta}_0|$ where $|\cdot|$ implies absolute value (and vice versa). We can also use a combination only when $C_1 > C_0$ and use concave functions for both $g_1$ and $g_0$ otherwise.

The next two sections discuss the behavior of the clique functions with respect to different forms of penalty functions to assist designing a suitable clique function for a specific task. The submodularity of the proposed clique function indicates which forms of the clique function is theoretically guaranteed to be optimized by Markov network inference. However, it is not imperative for the clique functions to be submodular for practical purposes. It has been empirically shown that there exists algorithms that can optimize Gibbs energy function that are not submodular. The discussion in this section describes the tractability of the proposed clique function from a theoretical point of view.

### 3.5 Effect of Penalty Functions on Inference

The penalty functions are ideally non-decreasing functions, that is, it increasingly penalizes more and more datapoints to be assigned to the other class. In this section, we plot the penalty functions against their arguments to illustrate their behavior with respect to changes in number of labels in the clique and the subset likelihood values. This discussion will assist designing a suitable problem specific clique function.

#### 3.5.1 Linear $g_c$

The linear form of $g_c$, $c \in \{0, 1\}$ increases from $l_c$ to $h_c$ in proportion to $k - \eta_c$ ($\eta_c + \eta_{1-c} = k$).

$$g_c(k - \eta_c) = l_c + \frac{h_c - l_c}{k}(k - \eta_c).$$

(3.7)
In Figure 3.3, the two linear penalty functions $g_1$, $g_0$ and the resulting clique potential function $E^k(X^k;V^k)$ for one subset (or clique in this dissertation) are plotted on the y-axis against $\eta_0$ values on the x-axis. Recall that, $\eta_0$ is the number of $x_i$ in the clique to be assigned a label 0. A simple set of parameters are chosen for the linear form: $l_0 = l_1 = 0$, $h_1 = h_0 = 1$. We plot the penalty functions $g_1$, $g_0$ and the clique potential $E^k(X^k;V^k)$ in blue, red and black respectively. The left and right plots in Figure 3.3 shows these function values for high $\lambda_1(V^k) = 0.8$ and low $\lambda_1(V^k) = 0.2$ respectively.

As shown in both the plots, an increase in the number of $x_i$ taking 0 as label increases the penalty $g_1$ (blue line) and decreases $g_0$ (red line). Also, these penalties are weighted by the likelihood values in the definition of proposed clique potential (Equation 3.3). If the likelihood $\lambda_1(V^k)$ for category $A$ is larger than that of class $B$, the penalty for class $A$ will have a higher weight and the resulting $E^k(X^k;V^k)$ will have an upward trend w.r.t. $\eta_0$ values, as shown in Figure 3.3 (a). That is, larger $\eta_0$ will be penalized more than lower $\eta_0$ values. Since the ultimate goal is to reduce the cost, this potential will try to reduce the number $\eta_0$ of variables to be assigned label 0 for this clique and increase the value of $\eta_1$. The situation reverses for low values of $\lambda_1(V^k) = 0.2$, see Figure 3.3(b). The inference algorithm attempts to find an optimal label assignment such that a summation of many clique functions, defined over all the cliques, attains the lowest possible cost.

![Penalty and energy functions: proposed](image1)

![Penalty and energy functions: proposed](image2)

Figure 3.3: Proposed clique potential and penalty functions vs $\eta_0$ values. Left: $\lambda_1(V^k) = 0.8$, right: $\lambda_1(V^k) = 0.2$. Blue and Red lines in the left image are penalty functions $g_c$

Furthermore, an increase of $l_c$ or a decrease of $h_c$ will lower the number of datapoints to be labeled as category $c$ since the former would introduce a penalty to assign label $c$ to any sample
in a subset and the latter would simply reduce the penalty to assign $1 - c$.

### 3.5.2 Nonlinear $g_c$

Nonlinear forms of functions, e.g., concave or convex, can also be used for penalties $g_c$. In general, a concave $g_c$ would be more conservative on the number of $x_{i_l}$ to be labeled as $1 - c$, that is, it will be more difficult to have label disagreement within the clique. A convex penalty function would impose less amount of cost to have few $x_{i_l}$ to take the opposite label. To understand why this is true, potential function $E^k(X^k; V^k)$ comprising convex and concave penalties with $\lambda_1(V^k) = 0.8$ are plotted against $\eta_0$ values in Figure 3.4. Though $E^k(X^k; V^k)$ with both concave and convex penalties (plots 3.4(a) and 3.4(b) resp.) increases due to large $\lambda_1(V^k)$, the rates of change are very different from each other. For low $\eta_0$ values, the rate at which $E^k(X^k; V^k)$ increases is much lower for convex $g_1$ and $g_0$ (plot 3.4(a)) than that for concave penalties (plot 3.4(b)). Therefore, the potential function will tolerate small number of $x_{i_l}$ to be assigned to 0 for convex $g_c$. On the other hand, $E^k(X^k; V^k)$ will be more strict on larger $\eta_0$. Similarly, Concave $E^k(X^k; V^k)$ becomes less sensitive to change for larger values of $\eta_0$. As a result, it tends to assign all $x_{i_l}$ to 0 rather than assigning $k - 1$ of the label variables to 0 and remaining one to 1.

In summary, this chapter explains how the clique potential function try to enforce the resulting labeling to conform with the likelihood values. It is also discussed for which forms of penalty functions the proposed method will produce the exact solution of our two class labeling
case. It is important to know these theoretical and analytical properties of the potential functions to design one for the problem at hand, even if we do not exactly follow these suggested forms. The next chapter will describe existing inference techniques to solve the inference problem for the Markov Network.
4.1 Inference

To determine the optimal labeling, any standard inference methods for undirected graphical models or Markov Networks can be applied. However, for linear $g_c$ functions, a simple and efficient procedure to perform inference is stated in Section 4.1.1. Section 4.1.2 describes other standard algorithms for inference with nonlinear penalties. These inference algorithms are well known in the Markov Random Field (MRF) literature; this section describes how they can be applied within our context.

4.1.1 Inference for Linear $g_c$

The linear form of $g_c(\cdot)$, $c \in \{0, 1\}$, increases from $l_c$ to $h_c$ in proportion to $k - \eta_c$ ($\eta_c + \eta_{1-c} = k$).

$$g_c(k - \eta_c) = l_c + \frac{h_c - l_c}{k} (k - \eta_c).$$  \hspace{1cm} (4.1)

For the linear penalties defined in Equation 4.1, the clique potential function is the weighted sum of two linear structures.

$$E^k(X^k; V^k) = \sum_{c \in \{0, 1\}} \beta_c \lambda^k_c(V^k) \left( l_c + \frac{h_c - l_c}{k} (k - \eta_c) \right).$$  \hspace{1cm} (4.2)

Recall that, $X^k = \{x_{i_1} \ldots x_{i_k}\}$ and $X = \{x_1, \ldots, x_n\}$. Therefore, the total Gibbs energy function (for single $k$) becomes

$$E(X; V) = \sum_{V^k \in V} \sum_{c \in \{0, 1\}} \beta_c \lambda^k_c(V^k) \left( l_c + \frac{h_c - l_c}{k} (k - \eta_c) \right).$$  \hspace{1cm} (4.3)
Ignoring the constant terms, minimizing the cost in Equation 4.3 is equivalent to minimizing
\[ \sum_{V^k \in V^k} \sum_{c \in \{0,1\}} \beta_c \lambda^k_c(V^k) \left( \frac{h_c - l_c}{k}(k - \eta_c) \right). \]
Due to linearity, we can break down the likelihood value to each datapoint by a ratio \( \frac{h_c - l_c}{k} \) and rewrite the cost function as follows.

\[ E(X; V) \propto \sum_{i=1}^{n} \left[ x_i \sum_{V^k \in V^k} \sum_{v_i \in V^k} \beta_0 \lambda^k_0(V^k) \frac{h_0 - l_0}{k} + (1 - x_i) \sum_{V^k \in V^k} \sum_{v_i \in V^k} \beta_1 \lambda^k_1(V^k) \frac{h_1 - l_1}{k} \right]. \tag{4.4} \]

It is straightforward to see that the value of \( x_i \) is simply the label for which the summation of likelihood ratios weighted by the slope of \( g_c(\cdot) \) is maximum.

\[ x^*_i = \arg \max_{c \in \{0,1\}} \sum_{V^k \in V^k \land v_i \in V^k} \beta_c \lambda_c(V^k) \frac{h_c - l_c}{k}. \tag{4.5} \]

This solution can be computed in \( O(n + |V^k|) \), with one pass over all the subsets and another pass over all the datapoints. For multiple \( k \), the corresponding weights for each tuple size will be added. Though standard methods like max-flow min-cut algorithm are also capable of producing the result, it is unnecessary to invoke them for this inference problem which is known to be a simple task, see [10].

### 4.1.2 Inference with Nonlinear \( g_c \)

For nonlinear \( g_c(\cdot), \ c \in \{0,1\}, \) e.g., concave or convex forms, the likelihood weights can not be evenly distributed to each of the datapoints as suggested in Equation 4.5. Problems with nonlinear \( g_c(\cdot) \) can be solved using a linear programming (LP) formulation suggested in [20, 74]. Other algorithms like sum-product belief propagation [77] or more efficient variants of it can also be used for the proposed method. Both these methods can work in situations where there are no pointwise information (unary potentials) available. Furthermore, they have been shown to produce the expected result with non-submodular energy functions, see [74, 69].

It is not clear yet how a general submodular higher order clique function can be reduced efficiently to pairwise interactions to apply Graph-cut algorithm though it has been shown theoretically to be possible [35, 125, 70]. Furthermore, it has not been studied how to apply
Graph-cut algorithm with only general form of submodular higher order potentials and no unary potentials. Therefore, we did not apply the popular Graph-cut algorithm in this dissertation.

In the following two sections we briefly describe the LP relaxation and belief propagation technique for inference. See [60, 128, 99] and references therein to learn more about various inference algorithms available in the literature.

**Linear Programming Relaxation**

Inference for our Markov Network can be achieved by solving a Linear Programming (LP) problem that relaxes the (discrete) labels to $[0, 1]$. The LP formulation introduces two types of decision variables, $y_i(x_i)$ for each datapoint-label combination and $y_q(X^k_q)$ for each subset-labels combination and minimizes the weighted summation of the potential functions (where $X^k_q$ is the $q$-th subset).

$$\min_{y_i,y_q} \sum_{i,c} y_i(c) E^1(c) + \sum_q \sum_{X^k_q \in \{0,1\}^k} y_q(X^k_q) E(X^k_q; V^k_q)$$

s.t. $\sum_{X^k_q \mid i=c} y_q(X^k_q) = y_i(c), \forall v_i \in V^k_q, \forall V^k_q \in \mathcal{V}^k$

$$\sum_c y_i(c) = 1 \forall i$$

$$y_i(\cdot), y_q(\cdot) \geq 0$$

Ideally, the decision variables are binary valued, $y_i(x_i) = 1$ for the optimal label $x_i^* = c$ of example $v_i$ and $y_i(x_i) = 0$ otherwise. By relaxing $y_i(x_i)$ to real values, the result of the LP minimization effectively provides a confidence value for the corresponding label. The hard labels for each datapoint can be recovered from the real valued LP solution $y_i^*(x_i)$, e.g., by assigning $x_i$ to $c^* = \arg \max_c y_i^*(c)$.

An LP relaxation formulation for MRF inference was first introduced in [112]. A significant drawback with this formulation for large problems is that the number of decision variables, $y_i(x_i)$ and $y_q(X^k_q)$, grows rapidly (there are $C^k$ variables for each clique $q$ where $C =$ total number of labels) with the the number of data samples. In particular, vision problems that treat each pixel as an observation, the number of observation is too large for the standard general
purpose LP solvers to solve efficiently. Several works of Komodakis et.al. [76, 75, 73, 74] proposes efficient algorithms for solving inference with impressive results. The study of [126] and [75] also establishes an important connection between LP relaxation and belief propagation methods and propose (faster) algorithms that solves this optimization problem. See the survey paper [128] for a review of studies in this field.

**Sum-Product Belief Propagation**

Sum-product belief propagation algorithms are best understood in the context of Factor Graphs [77]. A Factor Graph is a bipartite graph where each edge connects a variable node to a factor node (refer to Section 2.3). A variable node $i$ is associated to a data sample label $x_i$ and a factor node $q$ is associated to the clique potential $E^k(X^k_q, V^k_q)$ corresponding to data subset $q$ where $X^k_q$ is the labels of $q$-th subset. The Sum-Product belief propagation algorithm is a message passing algorithm to compute the marginal probability or ‘belief’ of label(s) or state(s) of each datapoint and subset. These marginals are then used to compute the optimal state of each example.

In a factor graph, a factor $q$ is connected to a variable node $i$ only if the corresponding factor $f_q$ has label $x_i$ as its argument. A message from the $q$-th factor node, corresponding to label subset $X^k_q$, to the $i$-th variable node quantifies with how much confidence the factor node $q$ to ‘believe’s that the $i$-th datapoint is in a certain state. A message from a variable node $i$ to a factor node $q$ can be regarded as a quantitative statement of what factors other than $q$ ‘believe’ that datapoint $i$ is in certain state. These messages are passed alternatively from variable node to (connecting) factor node, and vice-versa, until convergence.

Message from variable to factor node:

$$
\mu_{i \rightarrow q}(x_i) \propto \prod_{V^k_{q'} \ni i} \nu_{q' \rightarrow i}(x_i) 
\tag{4.10}
$$

Message from factor to variable node:

$$
\nu_{q \rightarrow i}(x_i) \propto \sum_{X^k_b \setminus x_i} E^k(X^k_q, V^k_q) \prod_{i' \in V^k_b \setminus i'} \mu_{i' \rightarrow q} 
\tag{4.11}
$$
The label $x_i$ for each sample is calculated from the real valued ‘belief’ $b_i(x_i) = \prod_{i \in V^k_q} \nu_{q \rightarrow i}$. As the belief $b_i(x_i)$ indicates approximate probabilities, one heuristic method (used in our experiments) to compute the discrete label for $x_i$ is to assign the label that maximizes $b_i(c)$. The optimal label configuration can also be determined to be the one that maximizes the joint probability. One interesting property of belief propagation is, it also allows us to compute the marginal probability of a specific configuration of labels for any particular clique. The belief of a label configuration $X^k_q$ of a specific clique $q$ can be computed as follows.

$$b_q(X^k_q) \propto E^k(X^k_q; V^k_q) \prod_{i \in V^k_q} \prod_{V^k_{q'} \ni i \quad q' \neq q} \nu_{q' \rightarrow i}(x_i) \quad (4.12)$$

These marginal probabilities of the clique labels will be used in learning the parameters of our framework (Section 4.2.2).

Belief propagation is guaranteed to produce the correct marginals for factor graphs defined on a tree structure [134]. The convergence is not guaranteed for general factor graphs with loops, but in practice it works very well for MRF inference [79, 29]. In fact, it has been proven that when Sum-product belief propagation converges, the resulting marginals form a minima of the Bethe free energy, a quantity from statistical physics which can be thought of as an approximate measure of the distance between a multivariate probability distribution and a set of marginals [134]. This connection between Bethe free energy and loopy belief propagation provided a sound theoretical justification for the application of belief propagation to networks with loops. Furthermore, this discovery has led to new belief propagation methods that minimize Bethe free energy directly, and are guaranteed to converge [136].

In our experiments, the LP relaxation and Belief Propagation algorithms have been used and observed to produce the required solution with a wide range of penalty functions. Much research has been (and is being) conducted on more efficient versions of these methods, both for pairwise and higher order interactions, e.g., [74, 75, 126, 69]. Faster implementation of the labeling method is possible with these efficient versions of inference algorithm. This study is focused on applying the Markov Network for labeling problems rather than developing an inference algorithm for Markov Networks. Therefore, we refrain ourselves from describing these efficient algorithms here.
4.1.3 Parameter Selection

Intuitively, an increase of $\min g_c$ or a decrease of $\max g_c$ will lower the number of datapoints to be labeled as category $c$ since the former would introduce a penalty to assign label $c$ to any sample in a subset and the latter would simply reduce the penalty to assign $1 - c$. Also, a concave $g_c$ would be more conservative on the number of $x_{ij}$ to be labeled as $1 - c$, that is, it will be more difficult to have label disagreement within the clique. A convex penalty function would impose less amount of cost to have few $x_{ij}$ to take the opposite label.

4.2 Learning

It is not difficult to realize that the same form of penalty function will not be effective for all instances of the labeling problem. There exists scenarios where designing a penalty function apriori is either difficult or may result in inferior performance. This section discusses methods to learn these functions from labeled data. The ability to train parameters of the proposed method extends the scope of applications and the possibility of getting better results.

4.2.1 Learning Penalty Functions

Given $J$ sets $V^k_j$, $j = 1, \ldots, J$, of groups of data, along with the likelihood values and the actual labels $X_j$ of the datapoints, our objective is to learn the penalty functions $g_c$, $c = 0, 1$. This section describes how to learn a parametric form of $g_c$ using Markov Network learning algorithms for log-linear models. Log-linear models [46] are popular in Conditional Random Field (CRF) community for several applications, e.g., parsing sequences of text or other samples [113]. Given functions $\phi_l(X^k, V^k)$ (known as factors or features) and mixing weights $w_l$ for each clique, the probability of any configuration in a log-linear model can be written as

$$p(X|V) = \frac{1}{Z(V)} \exp \left[ \sum_{V^k \in V^k} \sum_{l} w_l \phi_l(X^k; V^k) \right]$$  (4.13)

where $V$ is the set of all datapoints and $Z(V)$ is a normalizing term. In what follows, we describe two ways to redefine the penalty functions by a form that allow us to use learning methods for log-linear models and then provide a brief overview of these learning algorithms.
**Discrete** $g_c$

First, we express $g_c$ as a discrete function. Observe that, penalty functions $g_c$ are defined on $\eta(1-c)$ values, which are integers in our case. Therefore, it suffices for this model to learn the (discrete) values of $g_c(\eta(1-c))$ for all $c \in \{0, 1\}$ and $0 \leq \eta(1-c) \leq k$. Let us introduce two quantities as follows.

$$w_c^\alpha = \beta_c g_c(\alpha)$$ (4.14)

$$\phi_c(\alpha, V^k) = -\lambda_c(V^k) I(\eta(1-c), \alpha)$$ (4.15)

where $I(s, t)$ is an indicator function which equals to 1 only when $s$ is equal to $t$ and 0 otherwise. Notice that, in this case, $\phi_c$ functions are the features (for each clique) that assume nonzero values only when $\eta_{1-c} = \alpha$. The clique cost function defined in Equation 3.3 can be rewritten as a summation of impulse functions.

$$E^k(X^k; V^k) = \sum_c \sum_{\alpha=0}^k w_c^\alpha \phi_c(\alpha, V^k).$$ (4.16)

This definition of $g_c$ expresses the joint probability of any assignment as log-linear model. For this form of $g_c$, the values of $w_c^\alpha$ are learned for all $\alpha = 1, \ldots, \eta(1-c)$ and $c = 0, 1$.

**Second order polynomial** $g_c$

We may also learn a more constrained form of $g_c$ by assuming a second order polynomial form for it. In this case, this function can be expressed using the Taylor expansion around reference point 0.

$$g_c(\alpha) = g_c^{(0)} + \alpha g_c^{(1)} + \frac{\alpha^2}{2} g_c^{(2)}.$$ (4.17)

In Equation 4.17, $g_c^{(0)}$, $g_c^{(1)}$ and $g_c^{(2)}$ are the 0, 1st and 2nd order derivatives of $g_c$ at 0. The features for this case can be defined as follows.

$$\psi_c^0(\alpha, V^k) = -\sum_{\gamma=0}^k \lambda_c(V^k) I(\eta(1-c), \gamma)$$ (4.18)

$$\psi_c^1(\alpha, V^k) = -\sum_{\gamma=1}^k \alpha \lambda_c(V^k) I(\eta(1-c), \gamma)$$ (4.19)

$$\psi_c^2(\alpha, V^k) = -\sum_{\gamma=1}^k \frac{\alpha^2}{2} \lambda_c(V^k) I(\eta(1-c), \gamma)$$ (4.20)
Then, the cost function in Equation 3.3 can be expressed as a linear combination of features
\[\psi_e^c(\alpha), \; e = 0, \ldots, 2.\]

\[E(X^k; V^k) = \sum_c \sum_{e=0}^2 g_e^c(\alpha) \psi_e^c(\alpha, V^k) \tag{4.21}\]

For polynomial \(g_e\), we learn the values of \(g_e^c(\alpha)\) for all \(e = 0, \ldots, 2\) and \(c = 0, 1\). This redefinition of \(g_e\) has the benefit of regulating the learned form to be of some specific type. Also, regardless of the size \(k\) or data subset, we only need to learn \(3 \times C\) parameters, where \(C\) is the total number of classes. Next section briefly discusses existing techniques for learning a Markov network with the \(E(X^k; V^k)\) that are being discussed here.

**4.2.2 Learning algorithms**

In last two sections we have shown that the clique potential function of the proposed framework can be expressed as a linear combination of features or factors. Therefore the joint probability of any label configuration will assume the form of that of a log-linear model as stated in Equation 4.13. With discrete form of \(g_e\) the joint probability of a configuration \(X\) is as follows.

\[p(X|V) = \frac{1}{Z(V)} \exp\left\{ \sum_{V^k \in V^k} \sum_{c=0}^k \sum_{\alpha=0} w_c^\alpha \phi_e(\alpha, V^k) \right\} \tag{4.22}\]

where \(Z(V)\) is a normalizing term. The joint probability will be similar for second order polynomial \(g_e\) and we are omitting the derivation for it here. There are two types of algorithms that estimates the parameters \(w_c^\alpha\) from data, one that aims at determining the parameters that maximizes the log-likelihood [46]. The other type of algorithms maximizes the separation, or the label margin, between classes of datapoints [6].

**Estimation by likelihood maximization**

The log likelihood function for the training data is given by

\[l(w) = \sum_{j=1}^J \sum_{V_j^k \in V_j^k} \sum_{c=0}^k \sum_{\alpha=0} w_c^\alpha \phi_e(\alpha, V^k) - \log Z(V). \tag{4.23}\]

It has been shown that \(l(w)\) is concave [46]. Therefore, a simple gradient ascent algorithm is guaranteed to produce the global optimal values for \(w_c^\alpha\). Its is straightforward to see that
the gradient with respect to \( w^c_\alpha \) is becomes the difference between summation of observed and expected \( \phi_c(\alpha) \) values.

\[
\frac{\partial l}{\partial w^c_\alpha} = \sum_j \sum_{V^k \in V^j} \phi_c(\alpha, V^k) - \sum_{V^k \in V^j} \sum_{X^k} \phi_c(\alpha, V^k) p(X^k). \tag{4.24}
\]

We used a sum-product belief propagation algorithm [77] to compute the marginal terms \( p(X^k) \). Also, a regularizer term is added to the likelihood function to penalize large parameter values. It should be noted here that, there exist algorithms, e.g., conjugate gradient or L-BFGS, more efficient than simple gradient ascent for this maximization problem [46] that have been applied successfully for parameter estimation.

**Estimation by Margin maximization**

Other type of algorithms tries to estimate the parameters by maximizing the class margin of the labeled examples. Margin maximization is useful if the data distribution is biased to one of the classes or there are many noisy samples in the data. Bartlett et.al. [6] proposed a constrained optimization problem, in terms of primal variables \( w^c_\alpha \) for parameter learning in maximum margin setting. Their formulation minimizes a loss function, defined in terms of number of incorrectly labeled examples, and a regularizer term. An exponentiated gradient (EG) algorithm is described to minimize the objective that updates the primal variables \( w^c_\alpha \) similarly as in Equation 4.24. In addition, the EG algorithm also updates the the dual variables to minimize the subset-wise mislabeling error. Furthermore, the marginal terms are different from those in likelihood maximization – in [6], they are calculated from a Markov network where the dual variables act as potential functions.

More efficient version of both these algorithms have been described in [17]. In our experiments, parameters were learned by standard Gradient Ascent optimization to maximize the likelihood. One of applications demonstrates how learning the parameters can achieve superior performance over pre-defined penalties. We believe that, if labeled examples are available for a certain problem, learning penalty functions is a better option than choosing one from large number of possible forms.
Chapter 5

Difference and Similarities with Relevant works

In this chapter, we contrast our method with other algorithms that address closely related problems. The following sections demonstrate why existing algorithms, while being very successful for what they were designed for, are not suitable to solve the labeling problem we deal with. Specifically, we show how the proposed labeling method is different from clustering algorithms and Markov Random Field applications both in objective and in technique, i.e., in terms of what we solve and how we solve it.

5.1 Difference with Hypergraph Clustering

Given subset-wise information, hypergraph clustering methods partition the data into two (or more) subsets. Despite the apparent similarity in the goal both are trying to achieve, the proposed algorithm is fundamentally different from clustering algorithms. Hypergraph clustering algorithms utilize hyperedge weights that are computed based on similarity among datapoints in some feature space. Our algorithm computes the weights given a model associated with each of the two classes. As a result, clustering algorithms attempt to group together data samples that are ‘close’ to each other in some feature space, our algorithm puts samples that conform to some model in the same cluster. In fact, the proposed framework is more general in the sense that, we can use feature similarity as our model and can also perform a clustering based on the resulting likelihood measures. On the other hand, as we illustrate with a toy example, clustering algorithms can not solve problems that we address.

The most significant conceptual difference lies in the objective function that clustering and proposed algorithms optimizes. We explain it in the following section.
5.1.1 Objective Function

Standard hypergraph clustering algorithms either project hypergraphs to graphs and apply spectral clustering algorithms [48, 1], or modify the spectral clustering algorithm to handle hypergraphs [140]. We will explain the objective function for spectral clustering in terms of graphs to point out the characteristic that distinguishes our work from clustering. Almost all the hypergraph clustering algorithms also possess the same characteristic.

Spectral clustering algorithms minimize a product of a cut and a sum of reciprocals of associativity functions or volumes. We call the latter a restraint function. Using the foregoing notations, the definitions of these functions are given below.

\[
\text{Cut Function : } f_{\text{cut}}(x) = \sum_{i,j} \delta(x_i, x_j) w_{ij} \\
\text{Restraint Function : } f_{\text{res}}(x) = \frac{1}{\sum_i x_i d_i} + \frac{1}{\sum_i (1 - x_i) d_i} \\
\text{Objective: } \arg \min_x f_{\text{cut}}(x)f_{\text{res}}(x)
\]

(5.1)

(5.2)

(5.3)

where \(\delta(x_i, x_j) = 1\) only when \(x_i = x_j\) and zero otherwise and \(d_i\) is defined as the degree of \(v_i\) : \(d_i = \sum_i w_{ij}\).

It is important to realize that, without the restraint function, the solution is always trivial, all vertices are assigned to same cluster. Most of the spectral graph and hypergraph clustering algorithms use this (or very similar, for example [115] uses hyper-stochasticity) objective function. While the cut function \(f_{\text{cut}}(x)\) tries to minimize the strength of the connection between two clusters, the restraint function \(f_{\text{res}}(x)\) controls the size or associativity within the resulting clusters. As a result, we often observe these algorithms to produce somewhat ‘balanced’ clusters. That is, it will find subsets \(A, B\) so that not only the sum of edge weights crossing the partition is minimized, but also the connectivity among the subset members is maximized.

On the other hand, the proposed method minimizes a (weighted) summation of competing penalty functions and is not prone to ‘balance’ the size or associativity of the subsets produced by inference. For each subsets, each of the penalty functions \(g_1\) and \(g_0\) (in the definition of clique potential in Equation 3.3) tends to assign the label variables within the clique to its corresponding class, 1 and 0 respectively. The inference algorithm attempts to find the optimal
balance among the clique potential defined over all the cliques. As opposed to the clustering algorithms, our method does not try to enforce any ‘balance’ in the sizes or associativities of two partitions. That is why, the proposed method is capable of recovering disproportionate groups among the dataset where the clustering algorithm will fail due to its balancing tendency.

Next, we show a toy example where clustering algorithms can not produce useful result for one instance, model estimation, of the problem class we are addressing.

5.1.2 A Toy Example

Let us suppose we have points in 2D as displayed in Figure 5.1 and we wish to determine which 2D points belong to the lines.

![Figure 5.1: A toy example for two line estimation. Blue circles: input 2D point, red squares: detected line points. Left: input, middle: hypergraph clustering result, right: proposed algorithm.](image)

We follow a standard procedure for line fitting: first, k datapoints \((k \geq 2)\) are sampled randomly, then a line is estimated through these points and the percentage (also called inlier ratio) of all points approximately falling on the line is computed. This inlier ratio is used as hyperedge weights. We wish to verify if hypergraph clustering method is able to separate the line points from noises given these weights. For this example, we adopt the hypergraph clustering method described in [140].

The partitioning result of [140] is shown in Figure 5.1 (middle). The clustering algorithm separates the points into two clusters, each having one line and many noisy samples\(^1\). This perfectly conforms with the discussion above, the inter-cluster connectivity between the two

\(^1\)The same experiment was carried out with simple clustering algorithm with spatial proximity as the edge weights. The graph clustering algorithm also produced two clusters each with one line and many nearby noise samples.
clusters is minimum and each of the clusters has high internal associativity among the line points. None of the two lines can be estimated from this output generated by hypergraph clustering. We ideally want the line points to be separated from the noise, which is the output of the proposed algorithms, as shown in Figure 5.1 (right).

5.2 Markov Models

It is important to understand the difference between traditional applications of Markov Networks in vision\(^2\) and the proposed labeling algorithm especially because our method is also based on a general Markov structure. As opposed to the traditional Markov models the proposed method emphasizes more on higher order information, uses a hypothetical neighborhood structure not based on spatial proximity and proposes a novel clique function for our labeling problem.

5.2.1 Emphasis on Pointwise information

The fundamental difference between our algorithm and higher order Markov models lies in the manner higher order information is used. Most of these applications rely heavily on pointwise information for labeling; higher order clique functions are primarily used as smoothness factors. As an example, we describe an example problem, image segmentation, using CRF as described in [67]. Given an image as shown in Figure 5.2 (a), the objective is to divide the image into several spatially and perceptually coherent regions. To achieve this, the work of [67], as well as most other MRF applications (e.g., [108]), apply a pointwise texture (or any other region) detection algorithm to generate pointwise scores as displayed in Figure 5.2(b). Higher order information is then used to smooth out the isolated regions or discontinuities and produce the final result 5.2(c). Higher order information alone can not be used to compute the labeling in most vision application studies.

Our problem setup is substantially different from these scenarios, since we may not have pointwise information at all. Most existing higher order clique function will result in trivial

\(^2\)Recall that the models, that we describe here, are CRF models to be precise. In vision problems, MRF and CRF models are used synonymously, see Section 2.2.2.
solution in these cases. In the next section, we elaborate this difference, in more technical details and examples and explain why a different kind of clique potentials like the proposed one is necessary.

5.2.2 Difference in Higher Order Clique Potential

Studies with higher order Markov models generally consider pointwise potential functions as the ‘driving force’ or the main contributing factor for inference. Higher order clique functions act as smoothness terms designed to ensure consistency over the neighboring locations or to remove isolated regions. These functions takes the minimum value when all the labels in the clique are equal, without any bias to any particular class, and increases with the label disagreement among the variables. Examples functions are truncated linear combination of labels in the clique [74] or minimum of several truncated linear functions [65]. Without pointwise energies, these higher order potentials will lead to a trivial solution and assigns all the labels to one of the classes.

To illustrate this point, we will compare the forms of existing higher order clique potentials and the proposed one. Authors of [66] proposed a robust higher order submodular potential that was shown to produce excellent results for image segmentation using higher order cliques [67]. The higher order clique function defined in [66] is the truncated minimum of a non-decreasing concave function \( F_c(\cdot) \) of \( k - \eta_c \), where \( c \in \{0, 1\} \) in our notation.

\[
E_{\text{smooth}}^k(X^k; V^k) = \min \{ \min_{c \in \{0, 1\}} F_c(k - \eta_c), \gamma_{\text{max}} \}. \tag{5.4}
\]

However, the paper [66] later concentrates on a simpler form for \( F_c \) where it varies linearly
between \([\gamma_c, \gamma_{max}]\), with \(c \in \{0, 1\}\). Let us use the linear form of \(\mathcal{F}_c = \gamma_c + \frac{\gamma_{max} - \gamma_c}{k} (k - \eta_c)\) with parameters \(\gamma_c = 0\) and \(\gamma_{max} = \lambda(V^k)\) for class 1 and \(\gamma_{max} = 1 - \lambda(V^k)\) for class 0 \([67]\). In [65], a higher order version of Potts model was introduced as follows

\[
E^k_{potts}(X^k; V^k) = \begin{cases} 
\kappa_{min}, & \text{if all labels same,} \\
\kappa, & \text{otherwise} 
\end{cases}
\]  

(5.5)

where \(\kappa \geq \kappa_{min}\). The higher order clique function of [74] is also designed as a smoothness factor that penalizes the label difference in a particular spatial order. Since do not want to impose any order constraint on our cliques, it is not very relevant to the discussion here.

In Figure 5.3, we plot the potential clique value against \(\eta_0\) (recall \(0 \geq \eta_0 \geq k\) ) values for different definitions of potential functions: proposed (left), [66] (middle) and higher order Potts [65] for a specific clique. The first row of the images correspond to case where the likelihood value for class 1 is larger than that for class 0, i.e., \(\lambda(V^k) > 0.5\). The second row correspond to the opposite case, \(\lambda < 0.5\).

---

This is how the authors of [66] defined \(\gamma_{max}\) in [67]. A non-liner \(\mathcal{F}_c\) or constant \(\gamma_{max} = 1\) would not change the property that we are interested in.
As can be seen, regardless of the value of $\lambda(V^k)$, for all $V^k \in \mathcal{V}^k$, the Potts potential and that of [67] attain a minimum at both $\eta_0 = 0$, i.e., when all labels are assigned 0, and at $\eta_0 = k$ when all of the labels within the clique is assigned to class 1. Without any pointwise potential function, the resulting labeling will always be trivial for this kind of clique function and will not be useful in hypergraph labeling where we usually do not have (or want to use) higher order information. On the other hand, the proposed clique function is a summation of two penalty functions (shown in red and blue lines) and will never be equal to 0 for any clique. It is this property of the proposed function that prevents the method from producing trivial solutions. It adopts two different forms, upward or downward, in response to the value of $\lambda(V^k)$. It imposes a lower cost on smaller values of $\eta_0$ (that is, tries to minimize $\eta_0$) when $\lambda(V^k) > 0.5$ (top row). When $\lambda < 0.5$, it encourages labels to assume a label 0 by reducing the costs on larger values of $\eta_0$ (bottom row).

In fact, the clique function of [67] can generate a trivial solution even when the pointwise information is available but not strongly discriminative. In what follows, we present a toy example of such scenario where the higher order model of [67] fails but proposed method is able to recover the non-trivial solution.

**Toy Example:**

Let us assume a toy dataset with 5 samples that we wish to label as 0 or 1. The energy to be minimized to produce this labeling comprise only pointwise and triplet-wise costs. The penalties or costs to assign any sample to a specific class is listed in Table 5.1. Three groups of size $k = 3$ are sampled from this dataset and their weights (as defined in first paragraph of Section 3.2) are given in Table 5.2. These weights will be utilized to compute the cost as defined in Equation 5.4 (which is different from the cost suggested by the proposed clique function defined later in Section 3.3).

<table>
<thead>
<tr>
<th>Cost</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>$v_4$</th>
<th>$v_5$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$E^1(1)$</td>
<td>0.1</td>
<td>0.1</td>
<td>0.1</td>
<td>0.45</td>
<td>0.45</td>
</tr>
<tr>
<td>$E^1(0)$</td>
<td>0.9</td>
<td>0.9</td>
<td>0.9</td>
<td>0.55</td>
<td>0.55</td>
</tr>
</tbody>
</table>

Table 5.1: Pointwise costs for toy problem for which existing higher order MRF fails.
<table>
<thead>
<tr>
<th>Weight</th>
<th>{v_1, v_2, v_3}</th>
<th>{v_1, v_4, v_5}</th>
<th>{v_2, v_4, v_5}</th>
</tr>
</thead>
<tbody>
<tr>
<td>\lambda_1</td>
<td>0.9</td>
<td>0.2</td>
<td>0.2</td>
</tr>
<tr>
<td>\lambda_0</td>
<td>0.1</td>
<td>0.8</td>
<td>0.8</td>
</tr>
</tbody>
</table>

Table 5.2: Groupwise weights for toy problem for which existing higher order MRF fails.

The pointwise costs in Table 5.1 indicate that the first three samples $v_1$, $v_2$ and $v_3$ should be labeled as category $A$ since their $E^1(1)$ costs are much lower than $E^1(0)$. However, $E^1(\cdot)$ costs are not decisive for labels of $v_4$ and $v_5$. Table 5.2, on the other hand, tells us that $v_1$, $v_2$ and $v_3$ should be put in the category $A$ since the weight $\lambda_1$ is much larger than $\lambda_0$. But, the datapoints $v_4$ and $v_5$ should not be labeled the same as $v_1$ and $v_2$ since the $\lambda_1$ weights for the triples are much lower than corresponding $\lambda_0$.

Observe, first of all, that without the pointwise penalties, the clique function defined in Equation 5.4 will always produce trivial labeling, i.e., it will assign all datapoints to $c^* = \arg \min_{c \in \{0, 1\}} F_c$. For linear form of $F_c$, the label will be the one corresponding to minimum $\gamma_c$. The min operator over $F_c$ plays the pivotal role for deciding the labels in this case. Therefore, the result would not change even if we use a non-linear $F_c$.

The clique function $E^k_{smooth}$ of [66] generates a trivial solution even when the pointwise costs are incorporated. Let us use the linear form$^4$ of $F_c = \gamma_c + \frac{\gamma_{max} - \gamma_c}{k} (k - \eta_c)$ with parameters $\gamma_c = 0$ and $\gamma_{max} = \lambda_1$. The resulting label will be all ones, 1, 1, 1, 1, 1. Due to the min operators, the result does not change if we use non-liner $F_c$ or constant $\gamma_{max} = 1$ or if we use $\gamma_{max} = \lambda_0$ for $F_0$ and $\gamma_{max} = \lambda_1$ for $F_1$. With this clique function, changing parameter values would not be able to produce a non-trivial solution either. This higher order clique function $E^k_{smooth}$ was designed as a smoothness function in [67, 66]. It is not suitable for situations where we only have higher order information about the data, e.g. model estimation from noisy data. Moreover, in presence of both pointwise and higher order costs/likelihoods, apparently it cannot fully utilize the higher order information$^5$. However, it motivates us to propose the following clique function.

$^4$This is how the authors of [66] defined $\gamma_{max}$ in [67].

$^5$We show one more interesting toy example in supplementary material where the clique function in Equation 5.4 produces trivial solution.
Finally, it is worth noting that the proposed form of clique function with concave $g_c(\cdot)$ is also substantially different from that proposed in Theorem 1 of [65]. The objective of [65] was to transform higher order interaction into pairwise ones so that one can apply Graph-Cut [11] method for inference. The authors of [65] prove (in Theorem 1 of the paper) that higher order clique functions defined as concave function over summation of pairwise functions (between samples within the clique) are submodular. We are showing that summation of concave functions defined over the whole clique is submodular. Due to Jensen’s inequality, these two functions will not be equivalent even if we are able to find submodular pairwise functions for clique function in Equation 15 in [65]. Since we are not using (and there is no strong reason why we should use) graph-cut for our labeling, it is not important to reduce the higher order potentials into pairwise ones.

5.3 Difference with RANSAC-family

Standard model estimation like RANSAC and its variants also work on subsets of observations sampled randomly from the whole dataset. Given a noisy dataset of many observations, the task of a model estimation algorithm is to identify those (observations approximately satisfying the model, also called inliers). This is the problem the proposed method can be applied to solve. The proposed algorithm can solve any problem that RANSAC and its variants can handle. Moreover, our algorithm can also solve problems where RANSAC can not be used. In the following sections, we explain one such scenario, where RANSAC can not be used, and then point out another conceptual difference between the methods they follow.

5.3.1 Local vs Global Error

RANSAC, and its variants, estimate candidate models from each sampled subsets and compute the goodness-of-fit measures for each of them. To compute the correctness of the estimated model, RANSAC-type algorithms need to evaluate it against the whole dataset. That is, the error measure for RANSAC family is global. These algorithms can not work with local error measure, i.e., with accuracy of the model only with respect to the sampled subset $V^k$. This is the reason why RANSAC can not be used with part-based models.
On the other hand, the proposed method can work with both global and local error measures (we show it experimentally in Chapter 6). That is, to apply our algorithm, one may choose to evaluate the model estimated from random data subset against the whole dataset (global fit) or against the very subset it has been estimated from (local fit)\(^6\). For example, to estimate a line from noisy data, one may sample three points at random, fit a line through these points and then compute number of inliers, i.e., observations approximately satisfying the line, out of all observations in the dataset (global) or that out of these three observations that were used to estimate the line itself.

Local models can be useful for estimating multiple instances or partially observable instances of a model. Once the inliers are identified, the correct form of the model is estimated from these inliers only. If there are inliers from multiple instances (of the same model) present, one needs to resort to post-processing methods to compute the association between inliers and the model instances. For example, to identify multiple geometric models within an image (e.g., lines, circles), a connected component search may be useful to separate different instances.

Another scenario where local modeling can be useful is where it is known or assumed that there exists some geometric (or other) relationship among local parts of the object or pattern we wish to estimate. For example, the image of a bottle has edge segment constellation of certain shape on different local regions, e.g., at the bottom there must an L-shaped group of edges. The proposed method can also use information such local arrangement of parts to infer the whole object from image. Another example is non-rigid feature correspondence problem, where it is not feasible to model the transformation or relation between two entities as a whole. But, it is assumed that, up to a certain extent, the local arrangement of the parts should remain similar. None of the RANSAC-family algorithms can not work in setting because there are not model or procedure for the whole object or entity. Whereas Our algorithm can work with models defined locally, the label consistency among the members of overlapping subsets will ensure overall optimal labeling in this case.

---

\(^6\)For local evaluation, it is necessary to select data subsets of a size larger than the minimum size to compute a least square, or other error, measure.
5.3.2 Inlier Accumulation

RANSAC and variants follow a ‘replacement’ strategy to determine the largest set of inliers (refer to Chapter 6 for definition, if necessary). These algorithms always retain the largest set of inliers and discard the estimation results from all previous subsets. The proposed method aggregates the results of all past model estimation results in order to obtain a better decision about which datapoints in the set should be labeled inliers. Intuitively, it is not required for the proposed method to have at least one subset containing only inliers. If subsets with one or more inliers can estimate the model approximately well enough, the proposed method is able to extract more inliers than RANSAC since the proposed method accumulates the information from many such subsets whereas RANSAC replies on only one. It is also possible, to the parameters of our method for specific output requirements, e.g., minimizing false alarms while tolerating higher missed inlier rates.

We illustrate the difference between the procedures of the proposed method and RANSAC type algorithms in Figure 5.4. In this figure, we show the steps of the proposed method that are different from those in RANSAC in blue lines. These blue lines point out how the computation of likelihood measure and the optimal label configuration are fundamentally different from RANSAC.
<table>
<thead>
<tr>
<th><strong>RANSAC</strong>: Given N observations,</th>
<th><strong>Proposed</strong>: Given N observations.</th>
</tr>
</thead>
<tbody>
<tr>
<td>A. Repeat M times</td>
<td>A. Repeat M times</td>
</tr>
<tr>
<td>1. Sample a subset of size k.</td>
<td>1. Sample a subset of size k.</td>
</tr>
<tr>
<td>2. Fit a model on this subset.</td>
<td>2. Fit a model on this subset.</td>
</tr>
<tr>
<td>3. Calculate % of inliers</td>
<td>3. Compute likelihood values from fitting points to this model</td>
</tr>
<tr>
<td>• Inliers --- data within a certain “distance” from the model.</td>
<td>• Global or local: can be computed from all points or only the subset itself.</td>
</tr>
<tr>
<td>4. If larger set of inliers found, replace the best model and inliers with the current one.</td>
<td></td>
</tr>
<tr>
<td>B. Return best set and model.</td>
<td>B. Compute the best set of inliers using proposed labeling algorithm.</td>
</tr>
</tbody>
</table>

Figure 5.4: Algorithmic description of RANSAC and our method. The blue lines on the right panel point out the primary differences between them.
Chapter 6

Application: Model Estimation

One important application of the proposed method is model estimation. Let us suppose that we have \( n \) datapoints \( v_i, i = 1, \ldots, n \), in some feature space. Part of these data samples are generated from some model that we wish to estimate. The following two sections describe some previous work in model estimation and how this problem can be formulated as an application of proposed method respectively. Results on model estimation are reported in Sections 6.3, 6.4 and 6.5. Section 6.5 shows some interesting results on how the proposed method identify the whole object (circle) using a local model (a curve) for its parts.

6.1 Previous work

Robust model estimation from noisy data has been studied extensively in vision community for quite some time now. Initial studies like M, L or R-estimators minimize a complex and nonlinear loss function over the whole dataset to recover the model paramters \([58, 34]\). Another type of method, Hough Transform \([34]\), attempts to utilize the modes in the parametric space to determine model parameters. Random Sample Consensus (RANSAC) \([33]\) shifted the paradigm of robust model estimation by introducing a new strategy to exploit small subsets of data instead of the whole dataset. RANSAC randomly samples minimum number of samples to estimate a candidate model from data and then repeats the process for a certain number of times (statistically determined) to extract the best possible model.

The literature of model estimation became heavily engaged in developing extended and improved algorithms based on RANSAC. Notable works like MLESAC \([122]\) and MAPSAC \([123]\) designed statistical models for errors and reported substantial improvement in inlier detection. Recent studies concentrate more on reducing number of iterations required to recover the correct model. Pre-emptive RANSAC \([91]\) use a score on the hypotheses (candidate models) to
determine the best one while PROSAC updates a refined data subset of datapoints to sample from. In order to increase the number of inliers detected, Chum et.al. [16], incorporates an additional optimization step in selection of best inliers at each iteration. A more recent study by Raguram et.al. [101] combines the idea of Preemptive RANSAC and Wald-SAC [15] in an attempt to improve the performance. We refer the reader to [14] for a survey of model estimation methods based on RANSAC.

As already discussed in Section 5.3, the proposed method aggregates the information from all the randomly sampled subsets in a substantially different manner. The proposed method is different in terms of both the hypothesis evaluation and inference technique. In the following sections, we explain how model estimation can be formulated as labeling problem using higher order information and show experimental results.

6.2 Model estimation as labeling problem

We sample \( T^k \) subsets of size \( k \), fit the candidate model to these subsets and compute the model estimation errors. In terms of our notation, all the datapoints that follow the model (also called inliers) belong to class \( A \) and rest belong to class \( B \). There exists an analytical form of the model, corresponding to class \( A \), for this problem. The value of \( k \) is kept larger than or equal to \( s \) which is the minimum number of points required to fit a model (e.g. \( s = 2 \) to fit a line).

The subsets producing an error less than a problem specific threshold \( \delta^k \) constitute the set \( V^k \). The estimation error \( \epsilon \) of any member \( V^k \in V^k \) is transformed to a likelihood measure by a suitable transformation, e.g., \( \lambda_1(V^k) = 1 - \epsilon \) if we know \( 0 \leq \epsilon \leq 1 \) or \( \lambda_1(V^k) = \exp(-\epsilon/\sigma) \).

This weight is normalized (e.g., ratio of inliers) and the weight for class be is calculated as \( \lambda_0(V^k) = 1 - \lambda_1(V^k) \). Once all the samples and their associated weights are calculated, the proposed method is applied to determine which dataset should be labeled as inlier, i.e., \( x_i = 1 \).

It is important to mention here that, in model estimation, only higher order information is available. We do not have any knowledge about how likely each sample is to follow a certain model. For linear penalty functions for the proposed clique potentials, the inference algorithm (Equation 4.5) breaks the higher order costs into pointwise costs. Reducing the higher order costs into pointwise ones is not equivalent of using unary potentials only, because we do not
have a tool to compute this unary potentials. Nonlinear penalties may not be as attractive as linear ones, since it will require sophisticated inference algorithms. But, as we will see, clique functions with nonlinear penalties offer certain advantages that could be useful in special cases.

### 6.3 Result: Synthetic data

We plotted a line submerged in noise samples (inlier ratio 40%). For the proposed method, we sampled $T^k = 2000$ subsets of size $k = 3$ and used $\delta^k = 0.5$, linear $g_c$, and $[h_1, l_1, h_0, l_0] = [1.0, 0.2, 1.0, 0]$. The model was also estimated using RANSAC for comparison. We randomly chose pair of data samples for a maximum of 2000 times and used a threshold of $\delta_{\text{ransac}} = 0.4$ (value that produced minimum error) to determine inliers from the whole dataset w.r.t the estimated model. Over 100 runs of the proposed algorithm, the average miss and FA rates were $0.07 \pm 0.03$ and $0.07 \pm 0.01$ respectively. In comparison, RANSAC achieved miss and FA of $0.14 \pm 0.03$ and $0.09 \pm 0.00$. A sample qualitative output is shown in Figure 6.1.

![Figure 6.1: Line estimation. Left- input data, Right- output from proposed method.](image)

### 6.4 Results: Fundamental Matrix

Given two images of the same scene from different viewpoint, the objective is to find the point matches that conform with the camera model expressed by the fundamental matrix. Four pairs of images from the Corridor, Valbonne, Merton II, Library datasets of the standard Oxford
database\(^1\) were used in this experiment. We selected the two images with the largest variation in viewpoint, usually the first and last images (see Table 6.1).

For the proposed method each match is considered as a datapoint. We sampled 2500 subsets of size \(k = 8\) (we know \(s = 7\) in this case). Our approach is to investigate how well the proposed method works with a given number of samples and compare the results with RANSAC with equal number of subsets. Rather than determining the optimal number of subsets required to estimate a model, we would like to compare the performance of the proposed method with RANSAC given the same subsets. One can always choose the optimal number of subsets, computed probabilistically, for RANSAC and use the same set of subsets for the proposed method. Other parameter values are, threshold \(\delta^k = 1\), and linear parameters \([h_1, l_1, h_0, l_0] = [1.01, 0.02, 1.0, 0]\) for the clique function. In this dataset, there is only one instance of the model present and we adopted a global error measure to compute the likelihood values, i.e., the model estimated from a subset is evaluated against the whole dataset. The exact formula used to compute the model fitting error is the first order geometric error or the Sampson distance [52]. The ratio of inliers is used as the weight for class \(A\) and that of outliers is used for weight of class \(B\). For RANSAC, we sampled a subset of size 8 for at most 2500 times and used \(\delta_{\text{ransac}} = 0.001\) as distance threshold. The threshold values for RANSAC and the proposed method is chosen empirically to produce lowest error rates. We also compare the performance with three other variants of RANSAC, namely MLESAC, MAPSAC and LO-RANSAC, that were shown to be able to compute higher number of inliers in the survey paper [14]. The maximum number of iterations were kept the same as that of RANSAC and parameters in original implementations were retained.

Table 6.1 shows the image names (ratio of true inliers) and the result statistics such as fraction of missed inliers (ratio of missed example over inliers only) and ratio of false positive inliers (ratio of false positives over all matches) in 100 runs for all the methods\(^2\). As we can

---

\(^1\)http://www.robots.ox.ac.uk/vgg/data/data-mview.html

\(^2\)There are several parameters, instead of a single one, in the proposed algorithm that can be tuned to produce different error values. This is why we chose a tabular format instead of a Receiver Operating Characteristics (ROC) curve.
To show the advantage of using a nonlinear penalty term, let us imagine a scenario where we wish to tolerate a large miss rate (but not too high to miss required number of samples) but attain as low false positive rate as possible. To achieve this result, we experimented with

Table 6.1: Performance comparison for Model Estimation. Each row head shows the name and indices of the image used and the ratio of correct over incorrect matches. Each column shows the mean and std deviation of the missed inliers and false positives (FP) produced by the corresponding method.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>RANSAC Miss ± FA</th>
<th>MLESAC Miss ± FA</th>
<th>MAPSAC Miss ± FA</th>
<th>LO-RANSAC Miss ± FA</th>
<th>Proposed Miss ± FA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corridor, 50/100, {000, 010}</td>
<td>0.07 ± 0.06</td>
<td>0.01 ± 0.03</td>
<td>0.04 ± 0.04</td>
<td>0.17 ± 0.07</td>
<td>0.09 ± 0.02</td>
</tr>
<tr>
<td>Valbonne, 30/90, {000, 010}</td>
<td>0.13 ± 0.02</td>
<td>0.11 ± 0.03</td>
<td>0.19 ± 0.03</td>
<td>0.19 ± 0.03</td>
<td>0.16 ± 0.02</td>
</tr>
<tr>
<td>Merton, 50/150, {002, 003}</td>
<td>0.01 ± 0.02</td>
<td>0.33 ± 0.00</td>
<td>0.00 ± 0.00</td>
<td>0.03 ± 0.03</td>
<td>0.14 ± 0.02</td>
</tr>
<tr>
<td>Library, 50/150, {002, 003}</td>
<td>0.02 ± 0.01</td>
<td>0.17 ± 0.01</td>
<td>0.02 ± 0.01</td>
<td>0.01 ± 0.01</td>
<td>0.05 ± 0.01</td>
</tr>
</tbody>
</table>
Figure 6.2: Error distribution for fundamental matrix approximation. Left: miss, right: FA, top row: Corridor, bottom row: Valbonne.

A concave $g_0$ and a linear $g_1$ on the same set of data described above. The subsets and their weights for the proposed method were generated in the exact same manner. As explained before, concave $g_c$ functions are reluctant to assign the datapoints within a clique to label into different classes. Hence, a concave $g_0$ will produce more missed inliers and less false positives.

We compare the result with a two step recursive RANSAC method with decreasing threshold. The initial threshold was taken much lower than the previous experiments to produce as low false positives as possible.

Table 6.2 shows the miss and false positive results of recursive RANSAC (along with required threshold at first step), proposed method with linear and nonlinear penalties. It is clear that, very different thresholds for different datasets for recursive RANSAC are required to achieve similar miss and false positive rate as that of the proposed method with linear and nonlinear penalties. We used the same penalty function to produce outputs for all four datasets; this implies the robustness of our method over algorithms in RANSAC family for this purpose.
Proposed method, largest # Miss= 12

Proposed method, largest # FA= 13

MAPSAC, largest # Miss= 22

MAPSAC, largest # FA= 20

miss: proposed

miss: MAPSAC

false positives: proposed

false positives: MAPSAC

Figure 6.3: Worst case qualitative performances for detecting valid correspondences between two images of Corridor sequence. Showing the largest miss+FP case. Yellow (red) line: matching point pair missed (falsely detected) by the method.

This also shows that proposed method can achieve much lower false positive rates with non-linear penalties than it linear counterpart. More results for model estimation are presented in the supplementary material.

6.5 Results: Circle/Ellipse detection

This section describes an experiment where we used a part based model to detect the full object. We detect circles/ellipses in images and synthetic 2D points (with 40% inlier ratio) given edge points detected on image. The assumption made here is a circle or ellipse is a collection of curves fitted locally on different parts of the circle and ellipse. Thus, we the model of a curve (a second order polynomial to be precise) as our local model. Give the edge point detections on the image, the goal of the proposed algorithm is to determine which of them fall on a circle
Figure 6.4: Worst case qualitative performances for detecting valid correspondences between two images of Valbonne sequence. Showing the largest miss+FP case. Yellow (red) line: matching point pair missed (falsely detected) by the method.

or ellipse.

From the edge points detected on an image, \( k = 4 \) neighboring ones are selected randomly. Roughly 10,000 such subsets were sampled for this task. Then we fit a second degree polynomial on these \( k \) points. Since the model is local, the fitting error is computed on the subset that used to estimate the curve. The local fitting error is then converted to weight using \( \exp\left(-\frac{e^2}{\sigma}\right) \) where \( \sigma = 0.8 \) and we ignored all subsets with an error larger than 1. If the estimated curve turned out to be a line, we assign a very low weight on it. In Figure 6.5, the edge points labeled as \( x_i = 1 \) by the inference algorithm with linear penalty terms (parameters \([h_1, l_1, h_0, l_0] = [1.0, 0.25, 1.2, 0]\)) are colored in red (white in the last image). It is interesting to see that the proposed method can detect all the circles (multiple ones) in the images.

It is also possible to identify the circles in these images using a model for the whole circle
Table 6.2: Performance in minimizing false positives. Each row head shows the name and indices of the image used, the ratio of correct over incorrect matches and in the second line the threshold used for recursive RANSAC, miss and FP rates. Each column shows the mean and std deviation of the missed inliers and false positives (FP) produced by the corresponding method. Note that, RANSAC requires threshold to vary between a wide range to achieve similar result of the proposed method with fixed parameters.

(as opposed to those for curves) utilizing a circle fitting algorithm [100] for model fitting procedure. The proposed method produced the same result as in Figure 6.5 when we used such model. When applied to the same dataset with the same circle fitting algorithm, RANSAC was able to extract only the largest circle. By iterating the process, each time removing the inliers of the previous step, it is possible to recover all the circles. But, RANSAC needs to know the number of instances of a model (in this case circle) present in the dataset to obtain all of them. On the contrary, the proposed algorithm was able to identify all the model instances in a single iteration. It is possible, at least for the geometric structures, to compute the number of model instances simply by running a connected component search on the positively labeled data.
Figure 6.5: Circles detected by proposed method overlaid on the image.
Chapter 7
Application: Feature Matching

Identifying feature correspondence is an important problem in computer vision (see references in [27]). Given keypoints detected in two images, the goal is to find for each keypoint in one image, the matching keypoint one in the other image. Successful feature matching techniques, as described in Section 7.1, utilize both appearance descriptors and spatial arrangement among the keypoints to establish correspondence. We will also exploit both these types of information for feature matching. Section 7.2 explains how to use our method for feature matching and the following sections report experimental results on standard datasets.

7.1 Previous work

In general, matching features using only the appearance descriptor values would produce many incorrect matches. Successful algorithms for feature correspondence combine information about both appearance and geometric structure among the feature locations. Several methods [7, 114, 80, 22, 47] utilize the pairwise geometric consistency, along with the pointwise descriptor similarity, to design a cost function which is minimized using various optimization algorithms. For example, the works of [114, 22, 80] uses spectral techniques to compute the ‘soft’ assignment vector that is later discretized to produce the correct assignment of features. These works model the appearance and pairwise geometric similarity using a graph, either explicitly or implicitly, and are commonly known as graph matching algorithms. The soft assignment vector is computed by an eigen-decomposition of the compatibility or match quality matrix. Graph matching algorithms have been for various vision problems, e.g. [81, 103].

Caetano et.al. [12] discusses how the parameters of the aforementioned cost function can be learned provided several pairs with labeled correspondences. To be precise, this work primarily learn the optimal match scores from labeled training examples to maximize matching accuracy.
A more recent work [81] proposes to learn similar matching scores, in an unsupervised fashion. The authors of [81] repeatedly refines the soft assignment vector in absence of the true correspondence labels.

Higher order relationship among the feature points have also been investigated for matching problem. Zass et.al. [137] assumes two separate hypergraphs among the feature points on two images and proposes an iterative algorithm to match the the two hypergraphs. On the other hand, Olivier et.al. [27] generalizes the pairwise spectral graph matching methods for higher order relationship among the point matches. The pairwise score matrix is generalized to a high order compatibility tensor. The eigenvectors of this tensor is used as the soft assignment matrix to recover the matches.

7.2 Correspondence as labeling problem

In our framework, each feature correspondence is considered as a datapoint and we assume a hypergraph structure among these datapoints as in [27]. For subsets of such datapoints, we assume that the relationship among features of one image follow the same geometrical model as that present among the corresponding features on the other image. We compute a likelihood, using this geometrical model, from every subset of datapoints and use is as weight of the hyperedge.

Given two images $I_L$ and $I_R$, we denote $a_l$ and $a_r$ to be the indices of feature points from $I_L$ and $I_R$ respectively. In general, the total number $n_l$ of features detected on $I_L$ is different from the number $n_r$ of features detected on $I_R$. Each candidate match $(a_l, a_r)$ is considered to be a datapoint $v_i$, $i = 1, \ldots, n$, in this paper. The goal is to partition the subset $A$ of correct correspondences from that of the incorrect ones $B$. This is a data labeling problem where the binary label $x_i \in \{0, 1\}$ of $v_i$ needs to be assigned as $x_i = 1$ if $v_i$ belongs $A$ and to 0 otherwise.

We wish to utilize the the information about subsets of datapoints to enforce geometric consistency in matching. For a subset $\{v_{i_1}, \ldots, v_{i_k}\} = \{(a_{l_1}, a_{r_1}), \ldots, (a_{l_k}, a_{r_k})\}$ of size $k$, we assume the geometric relationship among $\{a_{l_1}, \ldots, a_{l_k}\}$ to be similar to the geometric relationship among $\{a_{r_1}, \ldots, a_{r_k}\}$. This similarity value is computed by a suitable function (e.g., exponential of negative Euclidean distance between triangle parameters).
Figure 7.1: Computing weight for matching. The quantitative difference between triangles on the left and right images are converted to weights.

We attempt to visually illustrate the computation of the similarity weights in Figure 7.1. The objective of matching is to establish the correspondence between feature locations of the image shown on the left and those on the right image. For each keypoint on the left image, we estimate candidate matches for it on the right two images, for example, in Figure 7.1 top row, the potential matches are between keypoints numbered 1, 2, 3 on both images. Another set of such matches are shown in the bottom image pair. The geometry among the features in each image are captured by neighboring feature triplets, as shown in yellow triangles. The similarity of the geometry is computed using the parametric difference between the triangle among the three keypoints 1, 2, 3 on left image and those on the right image. For the two triangle pairs shown on top and bottom images, the one on top row will have a higher similarity than the pair at bottom. We use Euclidean distance between the sines of angles of the triangle to compute parametric distance.

In addition, we also compute the appearance similarity between features on two images. The final likelihood value is a product of these two similarity measures and is denoted by
\( \lambda_1(V^k) \in [0, 1] \) where \( V^k \) is the subset \( \{v_{i1}, \ldots, v_{ik}\} \). Notice that, we are effectively dealing with a hypergraph with datapoints as the nodes and the subsets \( V^k \) as the hyperedges. The labeling algorithm partition the set of nodes into two sets \( A \) and \( B \) given such hypergraph.

The penalty functions \( g_1 \) and \( g_0 \) for both the classes \( A \) and \( B \) were learned using the Maximum Likelihood (ML) estimation described in Section 4.2.1. The results on standard datasets demonstrate the robustness of the proposed method with learned penalty functions for correspondence problem (Section 7.3).

### 7.3 Results: House and Hotel

We conduct our first experiment on the standard House and Hotel datasets. Each of these datasets contains a sequence of images of a toy house (or hotel) seen from increasingly varying viewpoint. There are 111 and 101 images available for House and Hotel datasets respectively. Locations of a set of keypoints, that appear on each of the image of the sequence, are available for both these sequences. We consider four set of image pairs where each image is \{20, 40, 60, 80\} frames apart from the other. The Geometric Blur (GB) [8] descriptor is used to represent each feature. Geometric Blur descriptor was shown to be able to capture local appearance quite well in [8]. This descriptor has also been used successfully for feature matching in [7, 121]. For each keypoint \( a_l \) in image \( I_L \), \( m = 3 \) candidate matches, denoted by the set \( \mu(a_l) \), are chosen based on largest normalized correlation between the GB descriptors. Each of the candidate matches is considered to be a datapoint \( v_i \).

We construct a hypergraph of edge cardinality \( k = 3 \) with these datapoints. For each feature point \( a_l \) in image \( I_L \), all possible triangles are generated among \( a_l \) and \( k_{NN} = 5 \) nearest neighbors. Any such triangle among \( \{a_{i1}, \ldots, a_{ik}\} \), has \( k^m \) possible matching triangles in image \( I_R \) induced by the set of candidate matches \( \{\mu(a_{i1}), \ldots, \mu(a_{ik})\} \). The geometric similarity of these triangle pairs are evaluated by the sum of squared difference of the angles. The assumption made here is that, up to a certain transformation, the local geometric arrangement remains unchanged and can be captured by a similarity transformation. However, appropriate measures

---

1The likelihood for class B is once again \( 1 - \lambda_1(V^k) \)
can also be incorporated to capture the affine or projective invariance to transformation. Similar assumption has been also utilized in the tensor matching algorithm [27]. This parametric difference between triangles is converted to geometric similarity weight using \( \exp(-\frac{\epsilon}{\sigma}) \) with a \( \sigma \) value of 0.2. The appearance similarity value is the normalized correlation between two GB descriptors computed for potential matching features. Each candidate match is assigned a weight that reflects the quality of the match computed by normalized correlation. In our experiments, we used \( m \) discrete values 1, 0.8, 0.6 for \( m \) candidate matches of each keypoint.

To compute the overall similarity \( \lambda_1(V^k) \) between two triangles, the weight of corresponding matches \( \{\mu(a_{l_1}), \ldots, \mu(a_{l_k})\} \) is multiplied with the geometric similarity weight computed from parametric difference between two triangles. This weight can be viewed as a joint probability of that the triangles match given appearance and geometry. All triangle pairs with angle difference below a threshold \( \theta = 0.5 \) are retained to form the set of hyperedges \( V^k \). This construction of hypergraphs among matches follows that of [27] and [137], except [27] searches all possible triangles in image \( I_R \) instead of searching the ones induced by candidate matches.

For each of the four sets of images that are \{20, 40, 60, 80\} frame apart, we use first five pairs to learn the penalty functions. We adopted the likelihood maximization method for learning the parameters of a discrete valued penalty function. The clique potential in this setting is as follows.

\[
E^k(X^k; V^k) = \sum_c \sum_{\alpha=0}^k w^c\alpha \phi_c(\alpha, V^k).
\] (7.1)

The \( w^c\alpha \) parameters are learned for all values of \( c \) and \( \alpha \). As described in Section 4.2.1, the weights \( \lambda_1(V^k) \) are embedded in the factors \( \phi_c(\alpha, V^k) \).

For each set of image pairs, first five pairs were selected to learn the parameters for the proposed matching algorithm. We learned the parameters for a discrete \( g_c \) by maximum likelihood (ML) method (refer to Section 4.2.1). Inference is carried out by the Sum-product belief propagation algorithm. The discrete one to one assignment were calculated using the beliefs as stated in Section 4.1.2. The performance of our algorithm is compared against that of [80] and [27] (implementation available at author’s website). For Tensor matching algorithm [27], the parameter values used by the authors for number of triangles to be generated, number of nearest neighbors of each triangle and the distances produced the best results and was also used in these experiments. For the graph matching algorithm [80], we used the exact same procedure.
as described in the paper with the same $m = 3$ candidate matches for each keypoint and the used 3 as the distance threshold to determine the neighboring keypoints (also tuned for best result). Figure 7.2 shows the percentage of incorrect matches produced by the proposed method and those produced by [27] [80] and simple GB feature descriptor match for house and hotel data.

As it is clear from the quantitative results, none of the spectral graph matching and hypergraph matching techniques was able to perform well on both of these two datasets. On the other hand, the proposed method, with learned cost functions is more robust and produces very low incorrect matches for both the datasets, especially for the large frame differences. We would like to point it out here that, in [27], the authors did not report the results on all possible pairs images that are a certain number of frames away from each other. Only the best matching cases (one pair for each interval) were reported which is same as the minimum error rates (lowest points on magenta curve) for all intervals.

Figures 7.3 and 7.4 show samples of some matching results of the proposed method on relatively challenging frame pairs that are 60 and 80 frames apart respectively. On each image, the top row shows the result with lowest incorrect matches and the bottom row shows that with largest incorrect matches for all pairs in the respective sequences. The matching results on Hotel datasets are not reported to avoid repetition.
In Figure 7.5, we show the discrete $g_c$ learned by the ML algorithm for $c = 0, 1$. As expected, the learned penalty functions resembles strongly to smooth concave and convex functions. The forms of $g_c$ functions also provides some insight about the subsets generated for matching. Note that, the trend of change in $w_0$ values, corresponding to $g_0$ function, resembles a convex function. A convex penalty imposes ‘lenient’ penalties on lower values of $\eta_1$, number of label variables assuming the opposite class, class 1. This penalty function would be effective when there are many subsets comprising very few (e.g., one) correct matches. For these subsets, a convex $g_0$ would allow to let few datapoints within the subset to assume the opposite label 1. Examining the matching triangles used for matching, one can easily verify that there are indeed many subsets that contains one correct matches and two incorrect matches in them. On the other hand, the triangles with all correct matches are rare and therefore the penalty function is ‘strict’ (i.e., concave) on the value of $\eta_0$. 

Figure 7.3: Best and worst results of the proposed method on House dataset on image pairs 60 frames apart.
69

Figure 7.4: Best and worst results of the proposed method on House dataset on image pairs 80 frames apart.

7.4 Results: KTH Activity

We applied our method on more KTH activity recognition data [12]. For this dataset, we chose three activities, walking, jogging and hand waving and for each of these activities we randomly selected two sequences. The experimental setup is almost same as above except the features are detected using Kadir-Brady (KB) keypoint detector algorithms [62] on both the images, i.e., we do not manually select keypoints on image. For each keypoint selected by the feature detector (Kadir-Brady) on the left image, the goal is to find its best match on the right image. We also applied the labeling algorithm with predefined linear penalty functions and compared the results to show the improvement achieved by learning $g_c$. For the learning algorithms, discrete $g_c$ functions are learned using the ML estimation procedure as before. All parameters for both methods are same for all the experiments in this and the next sections. Sample output matches are shown in Figure 7.6. The top row shows the output produced by the proposed method using
linear penalties, and the bottom row shows the results produced by discrete $g_c$ trained from data. The matching algorithm with learned penalty function were able to extract more accurate matches than that with linear penalties.

Figure 7.5: Parameters learned by ML method, left: $w_1$, right: $w_0$.

These results demonstrate the fact that simple predefined penalty functions will not be sufficient to produce satisfactory matching among two sets of features. Table 7.1 summarizes the quantitative matching performances of these two methods. The results clearly show that hypergraph labeling with learned penalty function consistently produces better results than the same method with predefined linear penalties. It is worth mentioning here that the proposed matching algorithm was applied on the (spatially clustered) keypoint locations detected by the

Figure 7.6: Improvement achieved by learning. Top: results of proposed method using a predefined linear penalty. Bottom: matches after learning. More correct correspondences are recovered by learned penalty function.
Table 7.1: Average number of correct and incorrect matches (NOT percentages) found on the image pairs. The proposed algorithm with learned penalty functions consistently produces more true positives with less false positives on all the sequences.

KB detector, which is why there are variable number of feature locations in different images. However, for quantitative performance evaluations, we manually counted number of correct and incorrect matches from the output of each of the methods (linear and learned $g_c$).

### 7.5 Results: Caltech Aeroplane and Motorbike

Finally, we are showing some more qualitative results on Caltech objects, such as airplanes and motorbikes, in Figure 7.7. The experimental setup is exactly same as that described in the last section.
Figure 7.7: Qualitative results on Caltech aeroplanes and motorbikes.
Chapter 8

Application: Object Localization

The problem of object localization aims at identifying the location of an object in the image. Localizing the object assists object detection/ recognition techniques by providing candidate locations to evaluate the object model instead of searching all over the image. We attempt to tackle localization by a part-based technique. Two object localization experiments described here treat edge segments and spatially clustered interest point as object parts. Given many detections of object parts in the image, both on the object and the background, the goal is to identify which ones belong to object we are interested in. In the following section, we describe some of the previous works on object localization before explaining how the proposed labeling method can be used for object localization using edge segments and point detections respectively in Sections 8.2 and 8.3.

8.1 Previous work

Several studies on object localization address the problem simultaneously with object recognition. For example, [96] initially applies a multiresolution object detector to determine tentative locations in the image and then refines the set of these locations, using a coarse to fine strategy, for localization. Yeh et.al. [135] also performs a pruning over the output of a multi-resolution classifier for concurrent localization and recognition. The authors of [42], on the other hand, utilize image segments to tackle the problem. Image segmentation algorithms typically produce many more segments than the number of coherent regions in the image. These segments are combined together by a Conditional Random Field, which incorporates both contextual and spatial information about the segments, to generate labels for the segments. Heitz et.al. [55] proposes a probabilistic distribution over a set of keypoints, detected by matching with a template model, to identify the object location. The work of Crandall et al. [23] uses a parts-based
model with manually annotated parts. Their follow up work [24] automatically learns the parts, but similar to the constellation method, uses these only as cues for object detection.

Other type of studies use contiguous contour segments [31] to represent object for localization. For example, [32] generates a codebook of these segments and describe the objects as collection of codewords. The location of the object is found at local maxima of votes from these codewords in a manner similar to Hough transform [34]. Authors of [104] also work on edge segments and apply a (partial) matching of detected edge segments to a object template to compute a score. The object edge segments are later aggregated using a center voting scheme. In the proposed method, we will also use edge segments for localization, but in a manner different from all these works (as described in next section).

8.2 Localization as Labeling: Edge Segment Features

Part based representation of an object has gained much attention in recent years in computer vision. Description based on arrangement of object parts is generally invariant to occlusion, transformation and pose variation up to a certain extent [30]. We model the object as constellation of edge segments, by considering edge segments as parts. Ferrari et.al. [31] shows how subsets of edge contour segments, called k Adjacent Segments (kAS), can capture sufficient object characteristics for recognition. In [31], the authors generate a codebook of representative kAS from training images and describe the object in terms of these subset of segments. Following [31], we also describe the object using kAS codewords. Each edge segment detected is considered a datapoint \( v_i \) that we wish to label as object (class A) or background (class B) edge and each kAS is considered as a subset \( V^k \) of datapoints. However, our goal is different from that of Ferrari [31]. In [31], the goal was to detect the object from image and the authors train a Support Vector Machine (SVM) classifier on kAS description of image regions for detection. On the other hand, we wish to localize the object using the geometric information encoded in the kAS description. In fact, the object localization part can act as a preprocessing step, instead of the sliding window approach, for object detection.

During testing, the probability of any kAS (edge subset) to belong to object (background) class \( A \) (\( B \)) is computed probabilistically using these codewords. Given training images, two
Figure 8.1: Generating wts for likelihood values for groups of edges (kAS). Left: a sample code, Right: kAS overlaid (in k = 3 different colors) on test image with weights.

Sets of codewords are computed for object class and background. Let $\zeta_o$ and $\zeta_b$ are two kAS codewords computed from object and background respectively. Then the likelihood measure for any group of edges $V^k$ on the test image is the maximum joint probability that $V^k$ and any codewords appears together.

$$\lambda_1(V^k) = \max_{\zeta_o} p(V^k | \zeta_o) p(\zeta_o | o)$$  \hspace{1cm} (8.1)

$$\lambda_0(V^k) = \max_{\zeta_b} p(V^k | \zeta_b) p(\zeta_b | b)$$  \hspace{1cm} (8.2)

The two priors $p(\zeta_o | o)$ and $p(\zeta_b | b)$ in Equations 8.1 and 8.2 quantify the discriminative power of the codewords and are computed offline. The conditional probabilities $p(V^k|\zeta_o)$ and $p(V^k|\zeta_b)$ are computed using the parametric distance between the kAS $V^k$ and the code $\zeta_o$. We used the same distance kAS function proposed in [31]. The distance is converted to probability (non-normalized) using $\exp\left(-\frac{\epsilon^2}{\sigma^2}\right)$ where $\epsilon$ is the kAS distance and $\sigma = 2$ for 3AS and $\sigma = 3$ for 4AS. We show a sample codeword computed from an object class (bottle) in the left column of Figure 8.1. The right column shows example kAS detected on test images and the likelihood values computed according to Equations 8.1 and 8.2. The kASs with similar shape as the codeword on the left column are assigned larger weights (top row, right column) than those assigned to kASs with different shape than the code (bottom row, right column). Though in this examples, the weights are informative enough to identify object edges, there will be many kASs that produce noisy weights. It will be shown in the results that only likelihood values of
edge subsets are not sufficient to correctly identify the object edges.

Given these likelihood values of test image kAS, we apply the proposed method to identify all the edges that should belong to the object, i.e., all the edges labeled as $x_i = 1$. It is worth mentioning here that, in this case we have two separate models for classes $A$ and $B$ to compute the likelihood values.

### 8.2.1 Results

The experiments were conducted on three objects of the the ETH shape dataset, namely applelogos, bottles and swan. For each type of object, half the images for each category were selected as training images. Out of all 3AS ($k = 3$) descriptors in the training set, we select representative ones to be the centers of a hierarchical clustering output. These representative descriptors, or codewords, are generated for both foreground and background.

During testing, we generate all the 3AS from the test image. For each 3AS, we generate the likelihood value for the edge segments to belong to the object(background) to be the distance from the closest object(background) code. As already stated, there are two (implicit) models in this experiment, one characterizes the object and the other correspond to the background. Finally, we run the proposed labeling algorithm to get the edges of an object. We employed both linear and nonlinear penalty functions in this experiment. For nonlinear $g_c$ functions we used concave functions for both the classes. As described earlier in Section 3.5, concave penalty functions are conservative in terms of labeling the datapoints within clique. Concave $g_c$ imposes larger penalties on labeling the datapoints to the opposite class than other forms of penalty functions. Experimentally we have found that strict penalty functions are more effective to label the edges accurately for both object and background classes.

Figures 8.2 and 8.3 show some output of proposed method for identifying object edges. The input and detected edge segments are overlaid on the images. Top row displays the input edges. To clarify that only kAS likelihoods , without the labeling technique, are not sufficient to identify the object edges, we show all the edges that belong to kAS segments that have $\lambda_1(V^k) > \lambda_0(V^k)$ on the second row from top. The proposed labeling algorithm with nonlinear penalties (fourth row) more accurately identifies the object edges from the background than that with linear penalties (third row). As explained before, concave penalty functions tend to
Figure 8.2: Qualitative performances of proposed method for detecting edges on objects (from left to right) applelogos, swan. Rows: top to bottom, input edges, edges with $p(kAS \in \text{object}) > p(kAS \in \text{bckgnd})$, proposed linear, proposed nonlinear.

prevent breaking up the clique members into different classes. As a result, the proposed method with nonlinear penalties attempts to attain more edges on the object and reduce the number of background edges to be labeled as 1 than its counterpart with linear penalties.

Table 8.1 summarizes average number of edges detected by these three methods on the object and outside of the groundtruth bounding box for the whole dataset. In comparison, proposed method with nonlinear penalties is able to detect more object edges with fewer false alarms than other two techniques describe here.
8.3 Localization as Labeling: Point Features

This section describes how to localize the object in an image by considering interest point detectors as parts. The candidate locations or parts are spatially clustered Kadir-Brady salient region centers [62]. Each candidate location is a datapoint for the problem and the algorithm should detect which of these datapoints belong to the object. We select subset of these parts and model their geometric properties by appropriate measures from all the training images. This model will be utilized to generate weights for subsets similarly sampled from detected parts in a test image. Then, our inference method will be used to prune out parts detected in the
Table 8.1: Number of edges detected by three methods, kAS likelihood only, proposed-linear and proposed-nonlinear. For each object class, the columns show average number of detected edges within and outside of the groundtruth bounding box.

<table>
<thead>
<tr>
<th>method</th>
<th>Bottles in</th>
<th>Bottles out</th>
<th>Applelogos in</th>
<th>Applelogos out</th>
<th>Swan in</th>
<th>Swan out</th>
</tr>
</thead>
<tbody>
<tr>
<td>kAS Pr</td>
<td>9.16</td>
<td>7.2</td>
<td>5.9</td>
<td>8.72</td>
<td>6</td>
<td>9</td>
</tr>
<tr>
<td>linear</td>
<td>8.83</td>
<td>6.86</td>
<td>5.81</td>
<td>7.09</td>
<td>5.06</td>
<td>4.85</td>
</tr>
<tr>
<td>nonlin</td>
<td>8.16</td>
<td>6.66</td>
<td>5.77</td>
<td>6.95</td>
<td>4.81</td>
<td>4</td>
</tr>
</tbody>
</table>

The Geometric Blur descriptor [8] is used for these datapoints to create a codebook or bag of features (in a reduced dimensional feature space obtained through PCA). The parts are represented by the most similar codeword in the dictionary of codes. We sample size $k$ subsets $\{v_{i_1}, \ldots, v_{i_k}\}$ of datapoints and represent them using parameters of geometric relation present among the members of the subset. Let $\pi$ be a permutation of $i_1, \ldots, i_k$, which orders $v_{\pi(l)}$, $1 \leq l \leq k$ in increasing x-coordinate; $\zeta = [\zeta_l]_{l=1}^k$ where $\zeta_l$ is a code that represents $v_{\pi(l)}$; and $\chi$ be a vector of geometric parameters for the relation present in $v_{\pi(l)}$.

Now, we wish to derive a probability for $\chi, \pi$ given object model description $\Theta_\chi^o, \Theta_a^o$ corresponding to geometric and appearance parameter for the object. The probability $p(\chi, \pi | \Theta_\chi^o, \Theta_a^o)$ can be expressed as a joint probability of $g$ and $\pi$ occurring together given the model.

$$p(\chi, \pi | \Theta_\chi^o, \Theta_a^o) = \sum_{\zeta} p(\chi, \pi | \zeta, \Theta_\chi^o, \Theta_a^o) p(\zeta | \Theta_\chi^o) = \sum_{\zeta} p(\chi | \zeta, \Theta_\chi^o) p(\pi | \zeta, \Theta_a^o) p(\zeta | \Theta_a^o).$$

We assume independence between geometric parameters $\chi$ and relative positions $\pi$ given ordered code representation $\zeta$, among $\pi_l$, $l = 1, \ldots, k$ given $\zeta_l$ and among $\zeta_l$ themselves. Therefore, the above expression can be simplified as follows.

$$p(\pi | \zeta, \Theta_a^o) = \prod_{l=1}^k p(\pi_l | \zeta_l, \Theta_a^o) \quad \text{and} \quad p(\zeta_l | \Theta_a^o) = \prod_{l=1}^k p(\zeta_l | \Theta_a^o). \quad (8.3)$$

The probability $p(\chi | \zeta, \Theta_a^o)$ for geometric parameters are estimated by Kernel Density Estimation (KDE) with diagonal bandwidth matrix. The probability $p(\pi_l | \zeta_l, \Theta_a^o)$ is modeled using the feature descriptors $\phi(v_{\pi_l})$ and $\phi(\zeta_l)$ for $v_{\pi_l}$ and $\zeta_l$ respectively, i.e., $p(\pi_l | \zeta_l, \Theta_a^o) \sim N(\phi(\zeta_l), 0.2I)$ where $I$ is the identity matrix. The priors $p(\zeta_l | \Theta_a^o)$ on the codes are computed empirically counting number of times $\zeta_l$ appeared in object.

An analogous model is learned for background subsets of parts. For each test image,
subsets of size $k$ is collected the same way. The likelihood weight for object class $A$ is
\[ \lambda_1(V^k) = p(\chi, \pi \mid \Theta_{A}^{\theta}, \Theta_{a}^{\theta}) \]
and that for background class $B$ is
\[ \lambda_0(V^k) = p(\chi, \pi \mid \Theta_{B}^{\theta}, \Theta_{b}^{\theta}) \]
where $\Theta_{A}^{\theta}, \Theta_{a}^{\theta}$ describe the background model. A group will have a large likelihood weight
\[ \lambda_1(V^k) \]
if most of the codes representing the object parts have high prior probabilities and the geometric arrangement among them also has high likelihood value given object model.

Given these weights, we run the proposed inference algorithm with both linear and nonlinear penalty functions $g_c(\cdot)$, $c \in \{0, 1\}$ to decide which parts $v_i$ should have $x_i = 1$, i.e., which part should belong to the object.

### 8.3.1 Results

The datasets used for this experiment are the Caltech cars (rear, with 126 images), airplanes images. Each dataset is split into two subsets to be used for training and testing. The training datasets are used to learn the codebook and object model probabilities. For both object and background parts, we identify the 3 nearest neighbors and sample subsets of sizes $k = 3$ respectively out of the part and its neighbors. The lengths of the sides of the triangle generated by subsets of size $k = 3$ (normalized by the perimeter) are used to encode the geometrical information among members of the group in addition to the internal angles.

We compared the results with a naive approach where the empirical probability of the code representing each part is used to determine its label. That is, the label $x_i$ of any $v_i$, is set to 1 if
\[ p(\zeta_i \mid \Theta_{a}^{\theta}) > p(\zeta_i \mid \Theta_{b}^{\theta}) \]
where $\zeta_i$ is the code representing $v_i$, and set to 0 otherwise. This method will be referred as Code Prior procedure hereafter.

Without an explicit model description, it is easy to realize that RANSAC can not be applied here. Furthermore, as we discussed before, the algorithms for higher order MRF inference in [66, 65, 67] can not be applied in this object model without pointwise costs. We utilized the the empirical code probability in order to employ the technique of [66] for Graph-Cut method with swap move in the present scenario with binary labels and only point-wise and triple-wise weights. Each location $v_i$ is considered as a node in the graph and its point-wise cost is inversely proportional to corresponding $p(\zeta_i \mid \Theta_{a}^{\theta})$ and $p(\zeta_i \mid \Theta_{b}^{\theta})$. The max weight for a 3-way clique, denoted by $\gamma_{\text{max}}$ in [66, 66], is set to $\sum_{\zeta} p(g \mid \zeta, \Theta_{a}^{\theta}) p(\pi \mid \zeta, \Theta_{a}^{\theta})$ for object class and $\sum_{\zeta} p(g \mid \zeta, \Theta_{b}^{\theta}) p(\pi \mid \zeta, \Theta_{a}^{\theta})$ for background class. This conforms with how the authors
of [66] defines $\gamma_{\text{max}}$ to be proportional to a goodness measure of the corresponding clique in Equation 12 of [67]. The higher order terms of [66] are distributed into members of the clique via an auxiliary variable. This is different from the manner that we break the higher order terms, compare Equation 4.5 of this article with Theorem 1 and Figure 3 of [66]. We used the parameters that produced best results for both the proposed method and for the method in [66].

For quantitative comparison, we plot the average number of images with a hit against the fraction of false detections allowed per image. If at least 5 of detected object parts fall within the object boundary of an image, we count it as one hit. The results are shown in Figure 8.4. For the car and airplane datasets, the code prior probabilities (black curve) are clearly not sufficient to identify the object parts. The proposed method, with linear (blue) and nonlinear (cyan) penalties, attains a higher hit rate than both the Code Prior approach and that of [66] (red) at lower false positives. As discussed before, higher order information is not used for discrimination in [66]; it is rather used as smoothness constraint. On the other hand, the proposed method uses the higher order information of a group for discrimination.

We are also showing some qualitative results in Figure 8.5. The white circles indicate possible locations of object parts, The red shows the detected parts by different methods. The results of code prior, [67], proposed linear and proposed nonlinear methods are arranged in columns from left to right. The proposed method is able to remove many false positive detections in the background retained by other two algorithms on left two columns.
Figure 8.5: Qualitative object localization results- from left to right columns: the result for code prior, that of [66], the proposed method with linear and that with nonlinear penalties respectively. The circles (filled with white) indicate the initially detected (spatially clustered) locations. The red * indicate the object parts detected by respective method.
Chapter 9

Conclusion

This study proposes a new labeling method for hypergraph structured data, where the primary information utilized for labeling is a model generated likelihood measure. More specifically, we would like label data samples into two classes given only (or primarily) higher order information about them. The higher order information is defined as a likelihood measure for the subsets to belong to each of the classes. An analytical or algorithmic procedure for each of these two classes are assumed to be available, but is not used for deciding the labels of data.

We model this problem with a general Markov Network. A hypothetical neighborhood is defined for the Markov Network to handle this sort of data. A novel clique function, suitable for labeling tasks, is also proposed. This clique function is composed of two penalty functions, each trying to label the examples to its corresponding class. Consequently, the overall cost function to be minimized to retrieve the labeling becomes a summation of penalty functions of two competing types corresponding to each of the classes.

The behavior of this clique function (or the penalty functions to be precise) is analyzed to suggest appropriate forms required for different problems. It has also been investigated when the overall cost function becomes submodular and therefore is guaranteed to produce the optimal result. We also discuss how to learn the penalty functions from labeled data of a specific problem instance.

The tasks we address to solve is different from the goals of hypergraph clustering and higher order Markov Random Field models. In contrast to clustering algorithms, we do not impose a factor prone to balance the cluster sizes or associativity between clusters. Unlike Markov Random Field models, the proposed algorithm does not utilize higher order information as smoothness factors. The labeling method is also more general than model estimation problems of RANSAC family.
The proposed labeling method has been applied to several different vision problems. Our method can be applied to estimate models from noisy datasets. For problems with analytical forms of model, e.g., fundamental matrix approximation, proposed grouping algorithm produced comparable results to existing techniques. Our algorithm can also identify geometrical models like circles and ellipses from the data. The proposed technique can also address problems with part-based models, i.e., a geometric model defined on local arrangement of object parts. It has been applied for object localization using this type of models.

Our labeling algorithm has also been applied for feature correspondence problem. A penalty function was learned from labeled matching features to address this problem. Experimental results suggest the resulting matching algorithm performs better than current state of the art matching algorithms.

This dissertation proposes a new labeling or grouping method using higher order information. This labeling algorithm was designed to solve problems in a general setting, i.e., our method is able to label the examples to maximally conform with given higher order weights. It has been shown to successfully address different types of vision problems. We anticipate this method to be an important contribution to vision community for labeling problems.
Appendix A

Boosting Adaptive Linear Weak Classifiers for Online learning and Tracking

Abstract

Online boosting methods have recently been used successfully for tracking, background subtraction etc. Conventional online boosting algorithms emphasize on interchanging new weak classifiers/features to adapt with the change over time. We are proposing a new online boosting algorithm where the form of the weak classifiers themselves are modified to cope with scene changes. Instead of replacement, the parameters of the weak classifiers are altered in accordance with the new data subset presented to the online boosting process at each time step. Thus we may avoid altogether the issue of how many weak classifiers to be replaced to capture the change in the data or which efficient search algorithm to use for a fast retrieval of weak classifiers. A computationally efficient method has been used in this paper for the adaptation of linear weak classifiers. The proposed algorithm has been implemented to be used both as an online learning and a tracking method. We show quantitative and qualitative results on both UCI datasets and several video sequences to demonstrate improved performance of our algorithm.

A.1 Introduction

Studies on online learning algorithms originated in computational learning community. The initial algorithms train several experts based on the labeled samples arriving sequentially and later combine the predictions of these experts to categorize any new example. The algorithms popular to the machine learning researchers like the weighted majority algorithm or winnow
algorithm, as discussed by Littlestone and Warmuth in [84] and by Littlestone in [83] respectively, belong to this family\(^1\). Both the weighted majority and winnow algorithms works as a committee of hypotheses to classify target samples. The popular (offline) Adaboost classifier [37] resembles these classifiers in the sense that it also combines several ‘weak’ hypotheses in classifying new observations. An online version of the boosting classifiers in has been recently proposed in a relatively recent study by Oza et.al [94].

Online learning algorithms can be immediately applied to the object tracking scenario. The tracker employs an online learning method to learn the object from frames arriving sequentially and then apply the classifier to detect the target in the next frame. Recently, the work of Avidan [5] and Grabner et.al. [49] has shown impressive results of using classifier based tracking methods. The classifier based methods initially learn a binary classifier to distinguish the object of interest from the (neighboring) background and then apply it at each new frame to locate the position of the object. Both the works of [5] and [49] uses AdaBoost [37] or an online variant of AdaBoost classifier.

Several previous works have addressed object tracking problem by approximating the distribution of feature responses representing an object using Kalman Filter [25], Particle Filter [59], Mean-shift method [19] etc. Density approximation tracking algorithms result in inferior performances to classifier based tracking algorithms in the cases when the target appearance undergoes substantial change over time or when there are similar objects nearby [5]. Results of our experiments will show how the proposed method is able to track objects in such scenarios where meanshift tracker fails.

One important issue to resolve in classifier based trackers is how to adapt with scene change and remain capable of identifying the object correctly. We should keep in mind that both the background and the object of interest may change with time. The previous studies [5, 49] replace a subset of current weak classifiers with a new one to cope with scene changes. Ensemble tracking [5] does not update the weak classifiers themselves. Instead at every frame, it replaces some of the older weak classifiers with several new weak classifiers. Grabner’s online boosting method [49] models the feature densities by simple Gaussians and update their

\(^1\)A more detailed discussion on online learning algorithms is provided by Blum in [9]
parameters at each frame using Kalman filtering method. Even if we assume that a Gaussian distribution is sufficient to model feature densities (which is usually not the case), the updating mechanism soon becomes complicated and slow if we wish to use higher dimensional weak classifiers. Furthermore, the method of [49] immediately throws away any weak classifier generating an error greater than or equal to 50%. To fill up the space evacuated by the leaving weak classifier, the methods in both [5] and [49] have to search a large subset of weak classifiers for replacement.

The primary contribution of this paper is an adaptation scheme for the weak classifiers themselves to conform with the changes over time. Similar to the method of ensemble tracking [5], we combine linear weak classifiers, learned in a least square fashion, and learn incrementally (online boosting). However, we neither replace weak classifiers for each data set nor do we throw out any weak classifier during training phase. Instead, the proposed method modifies the internal parameters of the base learners for the final classifier to blend with the change as long as the base learner has an error rate below 50%. The online boosting algorithm described in this paper uses linear regressors as the base learners. The adaptation scheme does not require us to keep the previous examples in the memory and does not need complex filtering techniques. This paper also demonstrates how the adaptation process can also ‘forget’ previous observations.

In the previous studies of online boosting that [5, 49], it has not been guaranteed that, interchanging a fixed number of weak classifiers will be able to identify and capture the change in pattern induced by new samples. Therefore, the number of weak classifiers to be replaced is an external parameter to these methods and the choice of such parameter is still unresolved. Furthermore, the time complexity of these methods will also increase with the increase in the number of base learners to be replaced. The proposed method, instead of replacement, keeps modifying the form of as many hypotheses as necessary to adapt to any new trend in the dataset. Therefore, the proposed method is capable of identifying the object with substantial change in the appearance during tracking. We show results where our algorithm supersedes previous method of tracking for tracking such objects in the results section. Our method also produces comparable performance w.r.t. its offline counterpart in classifying the UCI data.

The paper is organized as follows. We start with describing offline and online boosting
algorithms in sections A.2.1 and A.2.2 respectively. Sections A.3 and A.4 illustrates the least
square fitting of linear regressors, how to modify them to cope with new data and how to
incorporate these linear functions into online boosting. With a brief description of how to
apply online boosting for object tracking in section A.5, section A.6 analyzes the performance
of the proposed algorithm. Finally, we summarize our findings in Section A.7.

A.2 Background and Related Work

A.2.1 Offline Boosting

Boosting was proposed as a classification algorithm in [37]. Any input \( x \in \mathbb{R}^L \) is categorized
as one of the classes 1 or -1 using the sign of the function \( H(x) \). The function \( H : \mathbb{R}^L \rightarrow \{-1, 1\} \), also known as the strong classifier, is a linear combination of several other functions
\( f^k(x), k = 1, 2, \ldots K \).

\[
H(x) = \text{sign} \left[ \sum_{k=1}^{K} c^k f^k(x) \right]. \tag{A.1}
\]

The functions \( f(x) : \mathbb{R}^L \rightarrow \{-1, 1\} \), known as the base learners or weak classifiers in the
boosting literature, are also classifier functions except (as their name implies) they do not pos-
sess a high rate of accuracy. It has been proved in [37] that, even if the individual performances
of the weak learners are barely satisfactory (error rate > 50%), their combination could be
highly effective in terms of error.

The AdaBoost classifier is trained in an iterative fashion on the whole dataset \( \tilde{X} \in \mathbb{R}^{N \times (L)} \)
and their corresponding labels \( \tilde{y} \in \{-1, 1\}^N \). First, each of the samples are imposed a
uniform boosting weight \( w_i^0 = \frac{1}{N} \) and \( w^0 = [w_i^0]_{i=1}^N \) such that \( \sum_i w_i = 1 \). Then, at \( k \)-th step the the algorithm searches the \( f(x) \) producing the lowest expected error w.r.t. the
boosting weights \( w \). The mixing weight for the linear combination given in Eqn A.1 is cal-
culated by \( c^k = \frac{1}{2} \log \frac{1-e^k}{e^k} \). In the next iteration of boosting, the weights are modified by
\( w_i^{k+1} = \frac{w_i^k \exp(c^k y_i f^k(x_i))}{Z^k} \) so that the examples missed by \( f^k(x) \) receives a higher weight. The
weights are then normalized by \( Z^k \) to maintain \( w \) as a probability vector. In the following
description, we will omit the the superscripts of \( w^k \) and \( f(x) \) unless when they seem essential
for precise description.
A.2.2 Online boosting

The underlying idea for development of online boosting classifier is to learn incrementally. We wish to build the classifier in an environment where the samples arrive one after another as opposed to batch learning, where the whole dataset is available to learn from. The work by Oza et. al. [94] models the sequential arrival of samples by a Poisson distribution. Each of the weak classifiers of the pool is learned and updated on each sample \( k \) times in a row where \( k \) is a random number generated by \( \text{Poisson}(\lambda) \). If any example is misclassified by a base learner, the Poisson parameter \( \lambda \) increases. Therefore, the next base learner will concentrate more on learning the misclassified sample due to a large value of \( k \). The value of Poisson parameter \( \lambda \) is also accumulated to calculate the mixing weights of the hypotheses (e.g. \( c_k \) in Eqn A.1).

However, Oza et.al. did not discuss much about how to update these weak hypotheses. Grabner et. al. [49] introduced online boosting algorithm to the vision community and showed results on different problems. To update the weak hypotheses, the authors of [49] proposes to incrementally model the sample distribution with the help of a Kalman filter [127]. Their implementation also replaces a set of weak classifiers with a new one. Replacing a set of base learners with a new set has also been proposed by Avidan [5]. Avidan’s work was primarily concentrated on tracking using a boosting method that adapts to the change in scenes by adding new members and removing the old ones from the weak classifier pool. In this work, we only update the linear weak classifiers (learned in a Least Square method) to cope with the new data samples. The following subsections describe the update procedure of weak classifiers and how they are incorporated in boosting framework.

A.3 Adaptive Linear Weak Classifier

A.3.1 Weighted Linear Regressor

Let us suppose \( X \in \mathbb{R}^{N \times (L+1)} \) is the data matrix where there are \( N \) observations \( \mathbf{x} \in \mathbb{R}^{L} \) (the last column of \( X \) is a vector of all ones that is used for calculating the intercept). The corresponding labels for these examples are stored in \( \mathbf{y} \in \{-1, 1\}^{N} \). To solve a linear relation \( X\hat{\beta} = \mathbf{y} \) by least squares method, we have to minimize the error function \( (\mathbf{y} - X\hat{\beta})^{T}(\mathbf{y} - X\hat{\beta}) \). If the different samples have different importance weights quantified by the vector \( \mathbf{w} \), as they
do in AdaBoost, we have to minimize the error function \((y - X\beta)^TW(y - X\beta)\) where \(W\) is a diagonal matrix with \(w\) on its diagonal. The linear coefficients \(\beta\) that minimizes the error is given by the following expression.

\[
\beta = (X^TWX)^{-1}X^Ty.
\] (A.2)

Denoting the quantities \(X^TWX, X^TY\) as \(P \in \mathbb{R}^{L+1 \times L+1}\) and \(s \in \mathbb{R}^{L+1}\) respectively, the expression can be abbreviated as \(\beta = P^{-1}s\). The base learners we used in this study are linear classifiers \(f(x) : \mathbb{R}^L \rightarrow \{-1, 1\}\). The response of \(f(x)\) is calculated as follows:

\[
f(x) = \begin{cases} 
1, & \text{if } [x^T \ 1] \beta > 0; \\
-1, & \text{otherwise.}
\end{cases} \] (A.3)

### A.3.2 Adaptive Linear Regressor

As we mentioned earlier, our weak learners recalculate their parameter values for each new data subset. Let \(X_\tau\) denote the examples seen so far up to frame \(t\) and \(\beta_\tau\) denote the parameters (linear coefficients) of the regressor we learned from \(X_\tau\). Then, for a new subset \(X_\nu\), the new value of the linear coefficients \(\beta_{\tau+1}\) should be learned on \(X_{\tau+1} = [X_\tau^T \ X_\nu^T]^T\) and \(y_{\tau+1} = [y_\tau^T \ y_\nu^T]^T\) by the linear regression.

\[
\beta_{\tau+1} = P_{\tau+1}^{-1}s_{\tau+1}.
\] (A.4)

Here, \(P_{\tau+1} = X_{\tau+1}^T W_{\tau+1} X_{\tau+1}\), \(s_{\tau+1} = X_{\tau+1}^T W_{\tau+1} y_{\tau+1}\), \(W_{\nu}\) is a diagonal matrix having the boosting weights on the new samples \(w_\nu\) on its diagonal. It can be easily verified that, since \(X_{\tau+1} = [X_\tau^T \ X_\nu^T]^T\) and \(y_{\tau+1} = [y_\tau^T \ y_\nu^T]^T\), the two quantities required for computing \(\beta_{\tau+1}\) can be decomposed and expressed as a recursive summation of previous and new samples:

\[
P_{\tau+1} = X_{\tau}^T W_{\tau} X_{\tau} + X_{\nu}^T W_{\nu} X_{\nu} = P_{\tau} + P_{\nu} \\
s_{\tau+1} = X_{\tau}^T W_{\tau} y_{\tau} + X_{\nu}^T W_{\nu} y_{\nu} = s_{\tau} + s_{\nu}.
\] (A.5)

We are describing in the next section how to calculate \(P_{\tau+1}\) and \(s_{\tau+1}\) without storing all the previous examples seen so far.
### A.3.3 Weak Classifier Memory

Based on the decomposition provided in the previous section, to update the parameter $\beta_{\tau+1}$, we only need to store a matrix and a vector of sizes $L + 1 \times L + 1$ and $L + 1$ respectively and compute the two quantities $P_\nu$, $s_\nu$ only for the new samples. The simplistic form of Eqn A.5 suggests that we can also discard the quantities (i.e. ‘forget’ them) relating to very old examples if we wish to restrict ourselves to only the recent changes. So after the algorithm received a specific number of datapoints, the update equation changes to the following:

\[
P_{\tau+1} = P_{\tau} + P_\nu - P_{\tau-\omega},
\]
\[
s_{\tau+1} = s_{\tau} + s_\nu - s_{\tau-\omega}.
\] (A.6)

The value of $\omega$ decides for how long do we wish to ‘remember’ the contribution of any datapoint to our method. Obviously, we need to store all the quantities $P_{\tau}, P_{\tau-1}, \cdots, P_{\tau-\omega}$ and $s_{\tau}, s_{\tau-1}, \cdots, s_{\tau-\omega}$ in a queue. Since the dimensionality $L$ of the data is usually much smaller than the number of samples $N_\nu$ in the new set, the space complexity for this queue is not high.

![Figure A.1: Adaptation of linear classifier. Top row: without forgetting, bottom row: with forgetting.](image)

To visualize how this adaptation scheme works on any linear classifier, we generated synthetic two-class dataset. As Figure A.1 shows, the blue dots and red circles represent positive and negative examples respectively. The black line is the classifier learned by the proposed method with uniform weights ($W = I$). In the first three image of top row, we are incrementally adding new samples to the dataset and learning the weak classifier according to the update...
equation A.5 without forgetting. In three images of the bottom row, we are also removing the oldest subset and training the linear classifier by the update equation A.6 with forget mechanism. It can be easily observed how the linear classifier is correctly following the changing pattern of the dataset.

### A.3.4 Temporal weighting

According to the update equation A.5, all the new samples have equal importance and hence have the same contribution towards the modification of $\beta$. Therefore, after several time steps, the weak learner will become biased to the recent samples and contributions of the first set of examples will be gradually lost. As a result, the resulting classifier will be unable to classify observations similar to the first ones. Therefore, in this paper, we scaled the quantities used in equation A.5 with respect to time.

This study uses a temporal weight $\rho_\tau$ (decreasing with time) on the quantities $P_\tau$ and $s_\tau$ so that the updated values of them becomes a weighted cumulative sum.

\[
P_{\tau+1} = P_\tau + \rho_\tau P_\nu \\
s_{\tau+1} = s_\tau + \rho_\tau s_\nu \tag{A.7}
\]

The values of $\rho_\tau$ are predefined and decreases with $\tau$. In our implementation we used $\rho_\tau = \exp(-\tau/\sigma)$ where $\sigma$ is a predefined constant. In case of the ‘forgetting’, $P_{\tau-\omega}$ and $s_{\tau-\omega}$ have to be multiplied by their respective temporal weights before being subtracted in Eqn A.6. But then, all the $P$ and $s$ in the queue need to be re-weighted so that $P_{\tau-\omega+1}$ receives the largest weight followed by that of $P_{\tau-\omega+2}$ and so on. Therefore, the new values of $P$ and $s$ becomes

\[
P_{\tau+1} = P_\tau - \rho_1 P_{\tau-\omega} + \sum_{l=1}^{\omega-1} \rho_l P_{\tau-\omega+l} + \rho_\omega P_\nu \\
s_{\tau+1} = s_\tau - \rho_1 s_{\tau-\omega} + \sum_{l=1}^{\omega-1} \rho_l s_{\tau-\omega+l} + \rho_\omega s_\nu \tag{A.8}
\]

The effect of temporal weights on online boosting will be revisited in section A.4.

### A.4 Combining adaptive linear regressors for online boosting

Similar to the previous studies [94, 49], we apply the Adaboost training algorithm (modified) on every new subset of data $X_\nu$. All samples in $X_\nu$ are assigned uniform weights. For each
new subset $X_\nu$ of data and their labels $y_\nu$, we calculate the updated value of $\beta_\tau^j$ (using Eqn A.7 or Eqn A.8 depending on whether or not we wish to forget) of $j$-th weak hypothesis $f^j(x)$ to determine which $f^j(x)$ can most suitably conform itself with the changes. This base learner is immediately included in the strong classifier and the boosting weights $w$ are updated accordingly. For rest of the hypothesis, $\beta_\tau$ is not updated to $\beta_{\tau+1}$ until they are found to be the one generating minimum error w.r.t $w_\nu$ and included in the strong classifier.

We also added some modifications to the original boosting method to incorporate the adaptive linear regressors as the weak learners for online boosting. The following paragraphs will describe these changes to Adaboost.

**Need-based inclusion of base learners:** We start the learning process with $K$ base learners. The initial data subset is used to train few of the $K$ available base classifiers until all samples are adequately learned. We claim that, when the sum of boosting weights $w_\nu$ of the new samples decreases below a threshold, the examples need not be learned by anymore hypothesis. So, we stop training new weak learners when total weight is below a specific value and the remaining of the base learners remain dormant in the pool of weak classifiers. The same strategy has been followed for consequent datasets $X_\nu$ and their labels $y_\nu$. At $k=1$, instead of updating the linear base classifiers, first we apply them on the samples in $X_\nu$ to determine if any of the present learners can already classify them accurately or not. The update equation is enforced only when the minimum classification error with the current set of learners is not zero and therefore, the importance weight is significantly large.

**Updating Boosting weights $w$:** Every example receives the weight $w_i^0 = 1, i = 1, 2, \ldots N$ initially in our algorithm. We update the weights of the weak classifiers based on the performance of the weak classifier chosen in the latest iteration. But, if we normalize the importance weights after a perfectly correct classification (all samples were classified accurately), $w_i, i = 1, 2, \ldots , N_\nu$ will retain their previous values and their sum will not fall below the specified threshold. Therefore, the proposed online boosting method does not normalize the importance weight after updating.

**Effect of temporal weighting:** It may appear to the reader that, since we are using a temporal weight decreasing with time, after several time step the new datapoints would not have any influence on the boosting algorithm. To comprehend why that does not happen, it is important
OnlineBoost
For each subset of new data $X_\nu$ and their labels $y_\nu$

1. Start with uniform distribution $w_\nu$ and $dontLearn = 1$.
2. for $k = 1, 2, \cdots K$
   (a) if $dontLearn = 0$ then WeakLearn($w_\nu$).
   (b) $err^j = \text{CalcResp}(f^j, X_\nu)$.
   (c) $f^k = f^{j^*}$ where $j^* = \arg\min_j err^j$.
   (d) $c^k = \frac{1}{2} \log \frac{1 - err^{j^*}}{err^{j^*}}$.
   (e) $\forall w_i \in w_\nu$ $w_i = w_i \ast \exp(-c^k y_i f^k(x_i))$.
   (f) if $w^T T \nu < w_{th}$ then $dontLearn = 1$.
   (g) if $err^k > 0.5$ ignore $X_\nu, y_\nu$ and return.

WeakLearn($w_\nu$)
1. Compute new quantities $P_{r+1}$ and $s_{r+1}$.
2. Learn $\beta_{r+1}$ according to Equation A.7 or Equation A.8.

CalcResp($f_j, X_\nu, w_\nu$) returns $err^j$

1. $\lambda_{corr}^j = \lambda_{corr}^j + \sum_i w_i \delta(f^j(x_i), y_i)$ and $\lambda_{miss}^j = \lambda_{miss}^j + \sum_i w_i (1 - \delta(f^j(x_i), y_i))$ where $x_i \in X_\nu, y_i \in y_\nu$ and $w_i \in w_\nu$.
2. $err^j = \frac{\lambda_{miss}^j}{\lambda_{corr}^j + \lambda_{miss}^j}$.
3. $c^j = \frac{1}{2} \log \frac{1 - err^j}{err^j}$.

Note: Here $1$ is a vector of all ones and $\delta(a, b) = 1$ when $a = b$ and $\delta(a, b) = 0$ otherwise.

Table A.1: Algorithm: Boosting with Linear Adaptive Classifier.

to understand an important fact that not every classifier has a chance to observe each of the samples. This is due to the fact that, we stop training weak learners whenever $w^T T \nu \ 1$ decreases below a specific threshold. Therefore, the new subset arriving at time $t = 10$ may not be the $\tau = 10$th subset that $f^j(x)$ (where $1 \leq j \leq K$) experienced. Recall that, we are decreasing the value of $\rho$ according to the value $\tau$ which denotes the number of subset the corresponding weak classifier has actually learned on.

So, for the subset that was fed to the online boosting algorithm at time $t$, it will be the $\tau_1$-th and $\tau_2$-th new subset for two hypothesis $f^{j_1}(x)$ and $f^{j_2}(x)$ respectively. Therefore, even if for some weak learner the temporal weight $\rho_{\tau_1}$ is very small, for other classifier $f^{j_2}x$, the weight $\rho_{\tau_2}$ will be substantially large. Nonetheless, at some point of time, when the value of $\tau$ is large for all the base learners, any new subset will not receive the necessary attention. This is exactly the time when we should start ‘forgetting’ to make room for new observations.

Calculating $c^k$: The linear coefficients $c^j$ combining the base learners are determined using
the overall performance, that is, the overall classification accuracy (or error) on the whole set of training examples. These quantities are cumulatively stored into $\lambda^j_{\text{corr}} (\lambda^j_{\text{miss}})$ where $\lambda^j_{\text{corr}}$ accumulates the summation of boosting weights of samples correctly (incorrectly) classified. For details on these quantities, please refer to [94]. We need to keep in mind that, when we are ‘forgetting’ a subset, their corresponding $\lambda^j_{\text{corr}}$ and $\lambda^j_{\text{miss}}$ also need be discarded. The complete algorithm for proposed method of boosting adaptive linear weak hypotheses is stated in Table A.1.

### A.5 Application to tracking

An online learning method can be readily applied for tracking objects in a video. Avidan used a modified version of AdaBoost[5] for tracking objects in consecutive images. We have closely followed the implementation of [5] to apply our online boosting algorithm for tracking. The target is identified in the first frame by the smallest rectangle $R_{\text{inner}}$ containing only the object itself. Then a larger rectangle $R_{\text{outer}}$ is selected around the inner rectangle $R_{\text{inner}}$ to mark the background pixels. All the pixels in $R_{\text{inner}}$ are considered as positive examples (i.e. $y_i = 1$) and all the pixels in rectangle $R_{\text{outer}}$ are considered as negative examples (i.e. $y_i = -1$) for learning. In the next frame, the boosting classifier is applied on all the pixels in $R_{\text{outer}}$ to generate the responses the strong classifier. A meanshift algorithm [18] is applied to determine the new location of our target on this response image (also called the confidence map). Once the meanshift algorithm converges, two new rectangles $R_{\text{inner}}$ and $R_{\text{outer}}$ are redrawn around the new location to label the pixels and the strong classifier is retrained using the new data subset and their labels.

### A.6 Experiments and Results

#### A.6.1 Synthetic data

The 2D synthetic data of $N = 440$ samples was generated from a mixture of Gaussians as shown in the top-left image of Figure A.2. The blue ‘*’ and red ‘+’ denote the positive and negative examples respectively. We passed a pair of positive and negative examples to the proposed online boosting algorithm at a time. Since we do not wish to forget any observations,
weak learner memory is not used. The learned classifier was tested on another test dataset generated from the same mixture of Gaussians.

Fig A.2 visualizes how the weak classifiers are being generated and modified according to the changes on these synthetic dataset. We are using only $K = 10$ base learners for this illustration. Each hypothesis correspond to a line in Fig A.2. Initially, with $t < 20$, the algorithm tends to generate new weak classifiers and add them to strong classifier to learn the new examples (top row of Fig A.2). Then, when there are no more hypothesis left unused, the base learners start adapting to the changes (as can be seen in 2nd row of Fig A.2). Once the algorithm received sufficient samples, the set of weak classifiers are stabilized and remains almost unchanged till we finish learning (last row). One can easily notice how the the converged forms of the classifier are separating the boundaries of two classes.

Figure A.2: Synthetic 2D data (top-left image) and gradual adaptation of weak classifier at times $t = 4, 23, 53, 202, 440$.

To compare the performance of the online boosting method, we generated the classification error on the same dataset produced by an off-line Adaboost algorithm with logistic function as the base learner. We used the AdaBoost implementation of machine learning software Weka [129]. The total number of weak classifier for online and offline boosting was the same. Although we expect any online learning algorithm to exhibit inferior performance to that of its
offline counterpart, the proposed algorithm works much better than the offline boosting for this synthetic dataset (as shown in the first row of Table A.2).

![Table A.2: Classification rates on synthetic and UCI data](image)

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Training set</th>
<th>Size of test set</th>
<th>Proposed % correct</th>
<th>Offline % correct</th>
</tr>
</thead>
<tbody>
<tr>
<td>synthetic</td>
<td>440 + 440</td>
<td>440 + 440</td>
<td>94.43</td>
<td>72.61</td>
</tr>
<tr>
<td>ionosphere</td>
<td>75 + 75</td>
<td>150 + 51</td>
<td>84.07</td>
<td>83.08</td>
</tr>
<tr>
<td>breast cancer</td>
<td>100 + 100</td>
<td>112 + 257</td>
<td>90.24</td>
<td>91.32</td>
</tr>
<tr>
<td>diabetes</td>
<td>130 + 130</td>
<td>138 + 370</td>
<td>63.18</td>
<td>72.63</td>
</tr>
<tr>
<td>spam</td>
<td>500 + 500</td>
<td>1313 + 2288</td>
<td>87.67</td>
<td>89.58</td>
</tr>
</tbody>
</table>

A.6.2 UCI datasets

We followed the same strategy we used for synthetic data on UCI [2] datasets ‘ionosphere’, ‘breast cancer’, ‘diabetes’ and ‘spam’. First, each of these datasets were separated into training and testing subsets. Then, training samples were supplied to the learning algorithm in pairs of a positive and a negative example and none of the observations were ‘forgotten’. The results, using both the proposed online boosting method and the offline AdaBoost implementation of Weka on several UCI datasets are given in Table A.2. Both online and offline boosting classifiers comprise $K = 30$ base learners for all dataset but ‘spam’. Since ‘spam’ dataset is considerably larger than others, the total number of base learners used to learn it was $K = 50$. The results suggest that we can have competitive performance by online boosting with adaptive linear weak classifiers for real datasets too.

A.6.3 Tracking examples

For tracking, we processed the images of video following the procedures used in Avidan’s work [5]. As described in section A.5, the samples within the inner rectangle $R_{inner}$ were regarded as positive examples and that within $R_{outer}$ were considered as negative examples. In all the following experiments, the features used to represent the pixels were only R, G, B values.

The total number of weak learners used in all the tracking examples is 8. We fixed the number of previous frames that we wish to ‘remember’ and discarded earlier observations from the data. The queue length for weak learner memory was $\omega = 15$, i.e., the learners
‘forgets’ everything happened before 15 frames. The temporal weights are calculated with \( \sigma = 3 \). The procedure for outlier detection of [5] was also followed in our implementation. In what follows in this section, we will show some image sequences where the proposed method outperforms the meanshift [18] and ensemble trackers [5] respectively.

**Comparison with meanshift:** Figures A.3 (a), (b) and (c) compare the performance of the proposed online boosting algorithm to a meanshift tracker. Our first dataset (as shown in Figure A.3) (a) contains images of a police car chase. At a certain stage of chasing, the car being chased (also the object which we are tracking) completely turns around and then tries to flee again in another direction. We show in Figure A.3(a) that even though meanshift tracker was able to track the car after the collision, if fails to follow the object due to an occlusion by roadside pole and trees. But, our online learning adapts itself to the changes very rapidly and tracks it correctly. Since the current implementation of our algorithm can not modify the target window according to rotation or scale changes in the object, the target window was not redrawn according to the new appearance of the object.

The second dataset were recorded by a moving camera. In Figure A.3(b), three vehicles with similar appearances cross each other in the opposite direction. While the meanshift tracker confuses the target truck with the other one, the proposed learning algorithm remains capable of distinguishing the target. The last comparison (Figure A.3(c)) manifests how our algorithm adapts with illumination change (notice Frames 38 and 199) whereas meanshift tracker cannot.

**Comparison with ensemble tracker:** The ensemble tracking method [5] cope with the change in scene by replacing a set of base learners with new ones. The number of weak learners to exchange is an external parameter to the algorithm. We can not expect that replacing any fixed number of weak classifiers can always capture the change with time. This is exactly what happens when ensemble tracker looses the target object in Figures A.4 (a) and (b). In the video of cars on a city street at night (Figure A.4 (a)), the ensemble tracker gets distracted by the rotating red light of the police car and eventually ends up on another car facing in the reverse direction. Since we are modifying all the base learners (if necessary), the online boosting method tracks the object accurately. In Figure A.4(b), the tracker confuses the tracked person wearing a red jacket with another person wearing similar attire. The output of the proposed learning method, as shown in the bottom row(s), clearly exhibits the robustness of method for
both identification and capturing the changes.

Table A.3 displays number of frames of the aforementioned datasets correctly tracked by the proposed method and other methods. The number of frames were calculated manually from the output images. If in any frame, 25% of the target window does not contain the object (approximately, except CarChase sequence), we classify the frame as being incorrectly tracked. As we can see, in all the videos where the traditional methods fail, our method can track the target for almost the full length of the sequence.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Object</th>
<th>meanshift</th>
<th>ensemble</th>
<th>Proposed</th>
</tr>
</thead>
<tbody>
<tr>
<td>VehiclesCrossing</td>
<td>car</td>
<td>260/395</td>
<td>-</td>
<td>390/395</td>
</tr>
<tr>
<td>IlluminationChange</td>
<td>person</td>
<td>38/76</td>
<td>-</td>
<td>92/92</td>
</tr>
<tr>
<td>PoliceCarNight</td>
<td>car</td>
<td>-</td>
<td>40/290</td>
<td>280/290</td>
</tr>
<tr>
<td>PersonRedJacket</td>
<td>person</td>
<td>-</td>
<td>10/45</td>
<td>45/45</td>
</tr>
</tbody>
</table>

Table A.3: Frames exactly tracked by the proposed & other methods

A.7 Conclusion

This study proposes a new online boosting by continuous updating of weak classifiers. Results on artificial and real datasets shows the better performances achieved for both online learning and object tracking purposes by the proposed method than that of previous methods.

Acknowledgement: This research was partially funded by NSF CAREER award IIS-0546372 and Mitsubishi Electric Research Labs.
Figure A.3: Performance comparison (a), (b) and (c) with meanshift tracker. For each output sequence, top row: meanshift tracker, bottom row: proposed method.
Figure A.4: Performance comparison (a) and (b) with ensemble tracker. For each output sequence, top row: ensemble tracker, bottom row: proposed method.
Appendix B

Unsupervised Learning of Boosted Tree Classifier using Graph Cuts for Hand Pose Recognition

Abstract

This study proposes an unsupervised learning approach for the task of hand pose recognition. Considering the large variation in hand poses, classification using a decision tree seems highly suitable for this purpose. Various research works have used boosted decision trees and have shown encouraging results for pose recognition. This work also employs a boosted classifier tree learned in an unsupervised manner for hand pose recognition. We use a recursive two way spectral clustering method, namely the Normalized Cut method (NCut), to generate the decision tree. A binary boosting classifier is then learned at each node of the tree generated by the clustering algorithm. Since the output of the clustering algorithm may contain outliers in practice, the variant of boosting algorithm applied at each node is the Soft Margin version of AdaBoost, which was developed to maximize the classifier margin in a noisy environment. We propose a novel approach to learn the weak classifiers of the boosting process using the partitioning vector given by the NCut algorithm. The algorithm applies a linear regression of feature responses with the partitioning vector and utilizes the sample weights used in boosting to learn the weak hypotheses. Initial result shows satisfactory performances in recognizing complex hand poses with large variations in background and illumination. This framework of tree classifier can also be applied to general multi-class object recognition.

B.1 Introduction

Hand pose recognition is a non-trivial task with potential application in many areas such as human computer interaction, sign language interpretation etc. Hand pose recognition can be
formulated as an instance of the general multi class object recognition problem which is a very important goal in computer vision. This paper proposes an unsupervised learning framework for learning a boosted classifier tree for multi class object recognition (for this particular study, hand pose). The framework facilitates feature selection and feature sharing among large sets of features through learning boosted classifiers directly from soft labels obtained through spectral clustering.

We use spectral clustering, namely Normalized Cut [118] to obtain a top-down two-way hierarchical partitioning of the data. At each level of the tree generated by the hierarchical partitioning, we can train a classifier to discriminate the two sub-classes. However, we show that as a result of the spectral clustering, we can directly obtain weak classifiers which fit the obtained spectral partition. Such weak learner can then be boosted to form a classifier. As we can speculate, either or both of the subsets generated by the clustering algorithm may contain impurities. Therefore, we need to apply a boosting algorithm that has been shown to work well in the presence of noise, e.g. [102]. Ideally, the leaves of this classifier tree should comprise only the images of the same object class (in this study, same hand pose). At recognition time, an appropriate distance measure is used to find the representative image for the corresponding object class (the hand pose) at the leaves of this tree. The approach has been tried for hand pose recognition on a training set of images containing large variability in background, illumination and for some cases, the exact shape of the hand. Our algorithm were learned on a dataset consisting of five hand poses (three of them were used in [119]).

An overview of the whole approach is described in section B.2. The process of building the tree classifier is illustrated in section B.3 where sections B.3.1, B.3.2, and B.3.3 describe the clustering method, weak classifier learning and the soft margin AdaBoost respectively. Section C.5 explains the experimental setup and results obtained. Finally, section B.5 discusses the findings and future works on this study.

B.1.1 Related Work

There have been extensive research recently on recovering hand pose from an image. [13, 72, 71]. Hand pose recognition task was formulated as a database indexing problem for a large database of articulated hand images in [3, 4]. Since it may not always be possible to
label the training images perfectly due to high variability of gestures, Wu et.al. [133] uses a dataset of both unlabeled and labeled data for view independent gesture estimation. A template (generated by both edge and color features) matching technique has also been employed for hand pose detection in [119] that uses a dataset containing large variation in background and illumination. A linear classifier learned with oriented edges and marginalized templates has been shown to produce very good results for three hand poses. A detection tree classifier were used to prune out a large portion of the image, which is less probable to contain a hand, from the search. Since hand appearances in the real world images can vary largely in shape and color for the same hand pose class, a template based approach is not sufficient for pose estimation. In [93], a two-layer boosted classifier tree has been used to detect the hand in the image and recognize its pose. The top layer of the tree detects hand in an image window and the cascades of classifiers below recognize the hand pose. High success rates have been reported by the authors on hand poses with the same skin color of hands, background, and illumination which seem to underestimate the actual scenario.

Our work, however, does not deal with finding out a hand in any given query image. We are interested in discovering the pose provided an image of only the hand itself. We used an extended dataset of that was used in [119] for its large variability in background, illumination, and the exact shape of the hand for the same hand pose because we are interested in this study to recognize pose in realistic cluttered environment under varying conditions.

B.2 Overview of the approach

We build a decision tree by applying a top-down hierarchical clustering algorithm to the image dataset such that each leaf is expected to contain images from only one object class. Suppose any node of this tree contains image set $I$ and it’s children hold the subsets $I_A$ and $I_B$. We will learn a binary classifier at each such node so that any query image $x^q$ will be classified as a member of either of these two subsets. This query image will be classified by another classifier in the following level and this process will continue until $x^q$ reaches one of the leaves. A similarity measure $d()$ is used at the leaf to find the image closest to $x^q$ which represents the object class of it.

Suppose we have $N$ images $x_k$, $k = 1, 2, \ldots, N$ for training and they are stacked one on
top of the other in the matrix $X \in \mathbb{R}^{N \times L}$. We first apply a suitable feature $\phi$ on the images to transform the data matrix from input space to the feature space; i.e., from $X \in \mathbb{R}^{N \times L}$ to $X_\phi \in \mathbb{R}^{N \times L_\phi}$. Since the dimensionality $L_\phi$ of the feature space is usually too large for the clustering algorithm to generate meaningful clusters, we need to further apply a dimensionality reduction tool. Principal Component Analysis (PCA) is used to reduce the dimensionality of the transformed data matrix. Let us denote the data matrix in the space spanned by the principal components of $X_\phi$ as $Z_{PCA} \in \mathbb{R}^{N \times L_{PCA}}$.

We execute Normalized Cut (NCut) method to cluster the data $Z_{PCA} = [z_1^T \ z_2^T \ \cdots \ z_N^T]^T$ in the PCA space. Given the distance matrix $W \in \mathbb{R}^{N \times N}$ of the PCA space representation of images $z_k$, $k = 1, 2, \cdots, N$, the NCut method will partition the set into two disjoint subsets. The clustering algorithm is applied recursively to generate the tree structure.

Since the clustering algorithm may not be able to separate the groups of images of same pose perfectly, there may be some impurities in either or both $I_A$ and $I_B$. Therefore, we chose a variant of the AdaBoost [38] algorithm, namely the Soft margin AdaBoost [102] which maximizes the lower bound of the margin of the strong classifier in presence of noise, to be employed at each node. Our choice of the learning algorithm will be justified both theoretically and practically in latter sections. The classifier was trained using the feature responses $\tilde{\psi}_k = \psi(x_k)$ of a different set of features $\psi$. We have used a novel method for learning of the weak hypotheses from these responses $\tilde{\psi}_k$, using the clustering information produced by the NCut method and the probability distribution imposed on the training examples at each round of the boosting process.

**B.3 Unsupervised Learning of Tree Classifier**

**B.3.1 Normalized Cut Clustering**

Consider each point $z_k$, $k = 1, 2, \cdots, N$ in the PCA space as nodes of some graph $G = (V, E)$ where there is an edge from each node to all other nodes in $V$. Let $W$ be the weighted adjacency matrix for the edges in $E$. We calculated the distance between any two points $z_{k_1}$ and $z_{k_2}$, or equivalently, the weight for the edge between $z_{k_1}$ and $z_{k_2}$, using multivariate gaussian distance in the PCA space.

$$[W]_{k_1,k_2} = w(k_1,k_2) = \exp\left[ -\frac{1}{2}(z_{k_1} - z_{k_2})^T \Sigma^{-1}(z_{k_1} - z_{k_2}) \right]. \quad \text{(B.1)}$$
The bandwidth (or covariance) matrix \( \Sigma \) for the gaussian measure was determined by the local scaling approach proposed in [138].

Given the weight matrix \( W \), the Normalized Cut algorithm [118] will generate the optimum partitioning vector \( r \) to divide \( V \) (\( I \) in our case) into two disjoint subsets \( A \) and \( B \) (\( I_A \) and \( I_B \) respectively). The partitioning vector \( r \) is almost always real valued whereas we need a binary indicator vector \( b \) that assigns \( z_k \) to the appropriate cluster. We follow the approach described in [118] for searching the entry \( \delta \) in \( r \) that produces the minimum association between two clusters, i.e. the minimum NCut value. The two subsets \( I_A \) and \( I_B \) of \( I \) are obtained by checking the values of the eigenvector entries against the threshold \( \delta \).

\[
I_A = \{ x_k | b_k^+ = 1 \} \quad \text{and} \quad I_B = \{ x_k | b_k^- = -1 \}
\]

(B.2)

where \( b_k = \text{sign}(r_k - \delta) \) and the symbols \( b_k \) and \( r_k \) denotes the entries in the vectors \( b \) and \( r \) respectively for \( k = 1, 2, \cdots, N \). The resulting subsets are recursively partitioned into smaller sets in the same way until the algorithm satisfies some terminating condition.

The quantitative measure of association between two clusters, namely the NCut value, and the cluster size are used as a stopping criterion. One of the reasons to favor Normalized Cut method over other clustering methods is the fact that it makes no assumptions about the number of clusters and the distribution of points pertaining to it. The only external parameter we need to set, namely the \( NC_{th} \) value is intuitive and less strict for an assumption. Furthermore, as we will see in the next section, the partitioning vector \( r \) can be used to learn the binary classifier at each node of the classifier tree.

### B.3.2 One-pass Learning of weak classifiers

We will learn a binary boosting classifier [38] at each node of the tree generated by the NCut method. We propose a fast and elegant method for learning the weak classifiers at each stage of boosting by utilizing the information provided by the clustering algorithm. Using our notation, \( r \) (the second eigenvector of the standard eigensystem defined in [118]) stores the partitioning information for subsets \( I_A \) and \( I_B \). As we know [87], in ideal cases, \( r \) is a piecewise linear vector characterizing the correspondence between the points to the clusters to be discovered. Therefore, when classifying between the two subsets produced by the NCut algorithm, the vector \( r \) can be considered to contain the ‘soft labels’ (we will be using this term for \( r \) in rest of
the paper) for the classifying task. The binary vector $b$ (as defined in equation B.2) generated by thresholding $r$ against $\delta$ works as the actual labels for classification.

For the purpose of learning the binary classifiers, a different set of features $\psi^j, j = 1, 2, \cdots, L_\psi$ are applied on the images. Let us denote the $N$ dimensional vector for $j$-th feature responses of all the images $x_k, k = 1, 2, \cdots, N$ as $\tilde{\psi}^j = [\tilde{\psi}^j_1, \tilde{\psi}^j_2, \cdots, \tilde{\psi}^j_N]^T$ where $1 \leq j \leq L_\psi$. At each round $t$ of boosting process, we need to find a weak classifier making the least classification error in terms of the labels $b$ weighted by the probability distribution $\pi^t_k, k = 1, 2, \cdots, N$ imposed on the training samples at $t$-th round. We learn a binary weak classifier $h^j \in \{+1, -1\}^N$ for each feature $\psi^j$. The weak hypothesis $h^j$ assigns an image $x_k$ to subset $I_A$ (class $+1$) if the corresponding feature response $\tilde{\psi}^j_k$ is greater than some threshold $\theta^j$ and assigns $x_k$ to subset $I_B$ (class $-1$) otherwise.

$$h^j(x_k) = \begin{cases} 
+1 & \text{if } \tilde{\psi}^j_k > \theta^j \\
-1 & \text{if } \tilde{\psi}^j_k \leq \theta^j 
\end{cases} \quad (B.3)$$

The most discriminative feature among all the $\psi^j, j = 1, 2, \cdots, L_\psi$ will have distinct densities of responses for samples of two classes and will have a high correspondence between its entries and that of the label $b$. Since both soft labels $r$ and feature responses $\tilde{\psi}^j$ are real numbers, we expect the relationship reflected be more strongly between them. We model this correspondence by a linear relationship between $r$ and $\tilde{\psi}^j$ as follows:

$$r = \alpha^j \tilde{\psi}^j + \beta^j \quad (B.4)$$

where $\alpha^j$ and $\beta^j$ are two scalar coefficients. This assumption of linear dependence gives us a straightforward way to determine the threshold $\theta^j$ by $\theta^j = \frac{\delta - \beta^j}{\alpha^j}$.

The linear model between the soft labels and the feature responses thus eliminates the necessity for an exhaustive search (i.e. to check sufficiently large number of values in the range of the feature values) to compute $\theta^j$. The direction of the inequality signs in the definition of $h^j$ in equation B.3 may be reversed with the sign of $\alpha^j$ as shown in figure B.1.

Finally, the weak hypothesis $h^j_t$ to be selected at round $t$ is the one making the least weighted error against the actual labels $b$.

$$h^j_t = \arg \min_j \epsilon^j_t \quad \text{where} \quad \epsilon^j_t = \sum_{k=1}^{N} \pi^t_k I[b_k \neq h^j(x_k)]. \quad (B.5)$$

Here $I$ is the indicator function that outputs 1 if the expression within the square braces is true and outputs 0 otherwise.
We can use Least Square method to compute the linear coefficients $\alpha^j$ and $\beta^j$ in equation B.4. Let us define matrix $\tilde{\Psi}^j = [\tilde{\psi}^j_1]$ and vector $a^j = [\alpha^j \beta^j]^T$ to write the linear dependence in equation B.4 in the matrix form as $r = \tilde{\Psi}^j a^j$. Recall that at each round $t$ of boosting, we emphasize on learning a different subset of the training examples according to the probability measure $\pi^t_k$. Therefore, we need to use a weighted least square approximation to calculate the coefficients of the features that are biased towards the samples by higher $\pi^t_k$ than that of others. We need to find out the coefficients $a^j$ minimizes the weighted squared error

$$
\epsilon_{rgs} = (\tilde{\Psi}^j a^j - r)^T \Pi^t (\tilde{\Psi}^j a^j - r)
$$

(B.6)

where $\Pi^t = diag(\pi^t_1, \pi^t_2, \ldots, \pi^t_N)$. The coefficients $a^j$ that minimizes $\epsilon_{rgs}$ can be derived as (see [54] for reference)

$$
a^j = ((\tilde{\Psi}^j)^T \Pi^t \tilde{\Psi}^j)^{-1}(\tilde{\Psi}^j)^T \Pi^t r.
$$

(B.7)

**B.3.3 Soft Margin AdaBoost**

The weak hypotheses learned from the feature responses are combined to form a strong classifier. A large margin classifier should be preferred to ensure the robustness of the tree classifier. The authors of [111] have demonstrated that, the margin of the final classifier increases with the number of weak classifiers added to the final classifier of a boosting process and results in a continuous reduction in generalization error. There are two issues worth discussing at this point. First, if at any stage of the regular AdaBoost [38], the weighted error made by the weak classifier becomes zero, we need to abort the iterative process of adding new classifiers. The
second concern is, in noisy environment, adding more weak hypothesis will concentrate on learning the noisy samples rather than increasing the margin [102].

Rätsch et.al. [102], proposed a modification to the objective functional that AdaBoost-type algorithms minimize. Let us denote $H(x_k) = \sum_{t=1}^{T} c_t h_t(x_k)$ as the linear combination of $T$ weak hypotheses $h_t(x_k) \in \{-1, +1\}$, $t = 1, 2, \cdots, T$ according to the (non-normalized) coefficients $c = [c_1, c_2, \cdots, c_T]^T$. The authors defined the ‘hard’ margin for each sample $x_k$ as $\rho_k(c) = b_k \sum_{t=1}^{T} c_t h_t(x_k)$ (where $b_k$ is the label for $x_k$) and introduced a ‘soft’ margin $\tilde{\rho}_k(c) = \rho_k(c) + B(\mu_k)^p$ (where $\mu_k = \sum_{r=1}^{t} c_r \pi^r_k$) to induce tolerance to noise in the boosting process.

Replacing $\rho_k(c)$ by $\tilde{\rho}_k(c)$ in the objective function that AdaBoost tries to minimize, the iterative update formulae for $c$ and $\pi^t_k$ become:

$$c^{(t)} = [c^{(t-1)} c^t]^T \text{ with } c^t = \arg \min_{c^t \geq 0} \sum_{k=1}^{N} \exp \left\{ -\frac{1}{2} \left[ \rho_k(c^{(t)}) + B |c^{(t)}| \mu^p_k \right] \right\};$$

$$\pi^t_k = \frac{1}{Z_t} \exp \left\{ -\frac{1}{2} \left[ \rho_k(c^{(t)}) + B |c^{(t)}| \mu^p_k \right] \right\}.$$

(B.8)

There are two external parameters, namely $B$ and $p$, need to be tuned. The superscript $t$ for each quantity denotes the value of that specific variable at boosting stage $t$. The denominator $Z_t$ in the definition of $\pi^t_k$ is a normalizing constant to convert $\pi^t_k, k = 1, 2, \cdots, N$ into a probability distribution. As can be observed, these update rules allow to add as many weak hypothesis as we want even if the corresponding weighted error vanishes. The final form of the strong classifier is a sign function of the value of $H$ with normalized linear coefficients : $f(x_k) = \text{sign} \left[ \frac{H(x_k)}{\sum_{t=1}^{T} c^t} \right]$. We refer the reader to [102] for a detailed explanation of Soft Margin AdaBoost.

B.4 Experiment and Results

For our experiment, we used the datasets of [119] with two more types of hand pose images. These images were scaled to the size $64 \times 64$ pixels and converted to grayscale. The image set contains 5 hand poses, namely hand open to side, pointing to side, fist to the side, pointing towards the camera, and fist upwards. The background, illumination, exact hand shape and clothing of the images vary largely in the data, as can be seen from some example images in figure B.2 (a). We have randomly picked 343 images for training and 375 images (75 per pose)
for testing purposes. The sizes of image sets for different hand poses were not the same in the training set, but each set contains at least 50 images. Each pose of the test set contains 75 images. The task is to find out the hand pose of each of these 375 test images when presented to the learning algorithm.

Figure B.2: Images from training set and their feature transforms. (a) Sample images from training set. (b) Feature transform for clustering. From left to right, the training image, original edge image, cleared edge image, distance transform w.r.t. the cleared edge image

**Feature used for Clustering:** For clustering purposes, we have used the Distance transform (DT) [44] images as the feature $\phi$ (section B.2). To eliminate the interference of the background, the edge images that were used to calculate the DT of an image were manually cleaned to remove the edges in the background. Some of the images $x_k$, $k = 61, 156$ and their feature transforms $\phi(x_k)$ have been shown in figure B.2 (b).

**Building decision tree by clustering:** The PCA transform of the DT images in $X_\phi$ generates the $L_{PCA} < L_\phi$ dimensional data matrix $Z_{PCA}$ retaining 99% of the variance. To apply NCut clustering to $Z_{PCA}$, we need to calculate the distance matrix $W$. We consider distances of $\eta$ nearest neighbors of $z_k$ (according to the Euclidean distance in the PCA space) while computing the bandwidth matrix $\Sigma$ used to calculate $W$ in equation B.1 (see [138] for details). As we can infer, the height of the tree can be minimized if the clustering algorithm divides the image set $I$ into almost equal sized subsets $I_A$ and $I_B$. Therefore, we examined multiple clustering results for the weight matrix $W$ computed in $l_{PCA}$ (where $4 \leq l_{PCA} \leq L_{PCA}$) dimensional subspaces of the PCA space and with $\eta \in \{5, 10, 20\}$. We calculated the ratio
of sizes $I_A$ and $I_B$ discovered by NCut algorithm for all these $l_{PCA}$ and $\eta$ values. The combination of the values of $l_{PCA}$ and $\eta$ that produced the largest ratio $\frac{|I_A|}{|I_B|}$ was used for final clustering. The terminating NCut value was $NC_{th} = 0.375$ and the minimum size of cluster to be partitioned further was taken as 10. Some of the subsets at the leaves of the tree generated by recursively calling NCut algorithm are shown in figure B.3.

Figure B.3: Sample clusters generated by the NCut algorithm

We have found in the experiment that, NCut is actually separating the training set very well at the internal nodes. For example, at the root, the two subsets created by the clustering contain the poses \{open side, fist side, point side\} (the first three poses in figure B.2 (a) from left) and \{point toward camera, fist up\} respectively with two noise images. The former subset is further divided into sets containing hand poses \{open side\} with one noise image and \{point side, fist side\} (the second and third pose from left in figure B.2 (a)).

Features for classification: After generating the tree, we need to apply a binary classifier at each node. We can not use DT features for classification since the edges from background clutter (which were not present in training images) will distract the classifier. Therefore, the features we used for classification are oriented gaussians and their first and second partial derivatives at two bandwidths, namely \{8, 5\} pixels. They were generated for eight orientations ranging from 0 to $2\pi$ with an interval of $\frac{\pi}{4}$. For the elliptical gaussian filters, the bandwidths in $y$-direction are half of that used in $x$-direction. The soft margin AdaBoost [102] classifier is then learned at each internal node of the tree. The external parameters in equation B.8 were set to $B = 100$ and $p = 2$.

Pose recognition at leaf: When a query image $x^q$ reaches the leaf, we searched over the subset $\{x_k\}_{k=1}^{N_{leaf}}$ residing at the leaf based on a variant of Chamfer distance to find out the image $x_{closest}$ closest to $x^q$. The hand pose of $x_{closest}$ is assigned as the class of $x^q$. The cleaned edge images (as shown in figure B.2) of these $N_{leaf}$ images are utilized to calculate the distance.
The DT image of \( x^q \), calculated with respect to its own edge features, is compared to the clean edge images of the leaf. We then build \( N_{leaf} \) histograms, each comprising 10 bins, of the pixels \( x^q(i), i = 1, 2, \ldots, L \) according to the distance \( x^q(i) \) has to its closest edge feature in the cleaned edge image of \( x_k \) at the leaf. Out of these \( N_{leaf} \) histograms, we consider only frequencies of pixels in the first bin of the histogram. The leaf image having the largest number of pixels in the first bin is considered to be the best match for \( x^q \). This measure is more strict than Chamfer distance since it only considers the number of pixels which are nearest to the edge features (in the cleaned image) instead of taking into account the distance of all the pixels in \( x^q \).

### Table B.1: Recognition accuracies of the decision tree with different classifiers

<table>
<thead>
<tr>
<th>Pose</th>
<th>Fist side</th>
<th>Fist up</th>
<th>Point forward</th>
<th>Point side</th>
<th>Open side</th>
<th>Classifier</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fist side</td>
<td>0.96</td>
<td>0.01</td>
<td>0</td>
<td>0</td>
<td>0.03</td>
<td>softAB_w</td>
</tr>
<tr>
<td></td>
<td>0.93</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.67</td>
<td>softAB_exstv</td>
</tr>
<tr>
<td></td>
<td><strong>0.97</strong></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0.03</td>
<td>AB_w</td>
</tr>
<tr>
<td>Fist up</td>
<td>0.02</td>
<td>0.78</td>
<td>0.15</td>
<td>0.025</td>
<td>0.025</td>
<td>softAB_w</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td><strong>0.8267</strong></td>
<td>0.133</td>
<td>0.013</td>
<td>0.027</td>
<td>softAB_exstv</td>
</tr>
<tr>
<td></td>
<td>0.026</td>
<td>0.787</td>
<td>0.133</td>
<td>0.027</td>
<td>0.027</td>
<td>AB_w</td>
</tr>
<tr>
<td>Point forward</td>
<td>0</td>
<td>0.06</td>
<td><strong>0.94</strong></td>
<td>0</td>
<td>0</td>
<td>softAB_w</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0.12</td>
<td>0.88</td>
<td>0</td>
<td>0</td>
<td>softAB_exstv</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0.11</td>
<td>0.89</td>
<td>0</td>
<td>0</td>
<td>AB_w</td>
</tr>
<tr>
<td>Point side</td>
<td>0.06</td>
<td>0</td>
<td>0</td>
<td><strong>0.94</strong></td>
<td>0</td>
<td>softAB_w</td>
</tr>
<tr>
<td></td>
<td>0.08</td>
<td>0</td>
<td>0</td>
<td>0.89</td>
<td>0.03</td>
<td>softAB_exstv</td>
</tr>
<tr>
<td></td>
<td>0.11</td>
<td>0</td>
<td>0</td>
<td>0.88</td>
<td>0.01</td>
<td>AB_w</td>
</tr>
<tr>
<td>Open side</td>
<td>0</td>
<td>0</td>
<td>0.03</td>
<td>0.04</td>
<td><strong>0.93</strong></td>
<td>softAB_w</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0.027</td>
<td>0.03</td>
<td>0.053</td>
<td>0.89</td>
<td>softAB_exstv</td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0.05</td>
<td>0.06</td>
<td>0.89</td>
<td>AB_w</td>
</tr>
</tbody>
</table>

**Recognition results**: The results of hand pose recognition by soft margin AdaBoost classifier using the weak classifier learning (softAB_w) as described in section B.3.2 are summarized in table B.1 in the form of a confusion matrix. More precisely, the decision tree with softAB_w classifier correctly classifies 94% of the images of ‘Point forward’ (refer to the third row) but misclassifies 6% of them as ‘Fist up’. Notice that, the sidewise poses \{ open side, fist side, point side \} are generally confused among themselves. For the sake of comparison, we also generated the results using regular AdaBoost (AB_w) using the weak classifier learning and using soft margin AdaBoost without using the weak classifier learning (softAB_exstv). To get around with the scenario where the weighted error for the first weak hypothesis is zero in regular AdaBoost (i.e. \( \epsilon_1^j \) at the first stage of boosting), we applied a majority vote classifier by aggregating the weak hypotheses having zero error on the data subset at hand. As we observe,
softAB.w has the best performance among all the classifiers used in the study. The inferior classification result for the pose ‘fist up’ is due to the complexity in the data subset itself, which is why all the learning algorithm worked poorly for recognizing images from this set.

**Features selected by boosting algorithm:** In Figure B.4, we show part of the boosted classifier tree generated by our method. The images between the blue parentheses imply a cluster or subset of images. The red box, immediately below each cluster, shows the features selected by the boosting algorithm used at this level. In the top level, where the cluster has images with all poses, the features try to distinguish the hand from background. As the subsets are broken into smaller regions, the features try to learn characteristics specific to the poses that constitutes the cluster. Notice how the Gaussian derivative filters are aligned with fingers or other regions of the palm with similar appearances as themselves.
Analyzing margins for generalization: Since the clustering algorithm is performing very well on the training set, we do not expect the classifiers to experience highly noisy data to learn on. This is the reason why \( \text{AB}_w \) is also performing fairly well in this case. Nonetheless, the \( \text{softAB}_w \) algorithm produces significantly better results than that produced by \( \text{AB}_w \). The over-fitting of \( \text{AB}_w \) leads to an inferior generalization to the test set images. To justify the claim, let us consider the 4th row in table B.1 showing the results for the pose ‘point side’. We observe that more images from this class were confused by \( \text{AB}_w \) as pertaining to the pose ‘fist side’ than that were confused by \( \text{softAB}_w \). Let us examine the node where the algorithm driving the query image to a wrong direction, namely node 2 of the tree.

Figure B.5: Comparison of margin and base hypothesis. (a) Comparison between classifier margins of \( \text{AB}_w \) and \( \text{softAB}_w \). (b) Comparison between two approaches to learn the weak classifier in the first boosting step at node 2. Left, by linear regression and right, by exhaustive search

In figure B.5(a), we plot the classifier margins (label multiplied by the linear combination of the weak classifier outputs) against the class labels. We can easily sort out the noise image sitting at the middle of the x-axis of the plot with an image index of 76. The regular Adaboost assigns it a +1 label whereas the soft margin version of it assigns a -1. But observe that the regular Adaboost margin for this noise image is approximately 0.4 which is also the margin of many other images of class +1. The reason behind this is, regular Adaboost adds many
weak hypothesis just to learn this particular sample after a certain number of boosting steps (when the probability measure $\pi^t_{g}$ grows too high for this sample). Therefore, the importance of the other images are underestimated and eventually results in a low margin output. The soft margin version, on the other hand, does not emphasize on learning any particular example and concentrates on maximizing the overall margin of all examples in the training set. This characteristic is exhibited in figure B.5(a), where the average absolute value of the soft $\text{AB}_w$ margin is approximately 0.8 which is much larger than that of $\text{AB}_w$ which is approximately 0.5.

**Comparison of weak classifier learning approaches:** For the exhaustive search approach to learn the weak classifiers, we checked $Q \in \{10, 20, \cdots, 90\}$ quantile values of the feature responses of each feature to find out the threshold. Figure B.5(b) compares weak hypotheses learned this way with that learned using the proposed approach in section B.3.2 at node 2 (left child of root) of the decision tree in the first boosting round. Both of the approaches are selecting almost perfect weak classifiers, but the exhaustive search increases the time required for training by an order of 10. Increasing the number of values to check for a threshold did not produce significantly better results and slows the learning process even further. The reason why $\text{softAB}_\text{exstv}$ producing inferior recognition results is, since we are not relying on any model to determine the optimum threshold for the weak classifiers, it is not generalizing well to the test data.

The results from this work is not comparable to that of [119] because we recognize the pose when presented with an hand image rather than detecting hand and poses from an image. Furthermore, the authors in [119] have used only three types of hand poses, namely open side, point side, and fist side, for detection. Examining the whole dataset, we have found that images form these classes of hand poses have much stronger edge responses and the exact same shape of the hand throughout the subset of the images corresponding to the hand pose. These are all highly favorable characteristics for any classification method to be used for pose estimation. As a result, the best classifier in [119] was reported to have 99% accuracy with 5% false detection rate. Our dataset contains images of two more types of hand poses which are much more complicated than these images with regard to the edge responses and shape of the hand within each class. Nonetheless, our algorithm is producing very good results on these images.
B.5 Conclusion

This study proposes a boosted classification tree learned in an unsupervised fashion for selection and sharing of features in multi-class object recognition. The results of the implementation of the targeted dataset has shown very good results for hand pose estimation from images with random background, variation in illumination, clothing and the exact shape of the hand within each pose. Since we are not exploiting any characteristics of the hand itself for classification, this decision tree can be easily extended for any type of object recognition.

Acknowledgement: We would like to thank Dr. B. Stenger for providing us with the hand dataset used in this study.
Appendix C

A Framework for Feature Selection for Background Subtraction

Abstract

Background subtraction is a widely used paradigm to detect moving objects in video taken from a static camera and is used for various important applications such as video surveillance, human motion analysis, etc. Various statistical approaches have been proposed for modeling a given scene background. However, there is no theoretical framework for choosing which features to use to model different regions of the scene background. In this paper we introduce a novel framework for feature selection for background modeling and subtraction. A boosting algorithm, namely RealBoost, is used to choose the best combination of features at each pixel. Given the probability estimates from a pool of features calculated by Kernel Density Estimate (KDE) over a certain time period, the algorithm selects the most useful ones to discriminate foreground objects from the scene background. The results show that the proposed framework successfully selects appropriate features for different parts of the image.

C.1 Introduction

Background modeling and subtraction forms a core component in many vision systems. The main idea behind such a module is to automatically generate and maintain a representation of the background that is then used to classify any new observation as background or foreground. The information provided by such a module can then be utilized for performing high-level object analysis tasks such as object detection, tracking, classification and event analysis.

Various methods of increasing complexity have been considered in the literature. In [132].
A single Gaussian was considered to model the statistical distribution of the background at each pixel. Friedman et. al.[40] use a mixture of three Normal distributions to model the visual properties in traffic surveillance applications. Three hypothesis are considered - road, shadow and vehicles. The EM algorithm is used which gives very good model-fitting but is computationally expensive. Grimson et. al. [51] extend this idea by using multiple Gaussians to model the scene and develop a fast approximate method for updating the parameters of the model incrementally. Such an approach is capable of dealing with multiple hypothesis for the background and can be useful in scenes such as waving trees, beaches, escalators, rain or snow. The mixture-of-Gaussians method is quite popular and was to be the basis for a large number of related techniques [61, 53]. In[43], a statistical characterization of the error associated with this algorithm is studied. When the density function is more complex and cannot be modeled parametrically, a non-parametric approach able to handle arbitrary densities is more suitable. Such an approach was used in [28] where the use of Gaussian kernels for modeling the density at a particular pixel was proposed.

Another class of background modeling methods try to model the short-term dynamical characteristics of the input signal. Several authors [63, 68] have used a Kalman-filter based approach for modeling the dynamics of the state at a particular pixel. A simpler version of the Kalman filter called Weiner filter was considered in [124] that operates directly on the data. Such modeling may further be performed in an appropriately selected subspace [139, 92, 89].

Besides modeling the statistical distribution of the data, another factor in background modeling is the choice of the transformation that is applied to the original data in order to obtain the features that are used. Several features have been considered including raw color, normalized color, spatial gradients, texture and optical flow. The basic idea behind feature selection is that one wants to be invariant to certain types of changes in the visual space while maintaining a good detection for the foreground objects. For instance, in outdoor scenes, the classification must be invariant to a change of illumination that might occur due to the sun, clouds or light from a nearby light source. Similarly, in dynamic scenes such as ocean waves or waving trees, invariance to such periodic motion is critical. Each feature has its strength and weakness and is particularly applicable for handling a certain type of variation. For instance, normalized color, spatial gradients or texture features may be considered for obtaining invariance to
illumination[50, 61], while optical flow might be useful in handling dynamic scenes [88, 97]. On the other hand, such features may not be very suitable in other regions. For instance, spatial gradients or texture is not very suitable for a region that has low spatial gradients since such feature will be unable to detect any object that has a low gradient itself. Similarly, optical flow cannot be computed accurately in regions that have low texture and is thus not very useful in such regions.

In most background subtraction algorithms, the features are chosen arbitrarily and the same feature are used globally over the whole scene. No framework is known to the authors for selecting suitable features for different parts of the scene. The primary contribution of this paper is a generic formulation that is able to automatically select the features that obtain the best invariance to the background changes while maintaining a high detection rate for the foreground objects. We propose to address the problem as a classification problem where we classify the foreground objects from background pixels. An ensemble learning method, namely boosting classifier, seems appropriate in this scenario. Boosting algorithms usually generates a weighted linear combination of some weak classifiers that perform only a little better than random guess. We learn weak classifiers from the feature values at a pixel and combine the ones performing better than the others to produce a strong classifier. Thus we are effectively selecting different features at each pixel to distinguish foreground objects from the background. Once selected, we expect this combination of features to perform successfully afterwards, unless there are major changes in the scene.

The organization of the paper is as the following. Section C.2 gives a general overview of the framework. Sections C.3 and C.4 describes the general Realboost algorithm and how the it was used in background subtraction, respectively. Section C.5 explains the experimental setup and the results. Finally, we conclude by a brief discussion over the findings of this study and on directions for future work in section C.6.

**C.2 Feature Selection Framework**

In background subtraction, we generally have a sequence of $T$ training images. Typically a statistical model is learned from feature values for each pixel over $1 \leq \tau \leq T$ images of a
sequence which, is later used to identify the objects that do not belong to the scene. For the rest of the paper, we will model every pixel individually and carry out the same operation on each of them. Given $\mathcal{T}$ observed values $x^1, x^2, \cdots, x^T$ of certain features of a particular pixel, we can build a statistical model for the probability $\rho(x) = p(x|x^1, x^2, \cdots, x^T)$ of any new observation value $x$ at that pixel given its history. We can use single Gaussian, mixture of Gaussians [51], or Kernel Density estimation (KDE) [28, 88] to generate this model.

**Which Feature to Use:** Let us suppose we have $M$ different types of features and $x_j$, $1 \leq j \leq M$ is the $j$-th feature value (e.g. R,G,B, intensity, spatial gradients, temporal gradients, optical flow, etc.) for a certain pixel of an image. As discussed in the introduction, pixels of different part of the image can exhibit different characteristics. This observation leads us to claim that different features rather than a single one may produce better accuracy in modeling the background distribution at each pixel. Our objective is to find out the more useful features from a pool of $M$ features, use weights to quantify their importance and use their weighted combination to differentiate foreground objects from the background. The selected subset of features can be different from one pixel to another based on the characteristics of each pixel in the background (tree waving, water, sky, static road, etc.). This process of feature selection is supposed to be done only once at system initialization (e.g. while installing a new camera at a new scene). Once this process is done, a feature map is generated for the scene background depicting which features are best for which pixel in the image. This feature map can be used afterward to model the background. This process does not need to be performed again unless there is some significant structural change in the scene.

**Model Parameter Selection:** The choice of the parameters of the estimation process also affects the background model by a huge amount. For example, we know that KDE is vulnerable to the choice of the bandwidth parameter $\sigma$. It can produce totally erroneous output for a wrong choice of $\sigma$. Consider the situation when the kernel bandwidth does not correspond to the variance of the feature values we are trying to model. For pixels in an outdoor scene, the variance will be substantially different from one another. To argue in favor of this statement, let us draw histograms of pixels of an outdoor image shown in Figure C.1. The histograms for only green color values for pixels at a, b, c, d indicated in the input image (Figure C.1) over 200 image frames are shown in Figure C.2. As we can see, pixels in tree leaves (a, b) have higher
variances than that of pixels in more static regions like sky and grass (c, d). Therefore, kernel estimation with same bandwidth will fail to estimate the density of many pixels.

**Boosted Background Classifier:** The framework introduced in this paper provide a solution for these problems by choosing best features and parameters at each pixel out of a pool of feature/bandwidth combinations. We want to generate a classifier for pixel \( x \) of the following form

\[
F(x) = \sum_{t=1}^{T} w_t f_{j_t}(x) \tag{C.1}
\]

such that we will classify a certain pixel to be a background pixel if \( F(x) > 0 \) and a foreground object if \( F(x) \leq 0 \). In equation C.1, \( f_{j_t}(x) \) is some function of \( x_{j_t} \), where \( j_t \) is the index of \( t \)-th feature chosen and \( w_t \)'s are the values to measure the importance of \( f_{j_t}(x) \). From what we have already discussed, some function of the probability estimates of \( x_j \) may be a choice
for $f_{ji}(x)$ in equation C.1. The summation in equation C.1 is over $T$ of these functions where $T$ is a small number, representing the top features selected to model the background at each individual pixel. We can generate these estimates for all $M$ feature types and for a range of parameters of the estimation process. Therefore, we need an algorithm to select optimal feature and parameter from this pool of features and parameters for background subtraction. The algorithm should also calculate the weights for it’s choice of features simultaneously.

The boosting algorithm, RealBoost [110] seems ideal for this scenario. Unlike Adaboost (which combines weak hypotheses having outputs in $\{-1, +1\}$), RealBoost algorithm computes real-valued weak classifiers given real numbered feature values, and generates a linear combination of these weak classifiers that minimizes the training error. We can expect that, using the density estimates, the Realboost algorithm will be able to select the features most appropriate for any specific pixel.

### C.3 RealBoost

Let $S = \{(x^1, y^1), \cdots, (x^N, y^N)\}$ be a sequence of training examples where each instance $x^i, i = 1, 2, \cdots , N$ belongs to an instance space $\mathcal{X}$ and each label $y^i$ belongs to a label set $\mathcal{Y}$. We will consider the case of binary classification where $\mathcal{Y} = \{-1, +1\}$. Suppose we have $M$ different types of features for each $x \in \mathcal{X}$. Denote $x_j \in \mathcal{R}$ as the $j$-th feature value of $x$ and $\rho_j(x) = g(x_j)$ where $g : \mathcal{R} \to [0, 1]$ is some function applied to the feature values. The aim of boosting algorithm is to find a linear combination of some functions $h_j(x)$ of $\rho_j(x)$ of the form

$$ H(x) = \sum_{t=1}^{T} \alpha_t h_{ji}(x) , $$

where $h_{ji}(x) = f(\rho_j(x))$  \hspace{1cm} (C.2)

where $T$ is a small number of selected functions. According to this equation, an instance $x$ can be classified as an example of class $y^i = +1$ if $H(x^i) > 0$ and as an example of $y^i = -1$ otherwise [36, 110]. The $h_j(x) : [0, 1] \to \{-1, 1\}$ are called the weak hypothesis $^1$. At each round of boosting [36, 110], we choose a particular $h_{ji}(x)$ from all $h_j(x), j = 1, 2, \cdots M$;

$^1$Notice that, $h_j$ is actually a function of $\rho_j(x)$. We use the notation $h_{ji}(x)$ instead of $h(\rho_j(x))$ for simplicity.
i.e. we add one more term to the summation. Each weak hypothesis is given a weight \( \alpha_t \) based on its performance in classifying the training examples. These weights quantify how much confident we are on the corresponding hypotheses in predicting the unknown test cases.

In [110], the weak hypotheses were modified to produce real-valued output, i.e. \( h(x) : \mathbb{R} \rightarrow \mathbb{R} \) for suitably chosen \( h(x) \) on \( x \) so that it absorbs the corresponding confidence weights \( \alpha_t \)'s into themselves. We will adopt this strategy so that the weak hypotheses in equation (C.3) can be defined as \( h_j(x) : [0, 1] \rightarrow \mathbb{R} \). This can be accomplished by designing a weak learner algorithm such that the \( \text{sign}(h_j(x)) \) denotes the predicted label for \( x \) and \( |h_j(x)| \) denotes the 'confidence' of the prediction. Then the form of the strong classifier becomes

\[
H(x) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_j(x) \right). \tag{C.3}
\]

Let \( \Psi_j \) be the set values obtained by applying the \( \rho_j(\cdot) \) function at the training instances, i.e. \( \Psi_j = \{ \rho_j(x^1), \rho_j(x^2), \ldots \rho_j(x^N) \} \), where \( j = 1, 2, \ldots, M \). One approach to compute \( h(x) \) is to partition the set \( \Psi \) into disjoint subsets \( \Psi_1^j, \Psi_2^j, \ldots, \Psi_B^j \) and assign the same \( h^*_j \) to all \( h_j(x^*) \) such that \( \rho_j(x^*) \in \Psi_b^j \), where \( 1 \leq b \leq B \). The idea is to partition \( \mathcal{X} \) into smaller sets containing \( x \)'s that are similar in some sense according to the value of the function \( \rho_j(\cdot) \) and then use a single representative value of \( h^*_j \) for all instances in the same partition. Let us assume that we know how to partition \( \Psi \) into \( B \) subsets and define the term

\[
W_{t,c}^{j,b} = \sum_{i : \rho_j(x^*) \in \Psi_b^j \cap y^i = c} D_t(i)
\]

where \( c \in \mathcal{Y} \) and \( D_t(i) \) is the normalized weight associated with instance \( x^i \) at iteration \( t \) of the boosting [110]. It has been shown in [110] that the cost function (i.e. the error term) is minimized for a particular feature \( j \) at iteration \( t \) of boosting if \( h_j(x) = \frac{1}{2} \ln\left(\frac{W_{t,+}^{j,b}/W_{t,-}^{j,b}}{W_{t,-}^{j,b}/W_{t,+}^{j,b}}\right) \) for all \( x \) such that \( \rho_j(x) \in \Psi_b^j \). The minimized value of the cost function becomes

\[
Z_t^j = 2 \sum_b \sqrt{W_{t,+}^{j,b}W_{t,-}^{j,b}}; \quad j = 1, 2, \ldots, J \text{ at iteration } t. \tag{C.3}
\]

The algorithm determines \( j_t \) as follows

\[
j_t = \arg \min_{j \in J} Z_t^j.
\]

Analyzing the formulae above, we can interpret the quantity \( W_{t,+}^{j,b}/W_{t,-}^{j,b} \) as a measure of discriminative power of the weak hypothesis for \( x \) such that \( \rho_j(x) \in \Psi_b^j \).
All that we are left with is to find out a way to partition the instance set $\Psi_j$ into $\Psi_{j1}, \Psi_{j2}, \ldots, \Psi_{jB}$.

The function (or estimate) $\rho_j(x)$ of feature values lies within the range $[0, 1]$. We can always build a histogram within the range $[0, 1]$ with $B$ bins: $\text{bin}_b = \left(\frac{(b-1)}{B}, \frac{b}{B}\right)$ for $b = 1, 2, \ldots, B$.

Then we define the subset $\Psi^b_j = \{\rho_j(x) : \rho_j(x) \in \text{bin}_b\}$. Notice that this is a histogram of the values of the $\rho_j$ functions and not a histogram of the feature values themselves. Essentially we are creating partitions $X_1, X_2, \ldots X_B$ of the instance space $X$ comprising instance $x$ with close $\rho_j(x)$ values; i.e. with similar properties that are being quantified by $\rho_j$. Given these settings, we can describe the RealBoost algorithm as:

- Initialize the weights $D_1(i) = \frac{1}{N}$
- For $t = 1, 2, \ldots, T$
  1. For $j = 1, 2, \ldots, M$, Decompose $\Psi^j$ into $\Psi^j_1, \Psi^j_2, \ldots, \Psi^j_B$
  2. Calculate
     \[ W_{t,c}^j = \sum_{i: \rho_j(x^i) \in \Psi^j_c} D_t(i), \]
     for $j = 1, 2, \ldots, M$; and $b = 1, 2, \ldots, B$; where $c \in \{-1, +1\}$.
  3. Set the output of $h$ for all $x$ such that $\rho_j(x) \in \Psi^j_c$ as $h_j(x) = \frac{1}{2} \ln \left(\frac{(W_{t,+}^j + \epsilon)/(W_{t,-}^j + \epsilon)}{(W_{t,+}^j - W_{t,-}^j + \epsilon)}\right)$ where $\epsilon$ is a small positive constant for numerical stability.
  4. Select $h_{jt}$ minimizing the normalization factor $Z^j_t = 2 \sum_{b=1}^{B} \sqrt{W_{t,+}^{j,b} W_{t,-}^{j,b}}$; $j_t = \arg \min_j Z^j_t$
  5. Update $D_{t+1}(i) = D_t(i) \epsilon^{-y^i h_{jt}(x^i)}$
- The strong classifier is: $H(x) = \text{sign} \left[ \sum_{t=1}^{T} h_t(x) \right]$

C.4 Feature Selection in Background Subtraction

C.4.1 Statistical Background model

The density functions of the feature values constitutes the statistical model for the background. Using the notation of section C.2, the probability density function (pdf) for $j$-th feature for any pixel is estimated by Kernel Density Estimation (KDE) [28, 88] as the following

\[ p(x_j) = \frac{1}{T} \sum_{\tau=1}^{T} K_\sigma(x_j - x_j^\tau), \quad (C.4) \]

\footnote{A complete theoretical step by step construction and analysis of the boosting algorithm can be found in [110].}
where $K_\sigma$ is a kernel function. We use such probability estimate as the function $\rho_j(\cdot)$, i.e.
$\rho_j(x) = p(x_j)$, which (as defined in section C.3) maps from the real-valued feature to the $[0, 1]$ range. Choosing $K$ to be a Gaussian kernel $N(0, \sigma)$, the pdf can be written as

$$\rho_j(x) = \frac{1}{N \cdot C \cdot \sigma} \sum_{t=1}^{T} \exp(-\frac{1}{2\sigma^2}(x_j - x_j^t)^2).$$

(C.5)

Here $\sigma$ is the bandwidth parameter of the kernel function. We will use these estimates to learn the weak classifiers.

### C.4.2 RealBoost in Feature Selection

We divided the training set into two parts- first $T$ images are used to generate the statistical model for the background and then next $N$ images are used for boosting. All these $T + N$ images are assumed to be free of foreground objects. In this study, we consider background pixels as the positive examples. We have $M$ one dimensional features for each pixel to choose from (e.g. R,G,B, intensity, spatial gradients, temporal gradients, optical flow, etc.). For each of the features, we also need to select the parameters of KDE that produces the best weak classifier. Therefore, we calculate the KDE estimates of the feature values using different parameters and let RealBoost chose the best one.

Suppose the set of bandwidths used is $\Sigma = \{\sigma_1, \sigma_2 \ldots \sigma_Q\}$. So, for every pixel, we have $M \times Q$ feature/parameter combinations; i.e. we have $Q$ functions $\rho_{jq}$ for feature $j$ with bandwidth $\sigma_q \in \Sigma$. Consequently, we also have $M \times Q$ number of $\Psi_j(q)$ sets. Each of these sets will then be partitioned into $B$ bins to calculate the corresponding weak classifiers $h_j(x)$ (for each bin).

Let the $j$-th feature values for any pixel in $N$ consecutive images be $x_j^1, x_j^2, \ldots, x_j^N$. Given these values, the KDE estimates with parameter $\sigma_q$ are $\Psi_j(q) = \{\rho_{jq}(x^1), \rho_{jq}(x^2), \ldots \rho_{jq}(x^N)\}$. These values will be used to generate the weak classifiers in the boosting procedure (see section C.3). The negative examples are generated randomly from a uniform distribution. The KDE estimates of these random negative examples are also provided to the boosting algorithm along with the $\rho_{jq}(x)$s. Finally, we will have a strong classifier $H$ for each pixel as the output of the RealBoost algorithm by combining $T$ out of $M \times Q$ weak classifiers. Any new value $x_{new}$ of $x$ will be classified as a background pixel (positive example) if $H(x_{new}) > 0$ and as a
As discussed in section C.3, in each bin, the weak classifier quantifies the discriminative power of the corresponding feature (along with the associated parameter for KDE). We may also view the approach as a histogram approach for estimating the distribution of the $\rho_{jq}(x)$ values for some feature $j$ and bandwidth $\sigma_q$. Let us call the KDE estimates of positive examples as $\rho(x_+)$ and that of negative examples as $\rho(x_-)$ for any feature values computed with some bandwidth. For a feature to be useful in background subtraction, the probability estimates of feature values of positive examples should be substantially different than that of negative examples; i.e. the distributions of the $\rho(x_+)$ need to be fairly separated from that of $\rho(x_-)$. In other words, the KDE with a specific bandwidth is supposed to keep the amount of 'mixing' of $\rho(x_+)$ and $\rho(x_-)$ as low as possible. For example, in Figure C.4, the red and blue bars corresponds to the histogram bins of $\rho(x_-)$ and $\rho(x_+)$ respectively. The labels of the sub-images state the feature-bandwidth combination used for respective histogram. In this figure, the feature R with bandwidth 200 will be the most useful feature-parameter combination. In terms of probability, this is the same as saying a good feature-parameter combination should keep the estimated probability (approximated by the histogram) of $\rho(x_+)$ to lie in some range of values as different as possible than that of $\rho(x_-)$ to lie in the same range. This is precisely what is being done when we are minimizing the cost function $Z_1^{j_1} = 2 \sum_{b=1}^{B} \sqrt{W_{1,+}^{j,b}W_{1,-}^{j,b}}$. Recall that the first iteration of boosting all $D_1(i)$’s are equal. We are choosing the feature and bandwidth value that decreases the probability that $\rho(x_+)$ and $\rho(x_-)$ assume values in the same range. Therefore, the weak hypothesis calculates the difference between the log-probabilities of $\rho(x_+)$ and $\rho(x_-)$. In the subsequent iterations, the $D_t(i)$ is changed based on the performance of the of $h_{jt-1}$. Then, the examples that were missed in the last iteration are emphasized to minimize the probability that both $\rho(x_+)$ and $\rho(x_-)$ are present in bin $b = 1, 2, \ldots B$. Thus, the strong classifier simply gives us the differences of log-likelihoods for $\rho(x_+)$ and $\rho(x_-)$.

Since we are using random examples, the estimates for negative examples will be distributed all over the range $[0, 1]$. This is apparent from the histograms of estimates in Figure C.4. The presence of estimates for negative examples in the intervals where the estimates of positive examples are concentrated, will pull down the confidence level of the corresponding weak hypothesis. Consider the scenario where, pixel x is a background pixel (and hence a positive
example) but there is at least one weak classifier $h_r(x)$ in $H(x)$ that does not classify $x$ correctly. If the confidence value of $h_r(x)$ is high and the rest of the weak classifiers had their confidence values reduced, the value of $H(x)$ may become negative. Therefore, we cannot rely on the theoretical bias of 0 anymore. We use a bias $\delta < 0$ and modify the strong classifier in equation C.3 as follows:

$$H(x) = \text{sign} \left( \sum_{t=1}^{T} h_{j_t}(x) - \delta \right).$$  \hspace{1cm} (C.6)

This bias $\delta$ is the only control parameter in our algorithm. One method to determine the optimum $\delta$ is to change it’s value iteratively until a certain rate of false positives is attained [28].

### C.5 Experiments & Results

We implemented our algorithm on several indoor and outdoor (both color and gray level)image sequences. All the images used in the experiments had dimensions 320 $\times$ 240 pixels. In total 9 types of features, namely three color values R,G,B and spatial derivatives for each of these color channels in both $x$ and $y$ directions for each pixel of a color image, were used in this study. Some experiments only used a subset of these features. For intensity images, we used 3 features- the pixel intensity value and its spatial gradients in horizontal and vertical directions.

The first image sequence was a video taken from outside an office. The images in this video contains trees with moving leaves; cars and pedestrians on the road etc. We used $T = 100$ kernel points for KDE. The first experiment was carried out with only color values R,G,B as features and with kernel bandwidths of $\Sigma_R \in \{40, 160, 200\}$, $\Sigma_G \in \{60, 160, 240\}$ and $\Sigma_B = \{60, 100, 240\}$. We generated one dimensional KDE estimates for all these color features with each of the bandwidths in the corresponding $\Sigma$ set. Therefore, if we wish to produce a strong classifier with $T$ weak classifiers, there can be $\binom{9}{T}$ possible combinations to select from. We used 200 frames to learn the RealBoost classifier, i.e. to learn which features are performing better than the others. In this experiment the negative examples are 400 samples from a uniform distribution over [0, 255] for color values.

We anticipate the boosting process to choose features with larger bandwidth estimates in the regions surrounding pixels a and b in Figure C.1, and to select smaller bandwidth estimates in regions surrounding c. Figure C.3, shows some sample classifier maps. Classifier maps are
binary images $I_C$ to describe the feature and parameter selection. For any feature/parameter combination, we set a pixel in $I_C$ to 1, if the feature and bandwidth were chosen by RealBoost for that pixel. We are using $T = 3$ for the boosting process, i.e. each pixel can be chosen at most three times in all the classifier maps. Each classifier map is labeled by the feature name and kernel bandwidth at the bottom in Figure C.3. As we can see (refer to Figure C.1), in the low variance regions (e.g. the sky), boosting method selected weak classifier calculated using smaller bandwidth KDE estimates. Similarly, in regions with high variance in the feature values, weak hypotheses calculated with large bandwidth estimates were selected. Also notice that, the hypothesis using the green value as feature were selected more in number within the leaves of the tree and in the herb. The selection of the hypotheses can be explained better by the plots of the histograms of densities of the $\rho(x)$ at pixel $(101, 31)$ given in Figure C.4. The black and white bars represent the frequencies of $\rho_j(x_+)$ and $\rho_j(x_-)$ respectively in the corresponding bins. The number of bins used to build the histogram of $\rho(x)$ is $B = 10$ for all the experiments performed in this study. We observe that the lower bandwidth estimates are spread out in a wider range, $[0.4, 0.92]$ approximately, whereas the estimates of higher bandwidths are concentrated in $[0.8, 1)$. This will make the value of the cost function $Z_j^t$ (defined in section C.3) of the weak hypothesis constructed from higher bandwidth estimates lower than that of the others. Consequently, the weak hypothesis constructed from higher bandwidth estimates will be selected. Furthermore, among the features estimated with higher bandwidth,
Figure C.4: Distribution of KDE estimates of features with different bandwidths

R has the least spread in its distribution of probability estimates. Therefore, feature R with kernel bandwidth 240 will be the feature to choose for pixel (101, 31) provided that all weights of the training samples are equal.

We used the threshold value of $-13$ for the first sequence (at a false positive rate $5 \times 10^{-3}$ pixels/frame), which gives us the output shown in Figure C.5, bottom row. It should be made clear that no post-processing operation (e.g. removal of connected components) was applied to any of the outputs of the experiments we performed. Inclusion of spatial gradients produced similar output for these images.

Figure C.5 also compares the output using our approach with that of [88]. In [88], normalized color values along with the optical flow vector were used as features to generate the background model using Adaptive Kernel Density Estimation (AKDE). The approach of [88] was shown to be superior to the original KDE approach in [28]. For a fair comparison, the outputs of [88] were also displayed without any post-processing. Clearly, our method produces significantly fewer number of false detections than that of [88]. To make a quantitative comparison, we used an image sequence without any foreground objects and imposed a synthetic object to pass over the regions where it is most likely to have foreground objects. We
Figure C.5: Detection results using R,G,B only. The middle column uses the method of [88] and the right column shows the result of our approach.

used a circle of radius 10 pixels as the synthetic image generated by the method described in [28]. Figure C.6 depicts the ROC curves for both the algorithms. The proposed method supersedes the AKDE method both in terms of false positives and detection rate when used without the post-processing steps. The strategy of adaptive selection of elementary features for each pixel is shown to significantly outperform the method of background subtraction using adaptive statistical models with additional complex features. We also tested our method

Figure C.6: Receiver-Operator Characteristic (ROC) curves for results using proposed method and using the method in [88] (without post-processing)

with the same fatures, on an image sequence of a rainy day. For color channels R,G,B, three bandwidth triplets \{140, 100, 100\}, \{40, 30, 30\} and \{80, 80, 80\} were chosen to estimate their density. The first 50 images were used to learn the density and the next 170 images were used
to learn the RealBoost classifier. A total of 400 negative examples were sampled from $[0, 255]$. Figures C.7 and C.8 show some of the sample outputs and classifier maps respectively. We observe that the learning algorithm was able to segment the image into different regions that are most suitable for a certain feature. In this image sequence, we see that the hypothesis that uses B value as the feature and estimates the density with a kernel bandwidth 20 is most appropriate for the road and the hypothesis using G with the same bandwidth is appropriate for the trees and grasses. We keep changing the bias until a specific false positive rate is achieved. For this sequence, we had a false positive rate of almost zero for a bias of $-8$.

![Output images](image1)

Figure C.7: Outputs from a rainy outdoor sequence

![Classifier maps](image2)

Figure C.8: Classifier maps for the rainy sequence.

The algorithm was next tested on an indoor sequence. The use of only R,G,B as features in indoor images gives rise to lots of false positive due to shadows. So, we tested our algorithm adding six spatial gradients of these three features. The result, shown in Figure C.9, implies
that use of spatial gradients suppresses the shadows totally. It also produces many false negatives within the body of foreground object. Since usually there is not much variation inside the region of a foreground object, the feature values of the pixels inside the foreground object region produces high estimates given the background model. The reason behind this is that, in this case, the background model for spatial gradients is centered around zero. The two sets of kernel bandwidth used were \{50, 160\} and \{40, 10\} respectively for color values and spatial gradients. The bias value used for the strong classifier is \(-12\). The images used to learn the model and to choose the features were 80 and 120, respectively.

![Image](image_url)

Figure C.9: Results for the indoor sequence. Left- Input, middle- Output with R,G,B, right- Output with R,G,B + Spatial gradients.

Our final experiment was carried out on a sequence of gray level images taken in the woods. This sequence is also considered comparatively difficult since the foreground object often becomes occluded by tree branches and the background also has moving parts like tree leaves. Two sets of kernel bandwidths \{15, 50, 150, 200\} and \{60, 160, 200\} were used for the two types of features, namely intensity and spatial gradients. A bias of \(-15\) for the final strong classifier gives the output as in Figure C.10. Again, the boosting algorithm was able to pick a higher bandwidth for the upper regions with moving tree leaves and a lower bandwidth for the static portion of the image (Figure C.11).
C.6 Conclusion

A new framework for background subtraction has been proposed which addresses the problem as a classification problem. The classification algorithm will be generalized in the sense that after it has been learned for the first time, it will be able to satisfactorily subtract background from the image unless there are some structural changes in the scene. The choice of the classifier facilitates combining different features, which is analytically more justifiable and has been experimentally shown to perform better than previous methods [28, 88]. The learning method has been shown to follow a clear pattern to choose features which is interpretable in terms of the input image. However, the use of 1D features like spatial gradients introduces some negative examples which is inevitable in our experimental setup. As a future work, a combination of higher dimensional features with some additional constraints may be tried so that the adverse effect of one feature can be compensated by the contribution of the others.
References


Vita

Toufiq U Parag

2011 Ph. D. in Computer Science, Rutgers University, NJ (Expected).
2006 Master in Computer Science, Rutgers University, NJ.
2003 B Sc Engg in Computer Science and Engineering, Bangladesh University of Engineering and Technology, Dhaka, Bangladesh.

Fall 2009- Fall 2010 Teaching assistant, Department of Computer Science, Rutgers University.
April 2009- August 2009 Intern, Siemens Corporate Research, Princeton, NJ.
June 2008- March 2008 Research Associate, Center for Vision Research (CVR), York University, Toronto, Canada.
June 2006- August 2006 Intern, Mitsubishi Electric Research Labs, Cambridge, MA.
Spring, Fall 2005- Spring 2006 Teaching assistant, Department of Computer Science, Rutgers University.
Summer 2005, Spring and Fall 2004 Research assistant, Department of Computer Science, Rutgers University.
Summer 2004 Intern, Siemens Corporate Research, Princeton, NJ.