NORTHWESTERN UNIVERSITY

Image and Video Data Mining

A DISSERTATION

SUBMITTED TO THE GRADUATE SCHOOL
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS

for the degree

DOCTOR OF PHILOSOPHY

Field of Electrical Engineering and Computer Science

By

Junsong Yuan

EVANSTON, ILLINOIS

December 2009
ABSTRACT

Image and Video Data Mining

Junsong Yuan

The rapid advances in image content capture, storage and communication technologies have brought an opportunity to discovering knowledge from emerging image and video dataset. Despite of a lot of previous work in data mining and knowledge discovery, data mining techniques that are successful in text and transaction data may not simply apply to image and video data that are non-structured. Due to the structure and content variations in the visual data, it is not a trivial task to discover meaningful visual patterns from images and videos. This thesis presents a systematic study on image and video data mining and concentrates on two fundamental problems: (1) mining interesting patterns from image and video data, including common pattern discovery and semantically meaningful visual pattern discovery from images, as well as recurring events mining from videos and (2) discovering discriminative visual patterns for multimedia pattern recognition. The difficulties of structure and content variations for mining patterns are addressed and efficient data mining methods are studied to handle the large image and video dataset. Data mining algorithms toward image and video data are developed, which are crucial for understanding, organizing and retrieving large image and video data.
To my parents and to Jingjing
Acknowledgments

Foremost, I would like to express my sincere gratitude to my advisor Prof. Ying Wu for the continuous support of my Ph.D study and research, for his insightful guidance, immense knowledge, motivation, and enthusiasm. His guidance helped me in all the time of research and writing of this thesis.

I would like to thank the rest of my thesis committee: Prof. Aggelos K. Katsaggelos, Prof. Thrasyvoulos N. Pappas, and Dr. Jiebo Luo, for their encouragement, insightful comments, and hard questions.

My sincere thanks also goes to Dr. Zicheng Liu, Dr. Jiebo Luo, Dr. Wei Wang, Dr. Zhu Li, and Dr. Dongge Li, for offering me the summer internship opportunities in their groups and leading me working on diverse exciting projects.

I thank my fellow labmates in Northwestern Vision Group: Gang Hua, Ting Yu, Zhimin Fan, Ming Yang, Shengyang Dai, Jiang Xu, Jialue Fan, and Neelabh Gupta, for the stimulating group meetings and discussions, and for all the fun we have had in the last five years.

I want to thank my parents for all their endless love, support and encouragement though all the time of my study. Finally, but not least, I would like to express my deep thanks to my wife, Jingjing, for all of her love, sacrifice, understanding and help during my PhD study.
# Table of Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>ABSTRACT</td>
<td>3</td>
</tr>
<tr>
<td>Acknowledgments</td>
<td>5</td>
</tr>
<tr>
<td>List of Tables</td>
<td>8</td>
</tr>
<tr>
<td>List of Figures</td>
<td>10</td>
</tr>
<tr>
<td>Chapter 1. Introduction</td>
<td>17</td>
</tr>
<tr>
<td>Chapter 2. Mining Common Patterns from Image Data</td>
<td>22</td>
</tr>
<tr>
<td>2.1. Proposed Approach</td>
<td>24</td>
</tr>
<tr>
<td>2.2. Properties of the Algorithm</td>
<td>33</td>
</tr>
<tr>
<td>2.3. Experiments</td>
<td>37</td>
</tr>
<tr>
<td>2.4. Conclusion</td>
<td>42</td>
</tr>
<tr>
<td>Chapter 3. Mining Semantic Meaningful Patterns from Image Data</td>
<td>43</td>
</tr>
<tr>
<td>3.1. Introduction</td>
<td>43</td>
</tr>
<tr>
<td>3.2. Discovering Meaningful Visual Itemsets</td>
<td>50</td>
</tr>
<tr>
<td>3.3. Self-supervised Clustering of Visual Item Codebook</td>
<td>55</td>
</tr>
<tr>
<td>3.4. Pattern Summarization of Meaningful Itemsets</td>
<td>58</td>
</tr>
<tr>
<td>3.5. Experiments</td>
<td>62</td>
</tr>
<tr>
<td>3.6. Conclusion</td>
<td>68</td>
</tr>
</tbody>
</table>
List of Tables

2.1 Computational complexity comparison 37

2.2 Performance evaluation. The superscript * denotes the dataset containing multiple common patterns. See text for details. 40

3.1 Transaction database $T_1$. 52

3.2 Precision score $\rho^+$ and retrieval rate $\eta$ for the car database, corresponding to various sizes of $\Psi$. See text for descriptions of $\rho^+$ and $\eta$. 64

3.3 CPU computational cost for meaningful itemsets mining in face database, with $|\Psi| = 30$. 65

4.1 Complexity analysis and comparison. The cost of feature extraction is not considered. The parameter $\alpha > 1$ is the approximation factor determined by $\epsilon$ of $\epsilon$-NN query and the correct-retrieval probability $p$ of LSH. For the method in [1], $L$ is a constant depending on the sampling rate of the VPs. 85

4.2 Computational Cost with $\epsilon = \mu - 3\sigma$ and $B = 3$. 93

4.3 Comparison of basic forest-growing and improved forest-growing ($K = 24$ and $N = 94008$). 93

4.4 Comparison of different branching factors ($\epsilon = \mu - 3\sigma$). 95

4.5 Comparison of different criteria in selection $\epsilon$ for NN-search ($3\sigma$-criterion v.s. $2\sigma$-criterion). Branching factor $B = 3$. 95
4.6 Discovery of recurring motion patterns. The errors are highlighted. The total error rate for clustering is 11.29%.

5.1 Compositional feature discovery in 3 UCI data sets.

5.2 A 10-class event recognition dataset. The first row describes event class name; the second and third row indicate the number of events and images in that class respectively.

5.3 Boosting compositional features: class confusion matrix of leave-one-out test results. Each row indicates the classification results of the corresponding class. The overall accuracy is 80.7%.

5.4 Order distribution of the mined weak classifier pool $\Psi$. The values are averaged by all the leave-one-out tests.
List of Figures

2.1 Can you find those common posters (patterns in this case) in the two images? There are three of them (see Sec. 2.3). It is not an easy task, even for our human eyes. 23

2.2 Illustration of the basic idea of spatial hashing. There are three images, each of which contains the same common pattern $P$, which is represented by the orange rectangle. Each column corresponds to a same image ($t = 1, 2, 3$). Note this common pattern exhibits variations like rotation (the second image) and scale changes (the third image). We perform a $G \times H$ (e.g. $3 \times 3$) partitions for each image for $K$ (e.g. 3) times. Each of the first three rows shows a random partition ($k = 1, 2, 3$). The highlight region of each image indicates a good subimage (7 in total out of 81 candidates) that contains the common pattern. All of these good subimages are also popular ones as they can find enough matches (or supports) in the pool. The bottom row is a simple localization of the pattern, which is the intersection of the popular subimages in the corresponding image. 25

2.3 Similarity measure of two visual primitives $s(v,u)$, where $a,b,c,d,e,f$ denote visual primitives. We notice $s(v,u) = 0$ when $v \notin M_u$ and $u \notin M_v$. 29

2.4 Similarity matching of two subimages. Each point is a visual primitive and edges show correspondences between visual primitives. The flow between $V_R$ and $V_Q$ can be approximated by the set intersection $\tilde{Sim}(V_R, V_Q)$. 30
2.5 Illustration of the EVR. The figures show two different random partitions on the same image. The small orange rectangle represents the common pattern $P$. We compare two pixels $i \in P$ and $j \notin P$. The large blue region represents $R'_j$, the EVR of $j$; while $R'_i = P$. In the left figure, $R'_i$ is broken during the partition while $R'_j$ is not. Thus $i$ get a vote because $R_4$ (shadow region) is a popular subimage and the whole region is voted; while $j$ does not receive the vote. In the right image, both $R'_i$ and $R'_j$ are broken during the partition, so neither $i$ and $j$ is voted as no popular subimage appears.

2.6 The input is from Fig. 2.1. The 1st and the 2nd row show the voting maps and the rough segmentations respectively. The three common patterns are listed from the 3rd to the 5th row. Besides variations like rotation and scale change, the second poster suffers from partial occlusion in the left image. The hit ratio and the background ratio are $h_r = 0.76$ and $b_r = 0.29$ respectively.

2.7 The input is a set of four images (the 1st column). Each row corresponds to an image.

The common pattern “no parking sign” appears in the first three images. A comparison of applying the different partition times $K$ is shown in the 2nd ($K = 25$), 4th ($K = 50$) and 6th ($K = 75$) columns. The 3rd, 5th and 7th columns are voting maps associated with the corresponding images. More results can be reviewed at the supplemental materials.

2.8 Spatial hashing for image irregularity detection. An instance of the spatial hashing and the unpopular subimage are shown in the 2nd image. The subimage that contain green circles (denoting visual primitives) is the unpopular one. After segmenting the voting map (the 3rd image), we obtain the final irregularity detection result (the 4th image).

3.1 Overview for meaningful visual pattern discovery.
3.2 Illustration of the frequency over-counting caused by the spatial overlap of transactions.

The itemset \{A, B\} is counted twice by \( T_1 = \{A, B, C, E\} \) and \( T_2 = \{A, B, D, F\} \), although it has only one instance in the image. Namely there is only one pair of \( A \) and \( B \) that co-occurs together, such that \( d(A, B) < 2\epsilon \) with \( \epsilon \) the radius of \( T_1 \). In the texture region where visual primitives are densely sampled, such over-count will largely exaggerate the number of repetitions for a texture pattern.

3.3 Motivation for pattern summarization. An integral hidden pattern may generate incomplete and noisy instances. The pattern summarization is to recover the unique integral pattern through the observed noisy instances.

3.4 Evaluation of meaningful itemsets mining. The highlight bounding box (yellow) represents the foreground region where the interesting object is located. In the idea case, all the MI \( \mathcal{P}_i \in \Psi \) should locate inside the bounding boxes while all the meaningless items \( W_i \in \Omega^- \) are located outside the bounding boxes.

3.5 Performance comparison by applying three different meaningful itemset selection criteria, also with the baseline of selecting most frequent individual items to build \( \Psi \).

3.6 Comparison of visual item codebook before and after self-supervised refinement.

3.7 Selected meaningful itemsets \( \Psi \) (\(|\Psi| = 10\)) and their summarization results (\(|\mathcal{H}| = 6\)) for the face database. Each one of the 10 sub-images contains a meaningful itemset \( \mathcal{P}_i \in \Psi \).

The rectangles in the sub-images represent visual primitives (e.g. PCA-SIFT interest points at their scales). Every itemset, except for the 3rd one, is composed of 2 items. The 3rd itemset is a high-order one composed of 3 items. Five semantic visual patterns of the face category are successfully discovered: (1) left eye (2) between eyes (3) right eye (4) nose and (5) mouth. All of the discovered meaningful visual patterns have very
high precision. It is interesting to note that left eye and right eye are treated as different semantic patterns, possibly due to the differences between their visual appearances. One extra semantic pattern that is not associated with the face is also discovered. It mainly contains corners from computers and windows in the office environment.

3.8 Selected meaningful itemsets $\Psi (|\Psi| = 10)$ and their summarization results ($|H| = 2$) for the car database. Two semantic visual patterns that are associated with the car category are successfully discovered: (1) wheels and (2) car bodies (mostly windows containing strong edges). The $5_{th}$ itemset is a high-order one composed of 3 items.

3.9 Examples of meaningful itemsets from car category (6 out of 123 images). The cars are all side views, but are of different types and colors and located in various clutter backgrounds. The first row shows the original images. The second row shows their visual primitives (PCA-SIFT points), where each green circle denotes a visual primitive with corresponding location, scale and orientation. The third row shows the meaningful itemsets. Each red rectangle in the image contains a meaningful itemset (it is possible two items are located at the same position). Different colors of the items denote different semantic meanings. For example, wheels are dark red and car bodies are dark blue. The precision and recall scores of these semantic patterns are shown in Fig. 3.8.

3.10 Examples of meaningful itemsets from face category (12 out of 435 images). The faces are all front views but are of different persons. Each red rectangle contains a meaningful itemset. Different colors of the visual primitives denote different semantic meanings, e.g. green visual primitives are between eyes etc. The precision and recall scores of these semantic patterns are shown in Fig. 3.7.
4.1 A typical dance movement in the Michael Jackson-style dance, performed by two different subjects (first and second rows). Such a dynamic motion pattern appears frequently in the Michael Jackson-style dance and is a recurring event in the dance database. The spatial-temporal dynamics in human motions can contain large variations, such as non-uniform temporal scaling and pose differences, depending on the subject’s performing speed and style. Thus it brings great challenges in searching and mining them.

4.2 Mining recurring events through finding continuous paths in the matching-trellis. Each node denotes a primitive $S \in \mathcal{V}$, labeled by its temporal index. We show part of the whole matching-trellis from column 101 to 110. Given the dataset $\mathcal{V}$, i.e., the top row sequence, we query each $S \in \mathcal{V}$ and find its $K$ best matches, i.e., each column denotes a matching set $\mathcal{M}_S$. For instance, the matching set of $S_{101}$ is $\mathcal{M}_{S_{101}} = \{S_{120}, S_{122}, S_{410}, S_{374}, S_{198}, S_{721}\}$. The highlighted nodes constitute an established tree, which grows from left to right, with the temporal index increasing monotonically from root to leaf nodes. Each tree branch is a continuous path in terms of temporal indices, which indicates a discovered repetition corresponding to the original sequence in the top row. For example, the branch $\{S_{720}, S_{722}, S_{725}, S_{729}, S_{731}, S_{733}, S_{736}\}$ is a repetition of the top row segment $\{S_{101}, S_{102}, S_{103}, S_{104}, S_{105}, S_{106}, S_{107}\}$. The longest branch highlighted in orange is picked to represent the whole tree. Although we only show a single tree growing, because the algorithm mimics the process of growing multiple trees simultaneously, we call it “forest-growing”.
4.3 Improved forest-growing step from column 312 to 313, with branching factor $B = 3$.

(a) Two columns 312 and 313 in the matching-trellis; (b) The auxiliary array associated with column 313; each element in the first column stores a binary flag indicating whether the corresponding node is available (0 means not). The second column stores the row index of each node in column 313, e.g. 928 is in the 3rd row of 313; (c) The updated auxiliary array after growing from 312 to 313. In (a), the colored pair of numbers next to each node is the branch message $\{\text{Root, Length}\}$ to be passed to the descendants during growing, e.g. node 927 belongs to a tree branch whose root is 857 and the current length is 70. When node 927 grows to node 928 in the next column, it updates the message from $\{\text{Root} = 857, \text{Length} = 70\}$ to $\{\text{Root} = 857, \text{Length} = 71\}$ and pass it to 928. The three colors denote different branch status: a live branch (yellow), a new branch (purple) and a dead branch (green). See texts for detailed description.

4.4 How the branching factors $B$ influences the relationship between the branch length $L$ and its breaking probability $\text{Prob}_t$. The curve is drawn based on Eq. 4.4, with $p = 0.9$.

4.5 Selection of effective length $\lambda$, with $\epsilon = \mu - 2\sigma$ and $\frac{K}{N} \approx 0.022$.

4.6 A discovered recurring event consists of three motion instances. Each instance is a segment in the long motion sequences; the figures are sampled every 10 frames. All of the three instances belong to the same event in break dance, but are of various length and dynamics. More results of the discovered recurring motion patterns can be seen at http://www.ece.northwestern.edu/~jyu410/motionmotifs/.
5.1 Illustration of the induced transaction. By partitioning the feature space into sub-regions through decision stumps $f_A$ and $f_B$, we can index the training samples in terms of the sub-regions they are located. Only positive responses are considered. For example, a transaction of $T(x) = \{f_A, f_B\}$ indicates that $f_A(x) > \theta_A$ and $f_B(x) > \theta_B$.

5.2 Feasible and desired regions for data mining. We simulate the upper bound $B_{\epsilon_F} = \frac{1}{\lambda_2} - \lambda_1 \lambda_2 r_k$, with a specific choice of $r_k = 0.5$. The shading region corresponds to $0 \leq B_{\epsilon_F} \leq 1$. As shown in the color bar, the darker the shade, the smaller the upper bound of $\epsilon_F$ and thus are more favorable for data mining. The shading region above the red curve ($\lambda_1 \lambda_2 > 1$) is infeasible for data mining since no rule exists there according to Proposition 1. The region below the red curve ($\lambda_1 \lambda_2 \leq 1$) and between two red lines ($1 \leq \lambda_2 \leq \frac{1}{r_k}$) is the feasible choice of data mining parameters $\lambda_1$ (support) and $\lambda_2$ (confidence). The region below the red curve while above the blue curve ($B_{\epsilon_F} = 0.25$) is the desired region where we can get rules with $\epsilon_F \leq 0.25$. At the optimal point $\lambda_1 = \frac{1}{\lambda_2} = r_k$, where the red curve (tight bound of training error) meets the boundary $\epsilon_F = 0$, we get minimum training error and thus the perfect classification rule.

5.3 Comparison of real training error $\epsilon_F$ and its theoretical upper bound: (a) breast cancer data set, (b) wine data set, and (c) handwritten numeral data set. The closer the point to the 45 degree line, the tighter the upper bound.

5.4 Comparison of error between boosting decision stumps and boosting compositional features. The error is calculated through averaging all leave-one-out tests.
CHAPTER 1

Introduction

Despite decades of efforts in machine learning, artificial intelligence, and computer vision, current machines are still far from being intelligent enough to achieve human’s ability in understanding visual contents, such as images and videos. A big obstacle is that it is difficult to extract useful knowledge from the image and video data, such that machines can better understand the content of images. Although rich information are conveyed in images and videos, they are presented in terms of low-level data forms, from which high-level semantic information needs to be inferred. Such a task, although is intuitive to humans, is extremely difficult for machines.

The recent rapid advances in image and video content capture, storage and communication technologies have ushered an era of unprecedented growth of digital media content. Provided a huge amount of image and video data, it is important to discover interesting patterns and knowledge from them. Such interesting patterns include frequently occurring or semantically meaningful patterns from images and videos. Discovery of these patterns is of broad interests and has important applications in image and video content analysis, understanding and retrieval.

The tremendous amount of image and video data provide opportunities as well as challenges to analyze and understand image and video contents. On one hand, since knowledge can be better conveyed through the versatile data that can be harvested from various sources (e.g. flicker and youtube) rather than a small and specific dataset, it brings a solid foundation
for data-driven approaches for knowledge discovery from images and videos. Non-parametric models are popularly applied, where the learning process can be more effective and less biased by taking advantages of the huge data collection.

On the other hand, since the data is generated at a speed that far surpasses the speed we can analyze and consume them, it becomes a challenging task to discover valuable knowledge without being swamped, not only due to the computational bottleneck of storing and processing such a large data collection, but also because of the lack of data mining methods that can effectively handle the possibly noisy huge amount of data. For example, it is very difficult and time-consuming for machines to search a desired image or video clip from the web, or to understand and detect abnormal events from surveillance videos.

It is of essential importance to investigate novel data mining methods that can effectively and efficiently discover patterns from image and video data. However, without any \textit{a priori} knowledge of the interesting patterns, such as their sizes, contents, locations and the total number, it remains a challenging problem due to the enormous computational cost involved in searching the huge datasets. Traditional data mining techniques that are successful in transaction and text data cannot be simply applied to video and image data that contain high-dimensional features and have complex temporal or spatial structures. It is not a trivial task to discover meaningful visual patterns from images and videos, because the content variations and spatial/temporal dependency in the visual data greatly challenge most existing methods.

In this thesis, I present a systematic study of image and video data mining, with the focus of two fundamental problems: (1) mining interesting and \textit{a priori} unknown patterns from image and video data, and (2) discovering discriminative visual patterns for classification. In terms of pattern mining, I present several data mining algorithms toward image and video
data. The advantage of these methods is that they can tolerate pattern variations in mining recurring or common patterns, such as non-uniform temporal scaling of video patterns, as well as spatial image patterns that exhibit large appearance variations because of rotation, scale changes, slight view changes, color variations and partial occlusions. By carefully designing the mining procedure and applying the locality sensitive hashing to speed up the similarity matching in high-dimensional feature space, our method is efficient. In terms of feature mining, I study how discovered patterns can be used for pattern classification. I derive the conditions that frequent patterns can be discriminative features of upper bounded empirical training error. A corresponding data mining-driven approach is proposed for searching those discriminative features.

I summarize the image and video data mining problems and my contribution as below.

**Recurrent event mining in videos**

To discover recurring temporal patterns from videos, we propose an efficient forest-grow algorithm in Chapter 4 which can handle non-uniform temporal scaling of video repetitions. As any temporal video segments or spatial image regions could be a candidate of interesting patterns, there is enormous computational cost involved in exploring the huge solution space. It involves extremely huge computational cost if performing exhaustive search on all possible candidates of recurring patterns at various locations and sizes. Moreover, unlike discrete data such as texts which is well-structured and owns much less ambiguity, visual patterns appearing in images and video have large uncertainty and exhibit complex spatial or temporal structure. Thus it prevents directly traditional data mining methods to video data. By representing a video dataset as a sequence of video primitives (e.g. key frames
or video segments), and build a matching trellis which reveals the similarity matching relations among all the video primitives, we translate the recurring event discovery problem into finding continuous branches in the matching trellis, where each discovered continuous path corresponds to an instance of a recurring event. Our forest-growing algorithm can search for such continuous branches by growing multiple trees simultaneously in the matching trellis. A branching factor $B$ is introduced to constrain the tree growing, which can accommodate temporal and spatial variations in mining video repetitions. By using locality-sensitive hashing (LSH) to find the approximate matches of video primitives, we can build the trellis efficiently.

**Common pattern discovery in images**

To discover common patterns from images, instead of representing images as visual documents based on “bag of words” model, we present a novel approach base on spatial random partition and fast word-free image matching in Chapter 2. Represented as a set of continuous visual primitives, each image is randomly partitioned many times to form a pool of subimages. Each subimage is queried and matched against the pool, and then common patterns can be localized by aggregating the set of matched subimages. The asymptotic property and the complexity of the proposed method are given in this paper, along with many real experiments. In comparison to the “bag of words” model, the performance of our method does not depend on the quality of the visual word dictionary. Both theoretical studies and experiment results show its advantages.

**Semantically meaningful pattern discovery in images**

As highly repetitive patterns may not be always informative or semantically meaningful, we modify the pattern discovery criterion from mining frequent patterns to semantically
meaningful patterns in Chapter 3. Unlike Chapter 3 which discover re-occurrences of the same visual object, we target on image collections containing objects belonging to the same category (e.g. faces or cars). It is a more challenging problem because no two objects in the dataset are identical, and we need to handle intra-class object variations. We present a principled solution to the discovery of meaningful itemsets based on frequent itemset mining and apply a self-supervised clustering scheme of the high-dimensional visual features by feeding back discovered patterns to tune the similarity measure through metric learning. To refine the discovered patterns, a pattern summarization method that deals with the measurement noises brought by the image data is proposed. The experimental results in the real images show that our method can discover semantically meaningful patterns efficiently and effectively.

**Compositional feature mining for pattern classification**

To justify the importance of mining frequent patterns from images, I investigate how to apply frequent patterns as discriminative features for pattern classification in Chapter 5. In that chapter, we present the condition when can frequent patterns be discriminative features. The upper-bounded empirical error of the discovered feature is derived. A data mining algorithm for discovering such discriminative features is presented. Finally, the discovered features are further combined through multi-class boosting. Experiments on the visual event recognition problem demonstrate the effectiveness of this approach.
CHAPTER 2

Mining Common Patterns from Image Data

It is of great interest to automatically discover common visual patterns (if any) in a set of unlabeled images. Recent research has suggested its applicability in many potential applications, such as content-based retrieval [2,3], image categorization [4], object discovery [5–7], recognition [8] and segmentation [5,9–12], image irregularity detection [13] and similarity measure [14].

Because no prior knowledge on the common patterns is provided, this task is very challenging, even for our human eyes. Let’s look at the example in Fig. 2.1. This is much more difficult than pattern detection and retrieval, because the set of candidates for possible common patterns is enormous. Validating a single candidate (which is equivalent to pattern detection) has been computationally demanding, and thus evaluating all these candidates will inevitably be prohibiting, if not impossible.

This difficulty may be alleviated by developing robust partial image matching methods [4,7,13], but this is not a trivial task. Another idea is to transform images into visual documents so as to take advantage of text-based data mining techniques [5,15,16]. These methods need to quantize continuous primitive visual features into discrete labels (i.e., “visual words”) through clustering. The matching of two image regions can be efficiently performed by comparing their visual-word histograms while ignoring their spatial configurations. Although these methods are efficient, their performances are largely influenced by the quality of the visual word dictionary. It is not uncommon that the dictionary includes visual synonyms
and polysemys that may significantly degrade the matching accuracy. In addition, since a large number of images is generally required to determine the dictionary, these methods may not be suitable if pattern discovery needs to be performed on a small number of images.

This chapter presents a novel approach to efficient pattern discovery based on spatial hashing. Each image is represented as a set of continuous visual primitives, and is randomly partitioned into subimages for a number of times. This leads to a pool of subimages for the set of images given. Each subimage is queried and matched against the subimage pool. As each image is partitioned many times, a common pattern is likely to be present in a good number of subimages across different images. The more matches a subimage query can find in the pool, the more likely it contains a common pattern. And then the pattern can be localized by aggregating these matched subimages. In addition, the proposed method for matching image regions is word-free as it is performed directly on the continuous visual primitives. An approximate solution is proposed to efficiently match two subimages by checking if they
share enough similar visual primitives. It is provably correct that such an approximation is the upper bound estimation of the optimal one.

This new method offers several advantages. (1) It does not depend on good image segmentation results. According to its asymptotic property, it is provably correct that the patterns can be recovered regardless of its scale, shape and location. (2) It can automatically discover multiple common patterns without knowing the total number \textit{a priori}, and is robust to rotation, scale changes and partial occlusion. The robustness of the method only depends on the matching of visual primitives. (3) It is word-free but still computationally efficient, because of the use of the locality sensitive hash (LSH) technique.

2.1. Proposed Approach

2.1.1. Algorithm overview

Given a number of $T$ unlabeled images, our objective is to discover common spatial patterns that appear in these images. Such common patterns can be identical objects or categories of objects. The basic idea of the proposed spatial hashing method is illustrated in Fig. 2.2.
We extract a set of visual primitives \( V_I = \{v_1, ..., v_m\} \) to characterize each image \( I \). Each visual primitive is described by \( v = \{x, y, \vec{f}\} \), where \((x, y)\) is its spatial location and \( \vec{f} \in \mathbb{R}^d \) is its visual feature vector. Collecting all these visual primitives, we build the visual primitive database \( D_v = V_{I_1} \cup V_{I_2} \cup ... \cup V_{I_T} \), whose size is denoted by \( N = |D_v| \), where \( T \) is the total number of images. To index visual primitives, each \( v \) is associated with a unique integer \( z \) \((1 \leq z \leq N)\) for retrieving it from \( D_v \). Our algorithm is summarized in Alg. 1.
**Algorithm 1: Spatial Hashing for Pattern Discovery**

**Input**: a collection of unlabeled images: \( D_I = \{ I_i \} \)

**Output**: a set of subimage regions that correspond to common spatial patterns

1. **Hashing**: \( \forall \) image \( I_i \in D_I \), randomly partition it into \( G \times H \) subimages for \( K \) times. This outputs the subimage database \( D_R \) (Sec. 2.1.2).

2. **Matching**: \( \forall \) subimage \( R_i \in D_R \), query it in \( D_R \). This leads to a small set of *popular* subimages that have enough matches in \( D_R \) (Sec. 2.1.3).

3. **Voting**: \( \forall I_i \in D_I \), vote the corresponding regions of the discovered popular subimages \( R \subset I_i \) and accumulate all the votes to form a voting map (Sec. 2.1.4)

4. **Localization**: \( \forall I_i \in D_I \), segment its voting map to localize the common patterns (Sec. 2.1.4)

### 2.1.2. Spatial hashing

For each image \( I_i \in D_I \), we randomly partition it into \( G \times H \) non-overlapping subimages \( \{ R_i \} \) and perform such partition \( K \) times independently. We end up with in total \( M = G \times H \times K \times T \) subimages and form a subimage database \( D_R = \{ R_i \}_{i=1}^M \). Each generated subimage is characterized by a “bag of visual primitives”: \( R = (V_R, C_R) \), where \( V_R \subset V_I \) denotes the set of visual primitives contained in \( R \) and \( C_R \) is the bounding box of \( R \). Under a certain partition \( k \in \{ 1, 2, \ldots, K \} \), the \( G \times H \) subimages are non-overlapping, and we have \( V_I = V_{R_1} \cup V_{R_2} \cup \ldots \cup V_{R_{G \times H}} \). However, subimages generated from different partitions possibly overlap.

Under each partition, we are concerned on whether there exists a *good subimage* that contains the common pattern \( P \). This depends on if pattern \( P \) is broken under this partition.
Without losing generality, we assume that the pattern $\mathcal{P}$ appears at most once in each image. Supposing the spatial size of the image $\mathcal{I}$ is $(I_x, I_y)$ and the bounding box of the common pattern $\mathcal{P}$ is $C_\mathcal{P} = (P_x, P_y)$, we calculate the non-broken probability for $\mathcal{P}$ as the probability that none of the $(G - 1) + (H - 1)$ partition lines penetrates the pattern $\mathcal{P}$:

$$p = (1 - \frac{P_x}{I_x})^{G-1}(1 - \frac{P_y}{I_y})^{H-1}. \quad (2.1)$$

Given a partition of an image, we can find at most one good subimage with probability $p$, if the image contains no more than one such pattern. For instance in Fig. 2.2, there are in total 7 good subimages and 2 other ones are missed.
2.1.3. Matching and discovering popular subimages

The objective is to match subimages pair-wisely and discover “popular” ones from the pool $\mathbf{D}_R$. Here a popular subimage is the one that contains a common pattern $\mathcal{P}$ and has enough matches in $\mathbf{D}_R$.

subimage matching

Unlike the “bag of words” method, we cannot match subimages through “histogram of words” since our visual primitives are not quantized into words. Instead, $\forall \mathcal{R}, \mathcal{Q} \in \mathbf{D}_R$, we measure the similarity by matching their visual primitives $\mathbf{V}_R$ and $\mathbf{V}_Q$ directly. Matching can be formulated as an assignment problem:

$$\text{Sim}(\mathbf{V}_R, \mathbf{V}_Q) = \max \sum_{i=1}^{\mid \mathbf{V}_R \mid} s(v_i, \mathcal{F}(v_i)),$$

(2.2)

where $\mathcal{F}(\cdot)$ is the assignment function $\mathcal{F} : \mathbf{V}_R \rightarrow \mathbf{V}_Q$, i.e., for each $v_i \in \mathbf{V}_R$, $\mathcal{F}$ assigns its matching $u_j = \mathcal{F}(v_i) \in \mathbf{V}_Q$. $s(v_i, \mathcal{F}(v_i))$ is the similarity measure between $v_i$ and its assignment $u_j$. Two subimages are matched if their similarity $\text{Sim}(\mathbf{V}_R, \mathbf{V}_Q) \geq \lambda$, where $\lambda > 0$ is the subimage matching threshold. Generally, it is non-trivial and computationally demanding to solve this assignment problem when exclusive matching is required.

In this chapter, we present an approximate solution to this problem with a linear complexity. Firstly, we perform a pre-processing step on the visual primitive database $\mathbf{D}_v$. This is the overhead of our pattern discovery method. For each $v \in \mathbf{D}_v$, we perform the $\epsilon$-Nearest Neighbors ($\epsilon$-NN) query and define the retrieved $\epsilon$-NN set of $v$ as its match-set $\mathcal{M}_v = \{u \in \mathbf{D}_v : \|f_v - f_u\| \leq \epsilon\}$. In order to reduce the computational cost in finding $\mathcal{M}_v$ for each $v$, we apply Locality Sensitive Hashing (LSH) [17] that performs efficient $\epsilon$-NN queries.
Figure 2.3. Similarity measure of two visual primitives $s(v, u)$, where $a, b, c, d, e, f$ denote visual primitives. We notice $s(v, u) = 0$ when $v \notin M_u$ and $u \notin M_v$.

After obtaining all the match-sets, $\forall v, u \in D_v$, we define their similarity measure $s(v, u)$ as:

$$s(v, u) = \begin{cases} 
\exp \frac{-\|\vec{f}_v - \vec{f}_u\|^2}{a}, & \text{if } v \in M_u \\
0, & \text{otherwise}
\end{cases}$$  \hspace{1cm} (2.3)

where $\alpha > 0$ is a parameter and $M_u$ depends on the threshold $\epsilon$. $s(v, u)$ is a symmetric measure as $v \in M_u \iff u \in M_v$. This visual primitive matching is illustrated in Fig. 2.3.

Now suppose that $V_R = \{v_1, v_2, ..., v_m\}$ and $V_Q = \{u_1, u_2, ..., u_n\}$ are two sets of visual primitives. We can approximate the match between $V_R$ and $V_Q$ in Eq. 2.4, by evaluating the size of the intersection between $V_R$ and the match-set of $V_Q$:

$$\widetilde{\text{Sim}}(V_R, V_Q) \triangleq |V_R \cap M_{V_Q}|$$  \hspace{1cm} (2.4)

$$\geq \max_{\mathcal{F}} \sum_{i=1}^{\|V_R\|} s(v_i, \mathcal{F}(v_i))$$  \hspace{1cm} (2.5)

$$= \text{Sim}(V_R, V_Q),$$  \hspace{1cm} (2.6)

where $\widetilde{\text{Sim}}(V_R, V_Q)$ is a positive integer; $M_{V_Q} = M_{u_1} \cup M_{u_2} \ldots \cup M_{u_n}$ denotes the match-set of $V_Q$. We apply the property that $0 \leq s(v, u) \leq 1$ to prove Eq. 2.5. As shown in
Fig. 2.4, $\tilde{\text{Sim}}(V_R, V_Q)$ can be viewed as the approximate flow between $V_R$ and $V_Q$. Based on the approximate similarity score, two subimages are matched if $\tilde{\text{Sim}}(V_R, V_Q) \geq \lambda$. Since we always have $\tilde{\text{Sim}}(V_R, V_Q) \geq \text{Sim}(V_R, V_Q)$, the approximate similarity score is a safe bounded estimation and it never discards qualified matches. The intersection of two sets $V_R$ and $M_{V_Q}$ can be performed in a linear time $O(|V_R| + |M_{V_Q}|) = O(m + nc)$, where $c$ is the average size of the match-set for all $v$. Since $m \approx n$, the complexity is essentially $O(mc)$.

Finding popular subimages

Based on the matching defined above, we are ready to find popular subimages. Firstly, we denote $G_R \subset D_R$ as the set of good subimages which contain the common pattern $P$: $\forall R_g \in G_R$, we have $P \subset R_g$. A good subimage becomes a popular subimage if it has enough matches in the pool $D_R$. As we do not allow $R_g$ to match subimages in the same image as
$\mathcal{R}_g$, its *popularity* is defined as the number of good subimages in the rest of $(T - 1) \times K$ partitions. As each partition $k$ can generate one good subimage with probability $p$ (Eq. 2.1), the total matches $\mathcal{R}_g$ can find is a binomial random variable: $Y_{\mathcal{R}_g} \sim B(K(T - 1), p)$, where $p$ depends on the partition parameters and the shape of the common pattern (Eq. 2.1). The more matches $\mathcal{R}_g$ can find in $D_R$, the more likely that $\mathcal{R}_g$ contains a common pattern and more significant it is. On the other hand, unpopular $\mathcal{R}$ may not contain any common spatial pattern as it cannot find supports from other subimages.

Based on the expectation of matches that a good subimage can find, we apply the following truncated 3-$\sigma$ criterion to determine the threshold for the popularity:

$$\tau = \mu - 3\sigma = (T - 1)kp - 3\sqrt{(T - 1)kp(1 - p)}, \quad (2.7)$$

where $\mu = E(Y_{\mathcal{R}_g}) = (T - 1)kp$ is the expectation of $Y_{\mathcal{R}_g}$ and $\sigma^2 = Var(Y_{\mathcal{R}_g}) = (T - 1)kp(1 - p)$ is the variance of $Y_{\mathcal{R}_g}$. For every subimage $\mathcal{R} \in D_R$, we query it in $D_R \setminus I_t$ to check its popularity, where $I_t$ is the image that generates $\mathcal{R}$. If $\mathcal{R}$ can find at least $\lceil \tau \rceil$ matches, it is a popular one.

### 2.1.4. Voting and locating common patterns

After discovering all the popular subimages (denoted by set $S_R \subset G_R$), they vote for the common patterns. For each image, we select all popular subimages that are associated with this image. Aggregating these popular subimages must produce overlapped regions where common patterns are located. A densely overlapped region is thus the most likely location for a potential common pattern $\mathcal{P}$. Each popular subimage votes its corresponding pattern in a *voting map* associated with this image.
Since we perform the spatial hashing $K$ times for each image, each pixel $l \in I$ has up to $K$ chances to be voted, from its $K$ corresponding subimages $R^k_l (k = 1, \ldots, K)$ that contains $l$. The more votes a pixel receives, the more probable that it is located inside a common pattern. More formally, for the common pattern pixel $i \in P$, the probability it can receive a vote under a certain random partition $k \in \{1, 2, \ldots, K\}$ is:

$$Pr(x^k_i = 1) = Pr(R^k_i \in S_R)$$

$$= Pr(R^k_i \in G_R) Pr(vote(R^k_i) \geq \lceil \tau \rceil | R^k_i \in G_R)$$

$$= pq,$$  \hspace{1cm} (2.8)

where the superscript $k$ indexes the partition and the subscript $i$ indexes the pixel; $R^k_i$ is the subimage that contains $i$; $p$ is the prior that $i$ is located in a good subimage, i.e. $Pr(R^k_i \in G_R)$, the non-broken probability of $P$ under a partition (Eq. 2.1); $q$ is the likelihood that a good subimage $R^k_i$ is also a popular one, which depends on the number of matches $R^k_i$ can find. Specifically, under our popular subimage discovery criterion in Eq. 2.7, $q$ is a constant. Given a pixel $i$, \{${x^k_i, k = 1, 2, \ldots, K}$\} is a set of independent and identically distributed (i.i.d.) Bernoulli random variables. Aggregating them together, the votes that $i \in P$ can receive is a binomial random variable $X^K_i = \sum_{k=1}^{K} x^k_i \sim B(K, pq)$. Thus we can determine the common pattern regions based on the number of votes they receive.

Under each partition $k$, $P$ is voted by the popular subimage $R^k_P \in S_R$. Since $R^k_P$ contains $P$, it actually gives an estimation of the location for $P$. However, a larger size of $R^k_P$ implies more uncertainty it has in locating $P$ and thus its vote should take less credit. We thus adjust the weight of the vote based on the size of $R^k_P$. \forall \ i \in P, \text{ we weight the votes:}
where \( w_k^i > 0 \) is the weight of the \( k_{th} \) vote. Among the many possible choices, in this chapter we set \( w_k^i = \frac{\text{area}(I)}{\text{area}(R_{ki}^i)} \), meaning the importance of the popular subimage \( R_{ki}^i \). The larger the \( \text{area}(R_{ki}^i) \), the smaller weight its vote counts. Sec. 2.2.1 will discuss the criteria and principle in selecting a suitable \( w_k^i \). Finally, we can roughly segment the common patterns given the voting map, based on the expected number of votes a common pattern pixel should receive. This rough segmentation can be easily refined by combining it with many existing image segmentation schemes, such as the level set based approach.

2.2. Properties of the Algorithm

2.2.1. Asymptotic property

The correctness of our spatial hashing and voting strategy is based on the following theorem that gives the asymptotic property.

\textbf{Theorem 1. Asymptotic property}

\textit{We consider two pixels } \( i, j \in \mathcal{I} \), where \( i \in \mathcal{P} \subset \mathcal{I} \) is located inside one common pattern \( \mathcal{P} \) while \( j \notin \mathcal{P} \) is located outside any common patterns (e.g. in the background). Suppose } \( X_i^K \) \textit{and } \( X_j^K \) \textit{are the votes for } \( i \) \textit{and } \( j \) \textit{respectively, considering } \( K \) \textit{times random partitions. Both } \( X_i^K \) \textit{and } \( X_j^K \) \textit{are discrete random variables, and we have:}

\[
\lim_{K \to \infty} P(X_i^K > X_j^K) = 1.
\] (2.10)
The above theorem states that when we have enough times of partitions for each image, a common pattern region $\mathcal{P}$ must receive more votes, so that it can be easily discovered and located. The proof of Theorem 1 is given in the Appendix. We briefly explain its idea below.

**Explanation of Theorem 1**

We consider two pixels $i \in \mathcal{P}$ and $j \notin \mathcal{P}$ as stated in Theorem 1. We are going to check the total number of votes that $i$ and $j$ will receive after $K$ times partitions of $I$.

![Figure 2.5. Illustration of the EVR. The figures show two different random partitions on the same image. The small orange rectangle represents the common pattern $\mathcal{P}$. We compare two pixels $i \in \mathcal{P}$ and $j \notin \mathcal{P}$. The large blue region represents $\mathcal{R}'_j$, the EVR of $j$; while $\mathcal{R}'_i = \mathcal{P}$. In the left figure, $\mathcal{R}'_j$ is broken during the partition while $\mathcal{R}'_i$ is not. Thus $i$ get a vote because $\mathcal{R}_4$ (shadow region) is a popular subimage and the whole region is voted; while $j$ does not receive the vote. In the right image, both $\mathcal{R}'_i$ and $\mathcal{R}'_j$ are broken during the partition, so neither $i$ and $j$ is voted as no popular subimage appears.](image-url)
For each pixel \( l \in I \), we define its Effective Vote Region (EVR) as:

\[
\mathcal{R}_l' = \arg \min_{R} \text{area}(R| \mathcal{P} \subseteq R, l \in \mathcal{R}),
\]

where \( \mathcal{R} \) is a rectangle image region that contains both the common pattern \( \mathcal{P} \) and the pixel \( l \). Fig. 2.5 illustrates the concept of EVR. Based on the definition, both EVR \( \mathcal{R}_i' \) and \( \mathcal{R}_j' \) contain \( \mathcal{P} \). For the “positive” pixel \( i \in \mathcal{P} \), we have: \( \mathcal{R}_i' = \mathcal{P} \). On the other hand, for the “negative” pixel \( j \notin \mathcal{P} \), it corresponds to a larger EVR \( \mathcal{R}_j' \), and we have \( \mathcal{R}_i' \subseteq \mathcal{R}_j' \). Like pixel \( i \), whether \( j \notin \mathcal{P} \) can get a vote depends on whether its subimage \( \mathcal{R}_j^k \) is a popular one. Suppose the spatial size of the EVR \( \mathcal{R}_j' \) is \((B_x, B_y)\). Similar to Eq. 2.1, the non-broken probability of \( \mathcal{R}_j' \) is:

\[
p_j = (1 - \frac{R_x}{I_x})^{G-1}(1 - \frac{R_y}{I_y})^{H-1}.
\]

Following the same analysis in Eq. 2.8, \( x_j^k \) is a Bernoulli random variable:

\[
Pr(x_j^k) = \begin{cases} 
    p_jq, & x_j^k = 1, \\
    1 - p_jq, & x_j^k = 0,
\end{cases}
\]

where \( q \) is the likelihood of the good subimage being a popular one, which is a constant unrelated with \( p_j \) (Eq. 2.7). Thus whether a pixel \( j \notin \mathcal{P} \) can receive a vote depends on the size of its EVR. When considering \( K \) times random partitions, the total number of votes for pixel \( j \notin \mathcal{P} \) is also a binomial random variable \( X_j = \sum_{k=1}^{K} x_j^k \sim B(K, p_jq) \).

Since \( \mathcal{R}_i' \subseteq \mathcal{R}_j' \), we have \( B_x > P_x \) and \( B_y > P_y \). It is easy to see \( p_i > p_j \) by comparing Eq. 2.1 and 2.12. When we consider the unweighted voting (i.e. \( w_i^k = w_j^k = 1 \)), \( i \) is expected to receive more votes than \( j \) because \( E(X_i^K) = p_iqK > E(X_j^K) = p_jqK \). In the case of the weighted voting, we can estimate the expectation of \( X_i^K \) as:
\[ E(\mathbf{X}_i^K) = \sum_{k=1}^{K} E(w_i^k x_i^k) = \sum_{k=1}^{K} E(w_i^k) E(x_i^k) \]  
\[ = \sum_{k=1}^{K} pq E(w_i^k) = pq K E(w_i), \]  
(2.14)

(2.15)

where we assume \( w_i^k \) be independent to \( x_i^k \) and \( E(w_i) \) is only related to the average size of the popular subimage. Therefore to prove Theorem 1, we need to guarantee that \( E(\mathbf{X}_i^K) = p_i q K E(w_i) > p_j q K E(w_j) = E(\mathbf{X}_j^K) \). It follows that we need to select suitable weighting strategy such that \( p_i E(w_i) > p_j E(w_j) \). A possible choice is given in Sec. 2.1.4.

It is worth mentioning that the expected number of votes \( E(\mathbf{X}_i^K) = p_i q K E(w_i) \) depends on the spatial hashing scheme \( G \times H \times K \), where \( p_i \) depends on \( G \) and \( H \) (Eq. 2.1), \( q \) depends on both \( p \) and \( K \) (Eq. 2.7), and \( w_i \) depends on \( G \) and \( H \) as well. Our method does not need the prior knowledge of the pattern, but knowing the shape of the pattern can help choose better \( G \) and \( H \), which leads to faster convergence (Theorem 1). A larger \( K \) results in more accurate patterns but needs more computation. In general, \( G \) and \( H \) are best selected to match the spatial shape of the hidden common pattern \( \mathcal{P} \) and the larger the \( K \), the more accurate our approximation is but more computation is required.

### 2.2.2. Computational complexity analysis

Let \( M = |\mathbf{D}_R| = G \times H \times K \times T \) denote the size of the subimage database \( \mathbf{D}_R \). In general, \( M \) is much less than \( N = |\mathbf{D}_v| \), the total number of visual primitives, when selecting hashing parameters suitably. Because we need to evaluate \( \binom{M}{2} \) pairs, the complexity for discovering popular subimages in \( \mathbf{D}_R \) is \( O(M^2(mc)) \), where \( mc \) is the cost for matching two sets of visual primitives \( m = |\mathbf{V}_R| \) and \( n = |\mathbf{V}_Q| \) as analyzed before, where \( c \) is a constant. The overhead of our approach is to find the \( \mathcal{M}_v \) for each \( v \in \mathbf{D}_v \) (formation of \( \mathbf{D}_R \) is of a linear complexity
By applying LSH, each query complexity can be reduced from $O(dN)$ to less than $O(dN^{\frac{1}{a}})$ where $a > 1$ is the approximation factor \[17\] and $d$ is the feature dimension.

<table>
<thead>
<tr>
<th>Method</th>
<th>overhead</th>
<th>matching</th>
<th>discovering</th>
</tr>
</thead>
<tbody>
<tr>
<td>[15]</td>
<td>$O(dN^{ki})$</td>
<td>$O(k)$</td>
<td>$O(N^{2k})$</td>
</tr>
<tr>
<td>[13]</td>
<td>none</td>
<td>$O(Nd + mb)$</td>
<td>$O(N(Nd + mb))$</td>
</tr>
<tr>
<td>Ours</td>
<td>$O(dN^{1+\frac{1}{a}})$</td>
<td>$O(mc)$</td>
<td>$O(M^2mc)$</td>
</tr>
</tbody>
</table>

Table 2.1. Computational complexity comparison

We further compare the computational complexity of our method to two existing methods \[13\] \[15\] in Table 2.1. The overhead of \[15\] comes from clustering visual primitives into $k$ types of words through the $k$-means algorithm, and $i$ is the number of iterations. We estimate the discovering complexity of \[13\] by assuming that there are in total $O(N)$ number of queries for evaluation, each time applying the fast inference algorithm proposed in \[13\], where $b$ is the constant parameter. It is clear that our method is computationally more efficient as $M << N$. In the example shown in Fig. 2.1, we detect $N = 3416$ visual primitives for two images. When performing a hashing scheme as $G \times H \times K = 4 \times 3 \times 30$, we generate $M = 360 \times 2$ subimages. The CPU cost of the overhead of performing LSH is around 7.6 seconds (without parameter optimization), and the CPU cost of pattern discovery is around 4.7 seconds where the average cost of each subimage matching is less than 0.1 millisecond.

### 2.3. Experiments

We apply Scale Invariant Features (SIFT) \[18\] as the visual primitives although other local invariant features are certainly possible. Each SIFT descriptor $f_v$ is a 128-d vector which characterizes the local invariance of a visual primitive $v$. We set $\epsilon = 200$ in the LSH-based $\epsilon$-NN query; the subimage matching threshold is $\lambda = 40$. All the images are of size
640 × 480 and each image can generate 1000 to 3000 visual primitives. Without specific indication, we apply the spatial hashing scheme \( G \times H \times K = 4 \times 3 \times 50 \), where \( G \times H \) is selected according to the aspect ratio of each image. All the experiments are performed on a standard P4 @ 3.19G Hz PC (1 G memory). The algorithm is implemented in C++.

### 2.3.1. Common pattern discovery

The results of the pattern discovery on the two images in Fig. 2.1 are presented in Fig. 2.6. As each voting subimage is a rectangle, the voting map is a union of weighted rectangles. The brightness of the regions indicates the total votes each pixel receives, i.e. the likelihood belonging to a common pattern. At the moment, pattern boundary is not accurate enough as a very naive fixed threshold \( E(X^k) \) (Eq. 2.15) is applied. Depending on applications, accurate segmentation may not matter much. But if required, the boundaries can be further improved by incorporating other image segmentation techniques.

To evaluate our algorithm, we use 8 image datasets, where each dataset contains 1 to 3 different common patterns and each common pattern has 2 to 4 instances within the dataset. Each dataset contains at least 4 images where some of them may not contain any common pattern. Around half of the images have multiple common patterns. The instances of each common pattern exhibit possible variations like rotation, partial occlusion, scale and slightly viewpoint changes and are located in clutter backgrounds. Given an image dataset, we evaluate the performance by checking how accurate the regions containing the common patterns are recovered. Let \( R \) and \( GT \) be respectively the discovered common pattern regions and the bounding boxes of ground truth patterns. The performance is evaluated by using two criteria: hit ratio \( h_r = \frac{|GT \cap R|}{|GT|} \) and the background ratio \( b_r = \frac{|R - (R \cap GT)|}{|R|} \). Table 2.2 presents the performance. Since common patterns are of different shapes and
Figure 2.6. The input is from Fig. 2.1. The 1st and the 2nd row show the voting maps and the rough segmentations respectively. The three common patterns are listed from the 3rd to the 5th row. Besides variations like rotation and scale change, the second poster suffers from partial occlusion in the left image. The hit ratio and the background ratio are $h_r = 0.76$ and $b_r = 0.29$ respectively.

sizes, it is unfair to apply a unique non-broken probability $p$ for all image dataset. For each image dataset, the best result is reported by searching optimal $p$ from 0.15 to 0.30.

We describe our datasets as: $Dataset_A = \{Fedex\}$, $Dataset_B = \{Stop Sign\}$, $Dataset_C = \{No Parking Sign\}$, $Dataset_D = \{Tea Box\}$, $Dataset_E = \{Tag\}$, $Dataset_F = \{Book, Tea Box and Tag\}$, $Dataset_G = \{Books\}$, $Dataset_H = \{Posters\}$ (see supplemental materials for details).

Another example of pattern discovery in multiple images is presented in Fig. 2.7. Three of the four input images contain the common pattern: “no parking sign”, and the fourth
Figure 2.7. The input is a set of four images (the 1st column). Each row corresponds to an image. The common pattern “no parking sign” appears in the first three images. A comparison of applying the different partition times $K$ is shown in the 2nd ($K = 25$), 4th ($K = 50$) and 6th ($K = 75$) columns. The 3rd, 5th and 7th columns are voting maps associated with the corresponding images. More results can be reviewed at the supplemental materials.

<table>
<thead>
<tr>
<th>Dataset</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
<th>average</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hit Ratio</td>
<td>0.95</td>
<td>0.98</td>
<td>0.92</td>
<td>0.45</td>
<td>0.91</td>
<td>0.84</td>
</tr>
<tr>
<td>Bk Ratio</td>
<td>0.41</td>
<td>0.55</td>
<td>0.45</td>
<td>0.15</td>
<td>0.36</td>
<td>0.38</td>
</tr>
<tr>
<td>Dataset</td>
<td>$F^*$</td>
<td>$G^*$</td>
<td>$H^*$</td>
<td>average</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Hit Ratio</td>
<td>0.80</td>
<td>0.85</td>
<td>0.76</td>
<td>0.80</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Bk Ratio</td>
<td>0.53</td>
<td>0.37</td>
<td>0.29</td>
<td>0.40</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2.2. Performance evaluation. The superscript $*$ denotes the dataset containing multiple common patterns. See text for details.

image does not. We compare results under various partition times $K$ ($K=25,50,75$). One false alarm appears when $K = 25$. This is caused by fake popular subimages. However when a larger $K$ is selected, these fake popular subimages can be rejected as the threshold for popularity increases with $K$. In general, larger $K$ produces more accurate localization of the patterns, which supports Theorem 1.
2.3.2. Image irregularity detection

The spatial hashing method can also be applied for single image irregularity detection. Firstly, we perform the \( G \times H \times K \) spatial hashing on the input image to generate a subimages database. Secondly, for each generated subimage, we match it with all others. In contrast to common pattern discovery, we select the \textit{unpopular} subimages instead of the popular ones for irregularity detection. The subimage matching is performed based on Eq. 2.4, but a lower threshold is applied. Therefore common subimages are more easily to be matched. If there exists an irregular pattern in the image, then a subimage containing this irregular pattern is very likely to be an \textit{unpopular} one as it cannot match other subimages. After discovering all unpopular subimages, they vote for the irregular pattern and build the accumulated voting map similar to the discussion in Sec. 2.1.4. The only difference lies in the weighting strategy of voting. Now we weight the vote in proportion to the size of the unpopular subimage, i.e. \( w_i^k \propto area(R_i^k) \), because a larger unpopular subimage is more likely to include an irregular pattern. Fig. 2.8 shows an example of irregularity detection. We perform \( K = 75 \) partitions to obtain a good boundary. In order to avoid detecting the blank background, the subimages that contain very few visual primitives are ignored.

![Input Example of an Unpopular Subimage Voting Map (K=75) Irregularity Detection](image1)

**Figure 2.8.** Spatial hashing for image irregularity detection. An instance of the spatial hashing and the unpopular subimage are shown in the 2\textsubscript{nd} image. The subimage that contain green circles (denoting visual primitives) is the \textit{unpopular} one. After segmenting the voting map (the 3\textsubscript{rd} image), we obtain the final irregularity detection result (the 4\textsubscript{th} image).
2.4. Conclusion

We present a novel method based on spatial hashing for common pattern discovery and image irregularity detection. Our method is robust to various pattern variations. Although no word dictionary is needed, our method is still efficient because it employs LSH for fast $\epsilon$-NN query and uses the bounded approximation for subimage matching. The asymptotic property of the proposed algorithm provides a theoretical guarantee for its performance. Partition times $K$ trades-off between the accuracy of the localization and the speed. Although only SIFT features are used in the experiments reported, our method is generally applicable in using many other types of visual primitives (e.g. over-segment regions) and features (e.g. color histograms). The only requirement is that two subimages should be matched when they share a common pattern.
CHAPTER 3

Mining Semantic Meaningful Patterns from Image Data

3.1. Introduction

Meaningful patterns can be those that appear frequently, thus an important task for data mining and pattern discovery is to identify repetitive patterns. Frequent itemset mining (FIM) and its extensions [19][20][21] have been extensively studied. However, a highly repetitive pattern may not be informative or semantically meaningful. Therefore a more important task is to extract informative and potentially interesting patterns (e.g. semantically meaningful patterns) in possibly noisy data. This can be done by mining meaningful patterns either through post-processing the FIM results or proposing new data mining criteria, including mining compressed patterns [22][23][24], approximate patterns [25][26][27] and pattern summarization [28][29][30]. These data mining techniques may discover meaningful frequent itemsets and represent them in a compact way.

Such research in structured data (e.g., transaction data) and semi-structured data (e.g., text) has aroused our curiosity in finding meaningful patterns in non-structured multimedia data like images and videos [31][15][32][6]. For example, once we can extract some invariant visual primitives such as interest points [18] or salient regions [33] from the images, we can represent each image as a collection of such visual primitives characterized by high-dimensional feature vectors. By further quantizing those visual primitives to discrete “visual items” through clustering the high-dimensional features [15][16], each image is represented by a set of transaction records, with each transaction corresponds to a local image patch.
and describes its composition of visual primitive classes (items). After that, data mining techniques like FIM can be applied to such a transaction database induced from images for discovering meaningful visual patterns.

Although this idea appears to be quite exciting, the leap from transaction data to images is not trivial, because of two fundamental differences between them. Above all, unlike transaction and text data that are composed of discrete elements without ambiguity (i.e. predefined items and vocabularies), visual patterns generally exhibit large variabilities in their visual appearances. A same visual pattern may look very different under different views, scales, lighting conditions, not to mention partial occlusion. It is very difficult, if not possible, to obtain invariant visual features that are insensitive to these variations such that they can uniquely characterize visual primitives. Therefore although a discrete item codebook can be forcefully obtained by clustering high-dimensional visual features (e.g., by vector quantization [34] or $k$-means clustering [15]), such “visual items” tend to be much more ambiguous than the case of transaction and text data. Such imperfect clustering of visual items brings large challenges when directly applying traditional data mining methods into image data.

In addition to the continuous high-dimensional features, visual patterns have more complex structure than transaction and text pattern. The difficulty of representing and discovering spatial patterns in images prevents straightforward generalization of traditional frequent pattern mining methods that are applicable for transaction data. For example, unlike traditional transaction database where records are independent of each other, the induced transactions generated by image patches can be correlated due to spatial dependency. Although there exist methods [35] [36] [37] for spatial collocation pattern discovery from
geo-spatial data, they cannot be directly applied to image data which are characterized by high-dimensional features. Moreover, the spatial co-occurrences of the items do not necessarily indicate the real associations among them, because a frequent spatial collocation pattern can be generated by the self-repetitive texture in the image and thus is not semantically meaningful. Thus, finding frequent patterns based on FIM may not always output meaningful and informative patterns in the image data.

Given a collection of unlabeled images, the objective of image data mining is to discover (if there is any) semantically meaningful spatial patterns that appear repetitively among the images. For example, given a set of images each of which contains an identical object (e.g. a book or a logo) but with possibly different locations, scales and views, the task is to efficiently discover and locate them in the images. This is a challenging problem because we have no prior knowledge of the object’s size, location and pose, or whether such object exists at all. Some existing methods based on graph matching are computational demanding and the solution is prone to local minimum \cite{7} \cite{6}. Thus more efficient and robust algorithm is desirable. In this chapter, we aim at an even more challenging problem: given a category of images, for example each image contains a frontal face but from different persons, we expect to discover some meaningful patterns like eyes and noses that have semantic meanings and can well interpret the face category. To this end, the following three issues need to be further addressed.

- **Spatial dependency of visual primitives.** To discover frequent patterns in image data using FIM, we can induce a transaction database where each transaction consists of a set of visual items charactering a local image region. However, these induced transactions are not independent as the local patches have spatial overlaps in images. This phenomenon complicates the data mining process for spatial data,
because simply counting the occurrence frequencies is doubtable and a frequent pattern is not necessarily a meaningful pattern. Thus special care needs to be taken;

• **Ambiguities in visual items.** The unsupervised clustering of visual primitives is not perfect. A same visual item may convey different semantic meanings. Taking a circle-like visual primitive for example, it can represent a human eye or a car wheel under different context. Thus it brings ambiguities when discovering meaningful patterns. The polysemy word phenomena in text data also appears in images.

• **Incomplete patterns.** There are two kinds of imperfections when translating the image data into transaction data. First of all, the visual primitives can be miss detected in the feature extraction process, due to occlusion of the visual primitive, bad lighting condition or the unreliable detector. Secondly, even a visual primitive is extracted, it can be wrongly labeled into a visual item because of quantization error. These two types of errors will be reflected in the induced transaction database. Performing FIM in the noisy transaction database brings a big obstacle for recovering semantic patterns. For example, a semantically meaningful pattern may be split into a lot of incomplete sub-patterns.

This chapter presents a novel approach to discovering semantically meaningful visual patterns from images. By addressing the above three difficulties, our contributions are three-fold:

• **new criteria for meaningful itemset discovery.** The co-occurrence frequency is no longer a sufficient condition for the meaningful collocation patterns in images. A
more plausible meaningful itemset mining based on likelihood ratio test and traditional FIM is proposed to evaluate the significance of a visual itemset;

- **self-supervised refinement of visual items.** To reduce the ambiguities in visual items, a top-down refinement is proposed by taking advantage of the discovered visual patterns. They serve as self-supervision to tune the metric in the high-dimensional feature space of visual primitives for better visual item clustering.

- **pattern summarization.** To handle the possible imperfections from the image data, a pattern summarization method using normalized cut is proposed to further cluster these incomplete and synonymous meaningful itemsets into semantically-coherent patterns;

![Figure 3.1. Overview for meaningful visual pattern discovery.](image)

### 3.1.1. Notations and basic concepts

Each image in the database is described as a set of visual primitives: $\mathcal{I} = \{v_i = (\vec{f}_i, x_i, y_i)\}$, where $\vec{f}_i$ denotes the high-dimensional feature and $\{x_i, y_i\}$ denotes the spatial location of $v_i$ in the image. For each visual primitive $v_i \in \mathcal{I}$, its local spatial neighbors form a group $\mathcal{G}_i = \{v_i, v_{i_1}, v_{i_2}, \ldots, v_{i_K}\}$. For example, $\mathcal{G}_i$ can be the spatial K-nearest neighbors (K-NN) or $\epsilon$-nearest neighbors of $v_i$ ($\epsilon$-NN) under Euclidean distance. The image database $\mathbf{D}_\mathcal{I} = \{\mathcal{I}_t\}_{t=1}^T$ can generate a collection of such groups, where each group $\mathcal{G}_i$ is associated to
a visual primitive $v_i$. By further quantizing all the high-dimensional features $\vec{f}_i \in \mathbf{D}_I$ into $M$ classes through $k$-means clustering, a codebook $\Omega$ can be obtained. We call every prototype $W_k$ in the codebook $\Omega = \{W_1,...,W_M\}$ a visual item. Because each visual primitive is uniquely assigned to one of the visual items $W_i$, the group $G_i$ can be transferred into a transaction $T_i$. More formally, given the group dataset $G = \{G_i\}_{i=1}^N$ generated from $\mathbf{D}_I$ and the visual item codebook $\Omega$ ($|\Omega| = M$), the induced transaction database $\mathbf{T}$ is defined as follows.

**Definition 1. Induced Transaction Database**

The induced transaction database $\mathbf{T} = \{T_i\}_{i=1}^N$ contains a collection of $N$ transactions with $M$ visual items. A sparse binary matrix $X_{N \times M}$ can represent $\mathbf{T}$, where $x_{ij} = 1$ denotes the $i$th transaction contains the $j$th visual item in the codebook and $x_{ij} = 0$ otherwise.

Such an induced transaction database is essentially based on the centric reference feature model for mining association rules [36], although collocation pattern models like [35] are also feasible in our approach. Given the visual item codebook $\Omega$, a set $\mathcal{P} \subset \Omega$ is called a visual itemset (itemset for short). For a given itemset $\mathcal{P}$, the transaction $T_i$ which includes $\mathcal{P}$ is called an occurrence of $\mathcal{P}$, i.e. $T_i$ is an occurrence of $\mathcal{P}$, if $\mathcal{P} \subseteq T_i$. Let $T(\mathcal{P})$ denote the set of all the occurrences of $\mathcal{P}$ in $\mathbf{T}$, and the frequency of $\mathcal{P}$ is denoted as:

$$frq(\mathcal{P}) = |T(\mathcal{P})| = |\{i : \forall j \in \mathcal{P}, x_{ij} = 1\}|.$$

(3.1)

For a given threshold $\theta$, called a minimum support, itemset $\mathcal{P}$ is frequent if $frq(\mathcal{P}) > \theta$. If an itemset $\mathcal{P}$ appears frequently, then all of its sub-sets $\mathcal{P}' \subset \mathcal{P}$ will also appear frequently, i.e. $frq(\mathcal{P}) > \theta \Rightarrow frq(\mathcal{P}') > \theta$. To eliminate this redundancy, we tend to discover closed frequent itemsets [38]. The number of closed frequent itemsets can be much less than the
frequent itemsets, and they compress information of frequent itemsets in a lossless form, i.e. the full list of frequent itemsets $F = \{P_i\}$ and their corresponding frequency counts can be exactly recovered from the compressed representation of closed frequent itemsets. Thus this guarantees that no meaningful itemsets will be left out through FIM. The closed frequent itemset is defined as follows.

**Definition 2. closed frequent itemset**

If for an itemset $P$, there is no other itemset $Q \supseteq P$ that can satisfy $T(P) = T(Q)$, we say $P$ is closed. For any itemset $P$ and $Q$, $T(P \cup Q) = T(P) \cap T(Q)$, and if $P \subseteq Q$ then $T(Q) \subseteq T(P)$.

In this chapter we apply the modified FP-growth algorithm [39] to implement the closed FIM. As FP-tree has a prefix-tree structure and can store compressed information of frequent itemset, it can quickly discover all the closed frequent sets from transaction dataset $T$.

**3.1.2. Overview of our method**

We present the overview of our visual pattern discovery method in Fig. 3.1. In Sec. 3.2, we present our new criteria for discovering meaningful itemsets $\Psi = \{P_i\}$, where each $P_i \subset \Omega$ is a meaningful itemset. Further in Sec. 3.3, a top-down self-supervised clustering method is proposed by feeding back the discovered meaningful itemsets $\Psi$ to supervise the clustering process. A better visual item codebook $\Omega$ is then obtained by applying the trained similarity metric for better representing visual primitives. Finally, in Sec. 3.4, in order to handle the incomplete sub-pattern problem, we propose a pattern summarization method to further cluster those meaningful itemsets (incomplete sub-patterns) and recover the integral semantically meaningful pattern $H_j$. 
3.2. Discovering Meaningful Visual Itemsets

3.2.1. Visual Primitive Extraction

We apply the PCA-SIFT points [40] as the visual primitives. Such visual primitives are mostly located in the informative image regions such as corners and edges, and the features are invariant under rotations, scale changes, and slight viewpoint changes. Normally each image may contain hundreds to thousands of such visual primitives based on the size of the image. According to [40], each visual primitive is a $41 \times 41$ gradient image patch at the given scale, and rotated to align its dominant orientation to a canonical direction. Principal component analysis (PCA) is applied to reduce the dimensionality of the feature. Finally each visual primitive is described as a 35-dimensional feature vector $\vec{f}_i$. These visual primitives are clustered into visual items through $k$-means clustering, using Euclidean metric in the feature space. We will discuss how to obtain a better visual item codebook $\Omega$ based on the proposed self-supervised metric learning scheme in Sec. 3.3.

3.2.2. Meaningful Itemset Mining

Given an image dataset $D_T$ and its induced transaction database $T$, the task is to discover the meaningful itemset (MI) $P \subseteq \Omega$ ($|P| \geq 2$). To evaluate the significance of an itemset $P \subseteq \Omega$, simply checking its frequency $frq(P)$ in $T$ is far from sufficient. For example, even if an itemset appears frequently, it is not clear whether such co-occurrences among the items are statistically significant or just by chance. In order to evaluate the statistical significance of a frequent itemset $P$, we propose a new likelihood ratio test criterion. We compare the likelihood that $P$ is generated by the meaningful pattern versus the likelihood that $P$ is randomly generated, i.e. by chance.
More formally, we compute the likelihood ratio for an itemset $\mathcal{P} \subseteq \Omega$ based on the two hypotheses, where

$H_0$: occurrences of $\mathcal{P}$ are randomly generated;

$H_1$: occurrences of $\mathcal{P}$ are generated by the hidden pattern.

Given a transaction database $\mathbf{T}$, the likelihood ratio $L(\mathcal{P})$ of an itemset $\mathcal{P} = \{W_i\}_{i=1}^{\lvert \mathcal{P} \rvert}$ can be calculated as:

$$L(\mathcal{P}) = \frac{P(\mathcal{P}|H_1)}{P(\mathcal{P}|H_0)} = \frac{\sum_{i=1}^{N} P(\mathcal{P}|T_i, H_1) P(T_i|H_1)}{\prod_{i=1}^{\lvert \mathcal{P} \rvert} P(W_i|H_0)} \quad (3.2)$$

where $P(T_i|H_1) = \frac{1}{N}$ is the prior, and $P(\mathcal{P}|T_i, H_1)$ is the likelihood that $\mathcal{P}$ is generated by a hidden pattern and is observed at a particular transaction $T_i$, such that $P(\mathcal{P}|T_i, H_1) = 1$, if $\mathcal{P} \subseteq T_i$; and $P(\mathcal{P}|T_i, H_1) = 0$, otherwise. Consequently, based on Eq. 3.1, we can calculate $P(\mathcal{P}|H_1) = \frac{\text{freq}(\mathcal{P})}{N}$. We also assume that the items $W_i \in \mathcal{P}$ are conditionally independent under the null hypothesis $H_0$, and $P(W_i|H_0)$ is the prior of item $W_i \in \Omega$, i.e. the total number of visual primitives that are labeled with $W_i$ in the image database $\mathbf{D}_T$. We thus refer $L(\mathcal{P})$ as the “significance” score to evaluate the deviation of a visual itemset $\mathcal{P}$. In fact if $\mathcal{P} = \{W_A, W_B\}$ is a second-order itemset, then $L(\mathcal{P})$ is the mutual information criterion, e.g., the lift criterion, to test the dependency.

It is worth noting that $L(\mathcal{P})$ may favor high-order itemsets even though they appear less frequently. Table 3.1 gives an example, where 90 transactions have only items $A$ and $B$; 30 transactions have $A, B$ and $C$; 61 transactions have $D$ and $E$; and 19 transactions have $C$ and $E$. From Table 3.1, it is easy to evaluate the significant scores for $\mathcal{P}_1 = \{A, B\}$ and $\mathcal{P}_2 = \{A, B, C\}$ with $L(\mathcal{P}_1) = 1.67$ and $L(\mathcal{P}_2) = 1.70 > L(\mathcal{P}_1)$. This result indicates that $\mathcal{P}_2$ is a more significant pattern than $\mathcal{P}_1$ but counter-intuitive. This observation challenges our intuition because $\mathcal{P}_2$ is not a cohesive pattern. For example, the other two sub-patterns of $\mathcal{P}_2$,
Table 3.1. Transaction database $\mathbf{T}_1$.

<table>
<thead>
<tr>
<th>transaction</th>
<th>number</th>
<th>$L(P)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>AB</td>
<td>90</td>
<td>1.67</td>
</tr>
<tr>
<td>ABC</td>
<td>30</td>
<td>1.70</td>
</tr>
<tr>
<td>DE</td>
<td>61</td>
<td>2.5</td>
</tr>
<tr>
<td>CE</td>
<td>19</td>
<td>0.97</td>
</tr>
</tbody>
</table>

$P_3 = \{A, C\}$ and $P_4 = \{B, C\}$, contain almost independent items: $L(P_3) = L(P_4) = 1.02$. Actually, $P_2$ should be treated as a variation of $P_1$ as $C$ is more likely to be a noise. The following equation explains what causes the incorrect result. We calculate the significant score of $P_2$ as:

$$L(P_2) = \frac{P(A, B, C)}{P(A)P(B)P(C)} = L(P_1) \times \frac{P(C|A, B)}{P(C)}.$$ (3.3)

Therefore when there is a small disturbance with the distribution of $C$ over $\mathbf{T}_1$ such that $P(C|A, B) > P(C)$, $P_2$ will compete $P_1$ even though $P_2$ is not a cohesive pattern (e.g. $C$ is not related with either $A$ or $B$). To avoid those free-riders such as $C$ for $P_1$, we perform a more strict test on the itemset. For a high-order itemset $P$ ($|P| > 2$), we perform the Student $t$-test for each pair of its items to check if items $W_i$ and $W_j$ ($W_i, W_j \in P$) are really dependent (see Appendix 7.2 for details.) A high-order itemset $P_i$ is meaningful only if all of its pairwise subsets can pass the test individually: $\forall i, j \in P, t(\{W_i, W_j\}) > \tau$, where $\tau$ is the confidence threshold for the $t$-test. This further reduces the redundancy among the discovered itemsets.

Finally, to assure that a visual itemset $P$ is meaningful, we also require it to appear relatively frequent in the database, i.e. $frq(P) > \theta$, such that we can eliminate those itemsets that appear rarely but happen to exhibit strong spatial dependency among items. With these three criteria, a meaningful visual itemset is defined as follows.
Definition 3. Meaningful Itemset (MI)

An itemset \( P \subseteq \Omega \) is \((\theta, \tau, \gamma)\)-meaningful if it is:

1. frequent: \( \text{frq}(P) > \theta \);
2. pair-wisely cohesive: \( t\{W_i, W_j\} > \tau, \forall i, j \in P \);
3. significant: \( L(P) > \gamma \).

3.2.3. Spatial Dependency

Suppose primitives \( v_i \) and \( v_j \) are spatial neighbors, their induced transaction \( T_i \) and \( T_j \) will have large spatial overlap. Due to such spatial dependency among the transactions, it can cause over-counting problem if simply calculating \( \text{frq}(P) \) from Eq. 3.1. Fig. 3.2 illustrates this phenomena where \( \text{frq}(P) \) contains duplicate counts.

![Image Composed of Visual Items](image.png)

Figure 3.2. Illustration of the frequency over-counting caused by the spatial overlap of transactions. The itemset \( \{A,B\} \) is counted twice by \( T_1 = \{A,B,C,E\} \) and \( T_2 = \{A,B,D,F\} \), although it has only one instance in the image. Namely there is only one pair of \( A \) and \( B \) that co-occurs together, such that \( d(A,B) < 2\epsilon \) with \( \epsilon \) the radius of \( T_1 \). In the texture region where visual primitives are densely sampled, such over-count will largely exaggerate the number of repetitions for a texture pattern.
In order to address the transaction dependency problem, we apply a two-phase mining scheme. First, without considering the spatial overlaps, we perform closed FIM to obtain a candidate set of meaningful itemsets. For these candidates $F = \{P_i : \text{frq}(P_i) > \theta\}$, we re-count the number of their real instances exhaustively through the original image database $D_x$, not allowing duplicate counts. This needs one more scan of the whole database. Without causing confusion, we denote $\hat{\text{frq}}(P)$ as the real instance number of $P$ and use it to update $\text{frq}(P)$. Accordingly, we adjust the calculation of $P(P|H_1) = \frac{\hat{\text{frq}}(P)}{\hat{N}}$, where $\hat{N} = N/K$ denotes the approximated independent transaction number with $K$ the average size of transactions. In practice, as $\hat{N}$ is hard to estimate, we rank $P_i$ according to their significant value $L(P)$ and perform the top-K pattern mining.

Integrating all the contents in this section, our meaningful itemsets mining (MIM) algorithm is outlined in Algorithm 2.

**Algorithm 2**: Meaningful Itemset Mining (MIM)

**input**: Transaction dataset $T$, MI parameters: ($\theta, \tau, \gamma$)

**output**: a collection of meaningful itemsets: $\Psi = \{P_i\}$

1. **Init**: closed FIM with $\text{frq}(P_i) > \theta$: $F = \{P_i\}$, $\Psi \leftarrow \emptyset$;

2. **foreach** $P_i \in F$ do GetRealInstanceNumber($P_i$)

3. **for** $P_i \in F$ do

4.   **if** $L(P_i) > \gamma$ $\land$ PassPairwiseTtest($P_i$) **then**

5.     $\Psi \leftarrow \Psi \cup P_i$

6. **Return** $\Psi$
3.3. Self-supervised Clustering of Visual Item Codebook

Toward discovering meaningful visual patterns in images, it is critical to obtain optimal visual item codebook $\Omega$. A bad clustering of visual primitives brings large quantization errors when translating the continuous high-dimensional visual features $\vec{f} \in \mathbb{R}^d$ into discrete labels $W_i \in \Omega$. Such quantization error reflected in the induced transaction database can affect the data mining results significantly, and thus needs to be minimized.

To improve the clustering results, one possible method is to provide some supervisions, e.g. partially label some instances or give some constrains for pairs of instances belonging to the same or different clusters. Such a semi-supervised clustering method has demonstrated its ability in greatly improving the clustering results [41]. However, in our unsupervised clustering setting, there does not exist apparent supervisions. Thus an interesting question is: is it possible to obtain some supervisions from the completely unlabeled visual primitives? Although it is amazing to see the answer is yes, we can explain the reason based on the hidden structure of the image data. It is worth noting that those visual primitives are not independently distributed in the images and appearing in the transactions. There are hidden patterns that bring structures in the visual primitive distributions. And such structures can be observed and recovered from the transaction database. For example, if we observe that item $W_i$ always appears together with item $W_j$ in a local region, we can infer that they should be generated from a hidden pattern rather than randomly generated. Each pair of $W_i$ and $W_j$ is thus an instance of the hidden pattern. When such hidden patterns (structures) of the data are discovered through our meaningful itemsets mining, we can apply them as supervision to further improve the clustering results.
By discovering a set of MIs $\Psi = \{P_i\}$, we firstly define the *meaningful item codebook* as follows:

**Definition 4. Meaningful Item Codebook $\Omega^+$**

*Given a set of meaningful itemsets $\Psi = \{P_i\}$, an item $W_i \in \Omega$ is meaningful if it belongs to any $P \in \Psi$: $\exists P \in \Psi$, such that $W_i \subseteq P$. All of the meaningful items form the meaningful item codebook $\Omega^+ = \bigcup_{i=1}^{\vert \Psi \vert} P_i$.***

Based on the concept of meaningful item codebook, the original $\Omega$ can be partitioned into two disjoined subsets: $\Omega = \Omega^+ \cup \Omega^-$, where $\Omega^- = \Omega \setminus \Omega^+$. For any $P_i \in \Psi$, we have $P_i \subseteq \Omega^+$ and $P_i \not\subset \Omega^-$. Since only $\Omega^+$ can compose MI, $\Omega^+$ is the meaningful item codebook. Correspondingly we denote $\Omega^-$ as the *meaningless item codebook*, because an item $W_i \in \Omega^-$ never appears in any $P_i \in \Psi$. In such a case, $W_i \in \Omega^-$ should be a noisy or redundant item that is not of interests, for example, located in the clutter background of the image.

For each class $W_i \in \Omega^+$, its positive training set $D^+_{W_i}$ contains the visual primitives $v_i \in D_T$ that satisfy the following two conditions simultaneously:

1. $Q(v_i) = W_i$, where $Q(\cdot)$ is the quantization function from the continuous high-dimensional feature to the discrete item.
2. $v_i \in T(P_1) \cup T(P_2) \cup ... \cup T(P_c)$, where $P_j$ is the meaningful itemset that contains $W_i$, namely $\forall j = 1, ..., c, W_i \subseteq P_j$.

In summary, not all $v_i$ labeled with $W_i$ are qualified as positive training samples for item class $W_i \in \Omega^+$. We only choose those visual primitives that can constitute meaningful itemsets. Such visual primitives are very likely generated from the hidden pattern $\mathcal{H}$ that explains the MI.
With these self-labeled training data for each meaningful item \( W_i \in \Omega^+ \), we transfer the originally unsupervised clustering problem into semi-supervised clustering. Still, our task is to cluster all the visual primitives \( v_i \in D_T \). But now some of the visual primitives are already labeled after MIM. Thus many semi-supervised clustering methods are feasible to our task. Here we apply the nearest component analysis (NCA) [42] to improve the clustering results by learning a better Mahalanobis distance metric in the feature space.

**Neighborhood Component Analysis (NCA)**

Similar to linear discriminative analysis (LDA), NCA targets at learning a global linear projection matrix \( A \) for the original features. However, unlike LDA, NCA does not need to assume that each visual item class has a Gaussian distribution and thus can be applied to more general cases. Given two visual primitives \( v_i \) and \( v_j \), NCA learns a new metric \( A \) and the distance in the transformed space is: \( d_A(v_i, v_j) = (\vec{f}_i - \vec{f}_j)^T A^T A (\vec{f}_i - \vec{f}_j) = (A \vec{f}_i - A \vec{f}_j)^T (A \vec{f}_i - A \vec{f}_j) \).

The objective of NCA is to maximize a stochastic variant of the leave-one-out K-NN score on the training set. In the transformed space, a point \( v_i \) selects another point \( v_j \) as its neighbor with probability:

\[
p_{ij} = \frac{\exp(-\|A \vec{f}_i - A \vec{f}_j\|^2)}{\sum_{k \neq i} \exp(-\|A \vec{f}_i - A \vec{f}_k\|^2)}, \quad p_{ii} = 0.
\]  

(3.4)

Under the above stochastic selection rule of nearest neighbors, NCA tries to maximize the expected number of points correctly classified under the nearest neighbor classifier (the average leave-one-out performance):

\[
f(A) = \sum_i \sum_{j \in C_i} p_{ij},
\]

(3.5)
where \( C_i = \{ j | c_i = c_j \} \) denotes the set of points in the same class as \( i \). By differentiating \( f \), the objective function can be maximized through gradient search for optimal \( A \). After obtaining the projection matrix \( A \), we update all the visual features of \( v_i \in D_x \) from \( \bar{f}_i \) to \( A\bar{f}_i \), and re-cluster the visual primitives based on their new features \( A\bar{f}_i \).

### 3.4. Pattern Summarization of Meaningful Itemsets

As discussed before, there are imperfections when translating the image data into transactions. Suppose there exists a hidden visual pattern \( H_j \) (e.g. a semantic pattern “eye” in the face category) that repetitively generates a number of instances (eyes of different persons) in the image database. We can certainly observe such meaningful repetitive patterns in the image database, for example, discovering meaningful itemsets \( P_i \) based on Def. 3. However, instead of observing a unique integral pattern \( H_j \), we tend to observe many incomplete sub-patterns with compositional variations due to noise, i.e. many synonyms itemsets \( P_i \) that correspond to the same \( H_j \) (see Fig. 3.3). Again, this can be caused by many reasons, including the missing detection of visual primitives, quantization error of visual primitives, and partial occlusion of the hidden pattern itself. Therefore, we need to cluster those correlated MIs (incomplete sub-patterns) in order to recover the complete pattern \( H \).

![Figure 3.3. Motivation for pattern summarization. An integral hidden pattern may generate incomplete and noisy instances. The pattern summarization is to recover the unique integral pattern through the observed noisy instances.](image)
According to [28], if two itemsets \( P_i \) and \( P_j \) are correlated, then their transaction set \( T(P_i) \) and \( T(P_j) \) (Eq. 3.1) should also have a large overlap, implying that they may be generated from the same pattern \( \mathcal{H} \). As a result, \( \forall i, j \in \Psi \), their similarity \( s(i, j) \) should depend not only on their frequencies \( \hat{f}rq(P_i) \) and \( \hat{f}rq(P_j) \), but also the correlation between their transaction set \( T(P_i) \) and \( T(P_j) \). Given two itemsets, there are many methods to measure their similarity including KL-divergence between pattern profiles [28], mutual information criterion and Jaccard distance [43]. We apply the Jaccard distance here although others are certainly applicable. The corresponding similarity between two MI \( P_i \) and \( P_j \) is defined as:

\[
s(i, j) = \exp^{-1 \frac{|T(P_i) \cap T(P_j)|}{|T(P_i) \cup T(P_j)|}}.
\]

(3.6)

Based on this, our pattern summarization problem can be stated as follows: given a collection of meaningful itemsets \( \Psi = \{P_i\} \), we want to cluster them into unjoined K-clusters. Each cluster \( \mathcal{H}_j = \{P_i\}^{[\mathcal{H}_j]} \) is defined as a meaningful visual pattern, where \( \cup_j \mathcal{H}_j = \Psi \) and \( \mathcal{H}_i \cap \mathcal{H}_j = \emptyset, \forall i, j \). The observed MI \( P_i \in \mathcal{H} \) are instances of the visual pattern \( \mathcal{H} \), with possible variations due to imperfections from the images. We propose to apply the normalized cut algorithm [44] for clustering MI. Normalized cut is a well-known algorithm in machine learning and computer vision community. Originally it is applied for clustering-based image segmentation.

**Normalized Cut (NCut)**

Let \( G = \{V, E\} \) denote a fully connected graph, where each vertex \( P_i \in V \) is an MI, and the weight \( s(i, j) \) on each edge represents similarity between two MIs \( P_i \) and \( P_j \). Normalized cut can partition the graph \( G \) into clusters. In the case of bipartition, \( V \) is partitioned into two disjoined sets \( A \cup B = V \). The following cut value needs to be minimized to get the
optimal partition:

\[ Ncut(A, B) = \frac{cut(A, B)}{assoc(A, V)} + \frac{cut(A, B)}{assoc(B, V)}, \]  

(3.7)

where \( cut(A, B) = \sum_{i \in A, j \in B} s(i, j) \) is the cut value and \( assoc(A, V) = \sum_{i \in A, j \in V} s(i, j) \) is the total connection from the vertex set \( A \) to all vertices in \( G \). To minimize the \( Ncut \) in Eq. 3.7, we need to solve the following standard eigenvector problem:

\[ D^{-\frac{1}{2}}(D - S)D^{-\frac{1}{2}}z = \lambda z, \]

(3.8)

where \( D \) is a diagonal matrix with \( \sum_j s(i, j) \) on its diagonal and otherwise are 0; \( S \) is a symmetric matrix with \( s(i, j) \) its element. The eigenvector corresponding to the second smallest eigenvalue can be used to partition \( V \) into \( A \) and \( B \). In the case of multiple \( K \)-class partitioning, the bipartition can be utilized recursively or just apply the eigenvectors corresponding to the \( K + 1 \) smallest eigenvalues.

We summarize our visual pattern discovery algorithm as follows.
Algorithm 3: Main Algorithm

\textbf{input}: Image dataset \( D \),
\( \epsilon \) or \( K \) for searching spatial \( \epsilon \)-NN or K-NN,
MIM parameter: \((\theta, \tau, \gamma)\),
number of meaningful patterns: \(|H|\),
number of maximum iteration \( l \)
\textbf{output}: A set of meaningful patterns: \( H = \{H_i\} \)

1 \textbf{Init}: Get visual item codebook \( \Omega^0 \) and induced transaction DB \( T_{\Omega}^0 \); \( i \leftarrow 0 \);

2 \textbf{while} \( i < l \) \textbf{do}

3 \hspace{1em} \Psi^i = \text{MIM}(T_{\Omega}^i); \quad /* \text{get meaningful itemsets} */

4 \hspace{1em} \Omega^+_i = \cup_j \mathcal{P}_j, \text{ where } \mathcal{P}_j \in \Psi^i;

5 \hspace{1em} A^i = \text{NCA} (\Omega^+_i , T_{\Omega}^i); \quad /* \text{get new metric} */

6 \hspace{1em} \text{Update } \Omega^i \text{ and } T^i \text{ based on } A^i; \quad /* \text{re-clustering} */

7 \hspace{1em} \textbf{if} little change of \( \Omega^i \) \textbf{then}

8 \hspace{2em} \text{break;}

9 \hspace{1em} i \leftarrow i + 1

10 \textbf{S} = \text{GetSimMatrix} (\Psi^i);

11 \textbf{H} = \text{NCut} (\textbf{S}, |H|); \quad /* \text{pattern summarization} */

12 \textbf{Return } \textbf{H};
3.5. Experiments

3.5.1. Setup

Given a large image dataset $D_I = \{I_i\}$, we first extract the PCA-SIFT points [40] in each image $I_i$ and treat these interest points as the visual primitives. We resize all images by the factor of 2/3. The feature extraction is on average 0.5 seconds per image. Multiple visual primitives can be located at the same position, with various scales and orientations. Each visual primitive is represented as a 35-d feature vector after principal component analysis. Then $k$-means algorithm is used to cluster these visual features into a visual item codebook $\Omega$. We select two categories from the Caltech 101 database [45] for the experiments: faces (435 images from 23 persons) and cars (123 images of different cars). We set the parameters for MIM as: $\theta = \frac{1}{4}|D_I|$, where $|D_I|$ is the total number of images, and $\tau$ is associated with the confidence level of 0.90. Instead of setting threshold $\gamma$, we select the top phrases by ranking their $L(P)$ values. We set visual item codebook size $|\Omega| = 160$ and 500 for car and face database respectively when doing $k$-means clustering. For generating the transaction databases $T$, we set $K = 5$ for searching spatial K-NN to constitute each transaction. All the experiments were conducted on a Pentium-4 3.19GHz PC with 1GB RAM running window XP.

3.5.2. Evaluation of Meaningful Itemset Mining

To test whether our MIM algorithm can output meaningful patterns, we want to check if the discovered MI are associated with the frequently appeared foreground objects (e.g., faces and cars) while not located in the clutter backgrounds. The following two criteria are proposed for the evaluation: (1) the precision of $\Psi$: $\rho^+$ denotes the percentage of discovered meaningful
itemsets $\mathcal{P}_i \in \Psi$ that are located in the foreground objects, and (2) the precision of $\Omega^-$: $\rho^-$ denotes the percentage of meaningless items $W_i \in \Omega^-$ that are located in the background. Fig. 3.4 illustrates the concepts of our evaluation. In the ideal situation, if $\rho^+ = \rho^- = 1$, then every $\mathcal{P}_i \in \Psi$ is associated with the interesting object, i.e. located inside the object bounding box; while all meaningless items $W_i \in \Omega^-$ are located in the backgrounds. In such a case, we can precisely discriminate the frequently appeared foreground objects from the clutter backgrounds, through an unsupervised learning. Finally, we use retrieval rate $\eta$ to denote the percentage of retrieved images that contain at least one MI.

![Figure 3.4](image_url)

Figure 3.4: Evaluation of meaningful itemsets mining. The highlight bounding box (yellow) represents the foreground region where the interesting object is located. In the idea case, all the MI $\mathcal{P}_i \in \Psi$ should locate inside the bounding boxes while all the meaningless items $W_i \in \Omega^-$ are located outside the bounding boxes.

In Table 3.2, we present the results of discovering meaningful itemsets from the car database. The first row indicates the number of meaningful itemsets ($|\Psi|$), selected by their $L(\mathcal{P})$. It is shown that when adding more meaningful itemsets into $\Psi$, its precision score $\rho^+$ decreases (from 1.00 to 0.86), while the percentage of retrieved images $\eta$ increases (from 0.11 to 0.88). The high precision $\rho^+$ indicates that most discovered MI are associated with the foreground objects. It is also noted that meaningful item codebook $\Omega^+$ is only a small subset with respect to $\Omega$ ($|\Omega| = 160$). This implies that most visual items actually are not meaningful as they do not constitute the foreground objects. Therefore it is reasonable to get
rid of those noisy items from the background. Examples of meaningful itemsets are shown in Fig. 3.9 and Fig. 3.10.

Table 3.2. Precision score $\rho^+$ and retrieval rate $\eta$ for the car database, corresponding to various sizes of $\Psi$. See text for descriptions of $\rho^+$ and $\eta$.

| $|\Psi|$ | 1 | 5 | 10 | 15 | 20 | 25 | 30 |
|-------|---|---|----|----|----|----|----|
| $|\Omega^+|$ | 2 | 7 | 12 | 15 | 22 | 27 | 29 |
| $\eta$ | 0.11 | 0.40 | 0.50 | 0.62 | 0.77 | 0.85 | 0.88 |
| $\rho^+$ | 1.00 | 0.96 | 0.96 | 0.91 | 0.88 | 0.86 | 0.86 |

We further compare three types of criteria for selecting meaningful itemsets $\mathcal{P}$ into $\Psi$, against the baseline of selecting the individual visual items $W_i \in \Omega$ to build $\Psi$. The three MI selection criteria are: (1) occurrence frequency: $\hat{frq}(\mathcal{P})$ (2) T-score (Eq. 7.5) (only select the second order itemsets, $|\mathcal{P}| = 2$) and (3) likelihood ratio: $L(\mathcal{P})$ (Eq. 3.2). The results are presented in Fig. 3.5. It shows the changes of $\rho^+$ and $\rho^-$ with increasing size of $\Psi$ ($|\Psi| = 1, \ldots, 30$). We can see that all three MI selection criteria perform significantly better than the baseline of choosing the most frequent individual items as meaningful patterns. This demonstrates that FI and MI are more informative features than the singleton items in discriminating the foreground objects from the clutter backgrounds. This is because the most frequent items $W_i \in \Omega$ usually correspond to common features (e.g. corners) which appear frequently in both foreground objects and clutter backgrounds, thus lacking the discriminative power. On the other hand, the discovered MI is the composition of items that function together as a single visual pattern (incomplete pattern though) which corresponds to the foreground object that repetitively appears in the database. Among the three criteria, occurrence frequency $\hat{frq}(\mathcal{P})$ performs worse than the other two criteria, which further demonstrates that not all frequent itemsets are meaningful patterns. It is also shown from Fig. 3.5 that when only selecting a few number of MI, i.e. $\Psi$ has a small size, all the three criteria yield similar performances. However, when more MI are added, the
proposed likelihood ratio test method performs better than the other two, which shows our MIM algorithm can discover meaningful visual patterns.

![Graph showing performance comparison](image)

**Figure 3.5.** Performance comparison by applying three different meaningful itemset selection criteria, also with the baseline of selecting most frequent individual items to build $\Psi$.

By taking advantage of the FP-growth algorithm for closed FIM, our pattern discovery is very efficient. It costs around 17.4 seconds for discovering meaningful itemsets from the face database containing over 60,000 transactions (see Table 3.3). It thus provides us a powerful tool to explore large object category database where each image contains hundreds of primitive visual features.

**Table 3.3.** CPU computational cost for meaningful itemsets mining in face database, with $|\Psi| = 30$.

| # images $|D_I|$ | # transactions $|T|$ | closed FIM $[39]$ | MIM Alg.2 |
|---|---|---|---|
| 435 | 62611 | 1.6 sec | 17.4 sec |

### 3.5.3. Refinement of visual item codebook

To implement NCA for metric learning, we select 5 meaningful itemsets from $\Psi$ ($|\Psi| = 10$). There are in total less than 10 items shared by these 5 meaningful itemsets for both face and car categories, *i.e.* $|\Omega^+| < 10$. For each class, we select the qualified visual primitives as training samples. Our objective of metric learning is to obtain a better representation of
the visual primitives, such that the inter-class distance is enlarged while the intra-class distance is reduced among the self-labeled training samples.

After learning a new metric using NCA, we reconstruct the visual item codebook $\Omega$ through $k$-means clustering again, with the number of clusters slightly less than before. The comparison results of the original visual item codebooks and those after refinement are shown in Fig. 3.6. It can be seen that the precision $\rho^+$ of $\Psi$ is improved after refining the item codebook $\Omega$.

![Figure 3.6. Comparison of visual item codebook before and after self-supervised refinement.](image)

3.5.4. Visual Pattern Discovery through Pattern Summarization

For both car and face categories, we select the top-10 meaningful itemsets by their $L(\mathcal{P})$ (Eq. 3.2). All discovered MI are the second-order or third-order itemsets composed of spatially co-located items. We further cluster these 10 MI ($|\Psi| = 10$) into meaningful visual patterns using the normalized cut. The best summarization results are shown in Fig. 3.7 and Fig. 3.8, with cluster number $|H| = 6$ and $|H| = 2$ for the face and car category respectively. For the face category, the semantic parts like eyes, noses and mouths are identified by various patterns. For the car category, the wheels and car bodies are identified.

To evaluate our pattern summarization results, we apply the precision and recall scores defined as follows: $\text{Recall} = \frac{\# \text{ detects}}{\# \text{ detects} + \# \text{ miss detects}}$ and $\text{Precision} = \frac{\# \text{ detects}}{\# \text{ detects} + \# \text{ false positives}}$. 
detects \((\# \text{ detects} + \# \text{ false alarms})\). From Fig. 3.7 and Fig. 3.8, it can be seen that the summarized meaningful visual patterns \(\mathcal{H}_i\) are associated with semantic patterns with very high precision and reasonably good recall score.

**Figure 3.7.** Selected meaningful itemsets \(\Psi\) (\(|\Psi| = 10\)) and their summarization results (\(|\mathcal{H}| = 6\)) for the face database. Each one of the 10 sub-images contains a meaningful itemset \(P_i \in \Psi\). The rectangles in the sub-images represent visual primitives (e.g., PCA-SIFT interest points at their scales). Every itemset, except for the 3rd one, is composed of 2 items. The 3rd itemset is a high-order one composed of 3 items. Five semantic visual patterns of the face category are successfully discovered: (1) left eye (2) between eyes (3) right eye (4) nose and (5) mouth. All of the discovered meaningful visual patterns have very high precision. It is interesting to note that left eye and right eye are treated as different semantic patterns, possibly due to the differences between their visual appearances. One extra semantic pattern that is not associated with the face is also discovered. It mainly contains corners from computers and windows in the office environment.

**Figure 3.8.** Selected meaningful itemsets \(\Psi\) (\(|\Psi| = 10\)) and their summarization results (\(|\mathcal{H}| = 2\)) for the car database. Two semantic visual patterns that are associated with the car category are successfully discovered: (1) wheels and (2) car bodies (mostly windows containing strong edges). The 5th itemset is a high-order one composed of 3 items.
3.6. Conclusion

Traditional data mining techniques are not directly applicable to image data which contain spatial information and are characterized by high-dimensional visual features. To discover meaningful visual patterns from image data, we present a new criterion for discovering meaningful itemsets based on traditional FIM. Such meaningful itemsets are statistically more interesting than the frequent itemsets. By further clustering these meaningful itemsets (incomplete sub-patterns) into complete patterns through normalized cut, we successfully discover semantically meaningful visual patterns from real images of car and face categories.

In order to bridge the gap between continuous high dimensional visual features and discrete visual items, we propose a self-supervised clustering method by applying the discovered meaningful itemsets as supervision to learn a better feature representation. The visual item codebook can thus be increasingly refined by taking advantage of the feedback from the meaningful itemset discovery.
Figure 3.9. Examples of meaningful itemsets from car category (6 out of 123 images). The cars are all side views, but are of different types and colors and located in various clutter backgrounds. The first row shows the original images. The second row shows their visual primitives (PCA-SIFT points), where each green circle denotes a visual primitive with corresponding location, scale and orientation. The third row shows the meaningful itemsets. Each red rectangle in the image contains a meaningful itemset (it is possible two items are located at the same position). Different colors of the items denote different semantic meanings. For example, wheels are dark red and car bodies are dark blue. The precision and recall scores of these semantic patterns are shown in Fig. 3.8.
Figure 3.10. Examples of meaningful itemsets from face category (12 out of 435 images). The faces are all front views but are of different persons. Each red rectangle contains a meaningful itemset. Different colors of the visual primitives denote different semantic meanings, e.g. green visual primitives are between eyes etc. The precision and recall scores of these semantic patterns are shown in Fig. 3.7.
CHAPTER 4

Mining Recurring Events from Video Data

If an event occurs repetitively, it can be a pattern of great interest. In video data, such recurrence can be exact repetitions, like commercials in TV programs [46] and popular music in audio broadcasting [47] [1]. The recurrence can also be inexact repeats which are similar to each other and share the same spatial-temporal pattern, for example, the same human actions performed by different subjects as shown in Fig. 4.1. In video analysis, it is important to automatically discover recurring video events in understanding, organizing, and searching based on video contents. There are many related applications reported in the literature, such as commercial detection and analysis [46] [48], news topic threading and tracking [49] [50], news broadcast structure analysis [51] [52] [53], and many others mentioned in [54].

Compared with video clip search, where a query clip is usually provided by the user and the task is to find the matches in the video database [49] [55] [56] [57] [58], the problem of recurring event mining is more challenging, because it has to be unsupervised and blind of a target [59] [60] [1], as these is no query provided. In other words, there is generally no a priori knowledge of the events to be discovered, including (i) what the recurring events are; (ii) where they occur in the video; (iii) how long they last; and (iv) how many recurrences there are or even the existence of such a recurring event. Exhaustively searching for the recurrences by checking all possible durations and locations is computationally prohibitive, if not impossible, in large video databases.
Figure 4.1. A typical dance movement in the Michael Jackson-style dance, performed by two different subjects (first and second rows). Such a dynamic motion pattern appears frequently in the Michael Jackson-style dance and is a recurring event in the dance database. The spatial-temporal dynamics in human motions can contain large variations, such as non-uniform temporal scaling and pose differences, depending on the subject’s performing speed and style. Thus it brings great challenges in searching and mining them.

Although efficient algorithms have been proposed to reduce the computational complexity in finding exact repetitions [54] [51], mining recurring events remains to be a very challenging problem when the recurring patterns exhibit content or temporal variations (i.e. they are not exact repetitions). For instance, the same video event may vary depending on the encoding scheme and parameters (e.g. frame size/rate, color format), or content changes due to post-editing, not to mention the intra-class variations. Taking the human action patterns as another example, if we treat each typical action as a recurring event, such a recurring pattern can be performed very differently depending on the speed, style, and the subject [61] [62]. As shown in Fig. 4.1, although the two human actions belong to the same motion pattern, they are far from identical. Consequently, how to handle the possible variations in the recurring events brings extra challenges to the mining problem, especially given the fact that we have no \textit{a priori} knowledge of the recurring pattern [63].

To automatically discover recurring events, our emphasis in this work is not only on exact repeats such as duplicate commercials or music patterns as studied before [51] [54], but also events that are subject to large temporal and spatial variations, such as representative actions.
in human movements. To this end, we propose a novel method called “forest-growing” in this paper. First of all, a video or motion sequence is chopped into a sequence of video primitives (VPs), each characterized by a feature vector. Suppose the whole database generates in total \( N \) VPs, instead of calculating and storing a full \( N \times N \) self-similarity matrix as in previous methods [51] [64] [65], for each VP, we query the database and obtain its \( K \) best matches. A matching-trellis can be built to store the \( N \) query results, which is of limited size \( K \times N \ (K \ll N) \). This treatment saves both computational and memory costs, and it is still effective in preserving the best \( K \) matches that keep the important information. Based on the matching-trellis, it is clear that a temporally continuous path established in the trellis corresponds to a repetition of a recurring event. Without knowing the location and duration of the repetition, we can grow trees in the matching-trellis, where each branch is a continuous path and is associated with a repetition. The length of the branch can be automatically adapted to tolerate the content and temporal variations. Since the total number of recurring events is unknown, multiple trees need to be grown simultaneously. This process gives the name of our method “forest-growing”.

In the proposed method, several techniques are used to make the mining process computationally efficient. To speed up the process of building the trellis, we utilize locality sensitive hashing (LSH) [17] for approximate nearest neighbor (NN) query. LSH is a sub-linear method compared with exhaustive linear search. Considering the fact that we have in total \( N \) queries, the actual total complexity of building the matching-trellis is sub-quadratic with respect to the size of the dataset \( N \). Therefore, a large computational gain is achieved. To handle the possible variations in mining recurring events, as well as the potential inaccuracy caused by the approximate nearest neighbor search, a branching factor is introduced in our method in growing continuous branches in the forest. Using a carefully designed
message-passing scheme that needs one auxiliary array of size $N$, we achieve a $O(NK)$ complexity in finding all continuous paths. Thus the overall complexity of our method remains sub-quadratic with respect to the database size, and is memory-friendly compared with the methods that need to store the full $N \times N$ self-similarity matrix.

We highlight the advantages of our forest-growing method as follows:

- it can automatically discover all the recurring events without a prior knowledge, and the duration of each recurring event is dynamically determined without exhaustively searching for all possible ones;
- it can handle non-uniform temporal scaling and content variations by using a branching factor in growing the forest;
- it is computationally efficient with an overall sub-quadratic complexity with respect to the dataset size, by using locality sensitive hashing (LSH) and a carefully designed forest-growing process;
- it does not require video shot segmentation and can be easily extended to other types of time-series data.

The remaining of this paper is organized as follows. We briefly describe the related work in the literature in Section 4.1, followed by the description of the proposed forest-growing algorithm in Section 4.2. To test the efficiency of our method, we apply our algorithm to find repetitive commercials in TRECVID news video [66] and report the results in Section 4.4. To demonstrate the effectiveness of our method in handling large spatio-temporal variations, we run the algorithm on a 32,260 frame motion captured human dance data to discover recurring motion patterns and report the results in Section 4.5. We conclude in Section 4.6.
Figure 4.2. Mining recurring events through finding continuous paths in the matching-trellis. Each node denotes a primitive $S \in V$, labeled by its temporal index. We show part of the whole matching-trellis from column 101 to 110. Given the dataset $V$, i.e., the top row sequence, we query each $S \in V$ and find its $K$ best matches, i.e., each column denotes a matching set $M_S$. For instance, the matching set of $S_{101}$ is $M_{S_{101}} = \{S_{120}, S_{720}, S_{410}, S_{374}, S_{198}, S_{721}\}$. The highlighted nodes constitute an established tree, which grows from left to right, with the temporal index increasing monotonically from root to leaf nodes. Each tree branch is a continuous path in terms of temporal indices, which indicates a discovered repetition corresponding to the original sequence in the top row. For example, the branch $\{S_{720}, S_{722}, S_{725}, S_{729}, S_{731}, S_{733}, S_{736}\}$ is a repetition of the top row segment $\{S_{101}, S_{102}, S_{103}, S_{104}, S_{105}, S_{106}, S_{107}\}$. The longest branch highlighted in orange is picked to represent the whole tree. Although we only show a single tree growing, because the algorithm mimics the process of growing multiple trees simultaneously, we call it “forest-growing”.

4.1. Related Work

There have been many work in mining exact repeats from video [51] and audio [54] [1] [47] streams. In [1], an effective on-line audio stream mining system is proposed to extract repetitive audio segments in real time without human intervention. The method depends on robust audio fingerprints and its similarity search is accelerated by taking advantage of the dense
sampling rate in audio signals. The boundaries of the repeat segments can be accurately
determined by performing an exhaustive search. In [51], a video repetition mining method
is proposed, which can discover very short repeats from news videos. These short repeats are
program lead-in/lead-out clips that indicate the starting or ending points of a particular TV
program. Hence locating these short flag clips can help reveal and understand the structures
of the news videos. To speed up the similarity matching process, locality sensitive hashing
is applied in [51]. Besides mining repetitive video segments, there are also existing works in
finding repeats at the image or video shot level, such as near-duplicate image detection [49]
and identical shot detection [50] [67]. However, these methods cannot be directly applied to
recurring event mining, where an event can be of arbitrary length and may contain a number
of shots.

Other than repetition discovery from videos, there are increasing interests of mining
recurring patterns from human motion data as well in the computer graphics literature [68].
As more motion databases become available and their sizes increase, manually labeling and
categorizing motions becomes a very time-consuming, if not impossible, task. On the other
hand, representative and recurring motion patterns (motion motifs [63]) in human motion
data can reveal important semantic structures in human motions, which can be used for
motion analysis, automatic database annotation, motion retrieval [69] and motion synthesis
from existing data [70] [71]. Due to the large variations in human motions, it greatly
challenges the task of mining recurring patterns.

Related works in mining repetitive patterns from music have also been reported in the
literature. For example, the key melody that appears repetitively in the music can be used
in analyzing the themes of the song. In [72], a music repetition mining method is proposed
that can tolerate significant variations in parameters, such as dynamics, timbre, execution
of note groups, modulation, articulation, and tempo progression. Discovering such recurring events in music can help understand the music theme and structure [73] and it is helpful to construct indices and facilitate queries for music retrieval applications [74].

Motivated by the successes of mining repetitions from text data, one promising solution of recurring event discovery is to translate temporal sequences (e.g. music, videos, or human motions) to symbolic sequences that are similar to text strings. Thus, it is hoped that traditional text search and mining methods may be directly applied for temporal events discovery. For example, by treating music as note strings, we can find repetitive music segments by mining common sub-strings [74]. Similarly, by quantizing each video frame into a discrete symbol [52] and translating a video sequence into a DNA-like string, mining recurring events becomes a problem of discovering repetitive motifs from a string database. In spite of successes in previous work [75] [52] [76] [74], we notice that it is unnecessary to translate multimedia sequences into symbolic strings, in order to discover repetitions. Compared with text data, multimedia data are not characterized as symbolic sequences. For example, in video and motion sequence analysis, a general practice is to characterize a video frame or a human pose as a feature vector. Although mapping the continuous feature vectors to discrete symbols can significantly reduce the dimensionality, it inevitably introduces quantization errors, and it in turn degrades the representation power of the original continuous features, especially in high dimensional feature space. This paper presents a new event mining method that utilizes the continuous video features directly, instead of quantizing them into discrete symbolic labels.
4.2. Algorithm description

4.2.1. Overview

Without loss of generality, we denote the whole database as a long sequence: \( \mathcal{V} = \{ \mathbf{S}_i \}_{i=1}^{N} \), where \( \mathbf{S}_i \) is the video primitive (VP). Depending on the application, \( \mathbf{S} \) can be a video segment or an individual frame. After feature extraction, each \( \mathbf{S}_i \) is characterized by a \( d \)-dimensional feature vector: \( \mathbf{S}_i \in \mathbb{R}^d \).

We define that a sub-sequence \( \mathcal{V}_1 \subset \mathcal{V} \) belongs to a recurring event, if it is similar to another subsequence \( \mathcal{V}_2 \subset \mathcal{V} \). In such a case, both \( \mathcal{V}_1 \) and \( \mathcal{V}_2 \) are recurring instances of the same recurring event \( \mathcal{E} = \{ \mathcal{V}_1, \mathcal{V}_2 \} \). For a recurring event \( \mathcal{E} \), it is possible that its multiple instances vary from each other. As a data mining problem, before pattern discovery, it is unclear how long the recurring instances are, where they are, and how many of them there are. The algorithm is required to discover all these repetitions of various lengths and content.

Although it is difficult to discover recurring events of unknown lengths, it is straightforward to discover the repetitions of individual VPs. Given \( \mathbf{S}_i \in \mathcal{V} \), we denote all of its best matches as its matching set

\[
\mathcal{M}_{\mathbf{S}_i} = \{ \mathbf{S}_j : \| \mathbf{S}_i - \mathbf{S}_j \| \leq \epsilon, \forall j, |i - j| > \hat{N} \},
\]

where \( \epsilon \) is the similarity threshold; \( \| \cdot \| \) denotes the dissimilarity measurement, e.g. Euclidean distance; \( \hat{N} \) is the minimum temporal distance used to filter similar matches caused by temporal redundancy. Hence \( \mathcal{M}_{\mathbf{S}_i} \) does not include the temporal neighbors of \( \mathbf{S}_i \).

To make the notation consistent, we assume that the average size of \( \mathcal{M}_{\mathbf{S}_i} \) is \( K \) and describe the matching-trellis \( \mathcal{M}_\mathcal{V} \) as a \( K \times N \) matrix, where each column stores a matching set \( \mathcal{M}_{\mathbf{S}_i} \). As briefly explained in Fig. 4.2, mining recurring events can be translated into
the problem of finding continuous paths in the trellis $\mathcal{M}_V$, where each continuous path corresponds to a recurring instance. In the following, we discuss in detail how to efficiently build the matching-trellis in Section 4.2.2, and how to efficiently find the continuous paths in Section 4.2.3. Finally, how to cluster all discovered recurring instances into event groups is discussed in Section 4.2.4.

4.2.2. Step 1. Build the Matching-Trellis

As an overhead of our algorithm, we need to find the best matches for each $S_i \in V$, in order to build the matching-trellis. Exhaustive search of best matches is of linear complexity, thus is not computationally efficient considering that we have $N$ queries in total. To find the best matches more efficiently, we use LSH [17] to perform approximate $\epsilon$-NN query for each primitive $S_i \in V$. Instead of searching for the exact $\epsilon$-NN, LSH searches for the approximate $\epsilon$-NN, and can achieve sub-linear query time. Hence the total cost of building the trellis is reduced to sub-quadratic given $N$ queries.

We briefly explain how LSH works as follows. Essentially, LSH provides a randomized solution for a high-dimensional $\epsilon$-NN query problem. It sacrifices accuracy to gain efficiency. In LSH, there is a pool of hash functions. Each hash function $h(\cdot)$ is a random linear mapping from vector $S$ to an integer, $h : \mathbb{R}^d \rightarrow \mathbb{N}$,

$$h_{a,b}(S) = \left\lfloor \frac{a \cdot S + b}{r} \right\rfloor,$$

where $a$ is a random vector of $d$-dimension and $b$ is a random variable chosen uniformly from $[0, r]$. Under a specific hash function, two vectors $S_p$ and $S_q$ are considered a match if their hash values are identical. The closer $S_p$ and $S_q$ are in $\mathbb{R}^d$, the more possible that they
have the identical hash value, which is guaranteed by the property of \((r_1, r_2, p_1, p_2)\)-sensitive hash function \cite{17}. By pre-building a set of hashing functions for the database, each new query vector \(S_q\) can efficiently retrieve most of its nearest neighbors by only comparing its hash values against those in the database instead of calculating the pair-wise distances in \(\mathbb{R}^d\). Thus large computational cost can be saved.

However, despite the large efficiency gain from LSH, as a solution to approximate NN search, LSH may result in problems of missed retrieval. To compensate for the inaccuracy caused by LSH and to handle the content and temporal variations, we introduce a branching factor in Section 4.2.3 for forest-growing. Later, we discuss how to determine the parameter \(\epsilon\) in the NN search and the branching factor \(B\) in Section 4.3.

4.2.3. Step 2. Mining Repetitions through Growing a Forest

As explained before, each temporally continuous path in the matching-trellis indicates a repetition. However, to find all continuous paths through an exhaustive check is not efficient. As seen in Fig. 4.2, there are in total \(K^N\) possible paths of length \(N\) in the trellis, not to mention those of lengths shorter than \(N\).

Motivated by the idea of dynamic programming, we introduce an algorithm that simulates a forest-growing procedure to discover all the continuous paths in a matching-trellis. Every node in the \(K \times N\) matching-trellis can be a seed and start a new tree in the forest if it satisfies the following growing condition and does not belong to any existing trees.

**Definition 5. tree growing condition**

In the matching-trellis, a VP \(S_i \in \mathcal{M}_{S_q}\) can grow if there exists another available \(S_j \in \mathcal{M}_{S_q+1}\), such that \(j \in [i, i + B - 1]\). Here \(q, i, j\) denote the temporal indices, and \(B \in \mathbb{N}^+\) is the
branching factor that adaptively adjusts the growing speed.

Fig. 4.2 illustrates one grown tree in the matching-trellis. As a tree grows, it automatically establishes the temporal correspondences between its growing branches and their counterparts in the database (top row segment in dark green in Fig. 4.2). Repetitions are thus naturally discovered. The total number of repetitive instances is determined by the number of valid trees. The length of a repetition is determined by the length of the longest branch in a tree.

It is worthy noting that the branching factor $B$ plays an important role in the forest-growing procedure. It handles variations in the recurring events and ensures the robustness of our algorithm. Given a tree branch, its temporal index increases monotonically from the root to the leaf node, where the branching factor controls the growing speed. For example, if $B = 3$, a tree branch can grow up to 2 times faster than the original sequence (corresponding top row path as shown in Fig. 4.2), whose temporal index is strictly increased by 1 in each step. On the other hand, a tree branch can grow much slower than the original sequence when its temporal index increases by 0 in each step. In other words, the growing speed of a branch always adapts to the speed of its corresponding repetition in the top row. Hence we can now accommodate non-uniform temporal scaling among instances of the same recurring event. More importantly, by introducing the branching factor, our algorithm can also tolerate local errors as a tree grows, such as noisy frames or the inaccuracy due to the approximate NN search through LSH. For example, even if LSH fails to retrieve a matching node thus the node does not appear in the next column, the tree still has the chance to grow via the other $B - 1$ branches. So the recurring events can still be discovered despite the missed retrieval.
In terms of complexity, since there are in total $N$ columns in the trellis, the algorithm takes $N - 1$ steps to finish growing the forest. In each step, we need to check $K^2$ pairs of nodes between two consecutive columns. Therefore, the total complexity is $O(NK^2)$. To further improve the efficiency, we carefully design a message-passing scheme with one auxiliary index array of size $N$ to speed up each growing step. Each tree branch is described by a message $\{Root, Length\}$, where $Root$ denotes the temporal index of the tree root and $Length$ is the current length of the growing branch. This message is carried by every current leaf node in a tree and will be passed to its descendants as the tree grows. To determine if a leaf node can grow, instead of checking all of the $K$ nodes in the next column in the trellis, we only check the auxiliary array $B$ times to see whether any of its $B$ descendants exists.

Fig. 4.3 illustrates one growing step from column 312 to 313, using an auxiliary array for speedup. The auxiliary array is essentially a lookup table that tracks the availability of each matching node of 313 and stores the row indices of the matching nodes for tracing back to the trellis. Take one matching node of 312 for example, 927 finds out if it can grow to its descendants (927, 928 and 929) by simply checking the corresponding 3 elements in the auxiliary array. To keep the tree structure, we update the binary flags in the auxiliary array to ensure that each node has only one ancestor. For instance, the binary flags of cells [927-929] are set to 0 after they are taken by node 927 in column 312, so when 312’s next matching node 929 grows, it can only branch to node 931 which is still available. In each step, we need to grow $K$ nodes in the current column, where each node need to look up the auxiliary array $B$ times. Therefore, the complexity of growing one step in the trellis now becomes $O(KB)$, with a neglectable additional $O(2K)$ cost incurred from clearing and re-initializing the auxiliary array. Given $N - 1$ steps in total, the full complexity of the improved forest-growing is now $O(NKB)$, which is more efficient than the previous $O(NK^2)$. 


Figure 4.3. Improved forest-growing step from column 312 to 313, with branching factor $B = 3$. (a) Two columns 312 and 313 in the matching-trellis; (b) The auxiliary array associated with column 313; each element in the first column stores a binary flag indicating whether the corresponding node is available (0 means not). The second column stores the row index of each node in column 313, e.g. 928 is in the 3rd row of 313; (c) The updated auxiliary array after growing from 312 to 313. In (a), the colored pair of numbers next to each node is the branch message \{Root, Length\} to be passed to the descendants during growing, e.g. node 927 belongs to a tree branch whose root is 857 and the current length is 70. When node 927 grows to node 928 in the next column, it updates the message from \{Root = 857, Length = 70\} to \{Root = 857, Length = 71\} and pass it to 928. The three colors denote different branch status: a live branch (yellow), a new branch (purple) and a dead branch (green). See texts for detailed description.

4.2.4. Step 3. Clustering Tree Branches

After forest-growing, we only keep the longest branch to represent each tree. The validity of a tree is determined by the length of its longest branch. All valid trees are output to a
candidate set $\mathcal{T} = \{T_i : |T_i| \geq \lambda\}_{i=1}^l$, where $\lambda$ is the minimum valid length for pruning invalid trees, and $|T_i|$ denotes the length of the longest tree branch of $T_i$. Given the candidate set $\mathcal{T}$, we then progressively merge any two trees $T_i, T_j$ with significant temporal overlaps (set to $3/4$ times the length of the shorter branch) to further reduce the redundancy among trees.

After merging highly overlapped trees, we end up with a smaller set of $M$ recurring instances $\mathcal{T}' = \{V_i\}_{i=1}^M$, where each $V_i$ is described by a message $\{\text{Root, Length}\}$. To cluster these $M$ instances into $G$ event groups, we measure the similarity between any two instances $V_i$ and $V_j \in \mathcal{T}$, again, based on the matching-trellis. Our observation is that if $V_i$ is similar to $V_j$, then $V_j$ should appear in $V_i$’s matching-trellis. The similarity between two instances is hence defined as:

$$s(V_i, V_j) = \frac{1}{2} \left[ \frac{|\text{sim}(V_i, V_j)|}{|V_j|} + \frac{|\text{sim}(V_j, V_i)|}{|V_i|} \right], \quad (4.2)$$

where $|\text{sim}(V_i, V_j)|$ is the length of the longest branch obtained from growing $V_j$ in $V_i$’s matching-trellis. It is notable that $|\text{sim}(V_i, V_j)|$ can be different from $|\text{sim}(V_j, V_i)|$ as the forest-growing is nonsymmetric.

Finally, based on the resulting $M \times M$ similarity matrix, whose element is $s(V_i, V_j)$, we use the normalized cut [44] to cluster these $M$ instances into $G$ groups, where each group corresponds to a recurring event consisting of a number of recurring instances. Besides normalized cut, other advanced clustering methods for time-series data can be applied as well [62] [77].
Table 4.1. Complexity analysis and comparison. The cost of feature extraction is not considered. The parameter $\alpha > 1$ is the approximation factor determined by $\epsilon$ of $\epsilon$-NN query and the correct-retrieval probability $p$ of LSH. For the method in [1], $L$ is a constant depending on the sampling rate of the VPs.

<table>
<thead>
<tr>
<th>Method</th>
<th>Overhead</th>
<th>Pattern Discovery</th>
<th>Total Complexity</th>
<th>Memory Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naive Exhaustive Search</td>
<td>none</td>
<td>$O(N^3)$</td>
<td>$O(N^3)$</td>
<td>$O(N)$</td>
</tr>
<tr>
<td>Self-Similarity Matrix [51]</td>
<td>$O(N^{1+\frac{1}{\alpha}})$</td>
<td>$O(N^2)$</td>
<td>$O(N^2)$</td>
<td>$O(N^2)$</td>
</tr>
<tr>
<td>ARGOS [1]</td>
<td>none</td>
<td>$O(N^2/L)$</td>
<td>$O(N^2/L)$</td>
<td>fixed size</td>
</tr>
<tr>
<td>Basic Forest-Growing</td>
<td>$O(N^{1+\frac{1}{\alpha}})$</td>
<td>$O(NK^2)$</td>
<td>$O(N^{1+\frac{1}{\alpha}} + NK^2)$</td>
<td>$O(NK)$</td>
</tr>
<tr>
<td>Improved Forest-Growing</td>
<td>$O(N^{1+\frac{1}{\alpha}})$</td>
<td>$O(NK)$</td>
<td>$O(N^{1+\frac{1}{\alpha}} + NK)$</td>
<td>$O(NK)$</td>
</tr>
</tbody>
</table>

4.2.5. Efficiency and Scalability

A comparison between our algorithm and other methods is summarize in Table 4.1. Both [51] and our method use LSH to accelerate similarity matching, thus have the same overhead cost. However, our pattern discovery procedure through forest-growing in the matching-trellis is more efficient ($O(NK)$) compared with that in the full $N \times N$ matrix ($O(N^2)$). Moreover, our memory cost is lower than [51], since we only store the best $K$ matches in a $K \times N$ trellis, instead of using a $N \times N$ matrix. This presents a great advantage of applying our method to large databases. As an on-line mining method, [1] can perform real-time for broadcast audio streams. Although only linear or fixed memory is required in practice, the worst-case complexity of [1] is still quadratic.

In summary, the proposed forest-growing method has CPU and memory cost comparable with previous methods that focus on mining exact repeats [1] [51], but with added advantages of handling content and temporal variations. Compared with the basic forest-growing, by using an auxiliary array of length $N$, the improved version further reduces the complexity of pattern discovery from $O(NK^2)$ to $O(NKB)$, which is essentially $O(NK)$ because $B$ is a small constant as is discussed in the following section.
4.3. Discussions of Parameters

4.3.1. Branching Factor $B$

As mentioned before, besides tolerating temporal and local variations, a suitable choice of $B$ can also compensate for the inaccurate matching results caused by LSH. To select a suitable branching factor, we consider the following problem: if there exists a recurring instance of length $L$ in the database $V$, what is the probability that the instance fails to form a tree branch of length $L$ in the matching-trellis due to the missed retrieval by LSH? Suppose the correct retrieval probability of LSH is $p$, given branching factor $B$, the probability of breaking a branch at a given step is:

$$Prob_e = (1 - p)^B,$$  \hspace{1cm} (4.3)

when all the $B$ descendants are missed by LSH, hence the tree branch cannot grow any longer. Therefore, the probability of breaking a potential tree branch of length $L$ is:

$$Prob_b = 1 - (1 - Prob_e)^L = 1 - [1 - (1 - p)^B]^L,$$  \hspace{1cm} (4.4)

when any of the $L$ steps breaks.

Hence given $p$, a large branching factor $B$ decreases the break probability $Prob_b$, which consequently decreases the probability of missed detection of repetitions. In addition, a potential tree branch can survive large content and temporal variations with a large $B$.

To investigate how the branching factor $B$ influences the branch length $L$ and its breaking probability $Prob_b$, we show the relations between $L$ and $Prob_b$ in Fig. 4.4 with varying $B$. 
Here the correct retrieval probability $p = 0.9$ is chosen as the default parameter used for LSH. As can be seen in Fig. 4.4, if only strict continuous growing is allowed when expanding trees ($B = 1$), the breaking probability $Prob_t$ increases very fast with respect to the branch length $L$. For example, when $L = 100$, we have $Prob_t \approx 1$. This means that it is very likely a repetition of length $L = 100$ will be missed due to LSH. In fact, the break probability is already large enough even for short repetitions, e.g., $Prob_t \approx 0.5$ when $L = 7$. As expected, when more branches are allowed, it is less likely that a continuous branch will break because of missed retrieval. Specifically, when $B = 5$, the breaking probability $Prob_t$ is still small (around 0.1) even for long branches of $L = 10,000$.

Although a large $B$ increases the power of handling variations and noises, on the other hand, it may introduce random effects into the tree growing process. An extreme case is when $B = N$, where every possible path in the forest can be a tree branch, which generates meaningless results. In addition, the computational cost of growing the forest ($O(NKB)$) will increase as $B$ increases. In our experiments, we select $B \leq 5$.

4.3.2. Nearest-Neighbor Search Parameter $\epsilon$

It is important to select an appropriate $\epsilon$ for approximate-NN search as it determines the quantity of the best matches retrieved hence the size of the trellis as well. An improper choice of $\epsilon$ will result in either insufficient number of retrieved NNs or an excessive number of NNs [78]. In practice, a small $\epsilon$ is preferred for large datasets and memory-constrained conditions, so that the $N \times K$ trellis is of limited size and can be loaded into the main memory easily. On the other hand, a larger $\epsilon$ retrieves more NN candidates thus reduces the chance of missed retrieval by LSH.
Figure 4.4. How the branching factors $B$ influences the relationship between the branch length $L$ and its breaking probability $Prob_t$. The curve is drawn based on Eq. 4.4, with $p = 0.9$.

Considering both requirements, instead of selecting a constant $\epsilon$ in Eq. 4.1, we set it as a data-dependent parameter:

$$\epsilon = \mu - \tau \times \sigma,$$

where $\mu$ and $\sigma$ are the estimated mean and standard deviation of the pair-wise distance $d(S_i, S_j)$, and $\tau$ is the parameter controlling the threshold. Under the assumption of a Gaussian distribution, if we select $\tau = 2$, we will retrieve around 2.2% VPs as the NNs. In such a case, we have $K \approx 0.022N << N$. 
4.3.3. Minimum Length of Valid Repetition $\lambda$

As mentioned in Section 4.2.4, for a discovered repetition to be valid, it must have a minimum length of $\lambda$. In the matching-trellis, $\lambda$ determines the minimum length of valid tree branches. On one hand, to avoid discovery of trivial short repetitions that are caused by noise or randomness, we require $\lambda$ to be long enough to filter these pseudo branches. On the other hand, $\lambda$ should not be too long so we will not miss valid short repetitions.

To help select appropriate $\lambda$, we estimate the probability in forming short repetitions due to the randomness in the matching-trellis. Considering a random matching-trellis of size $N \times K$, where each column, i.e., a matching set, contains $K$ nodes selected randomly from the $N$ candidates. We are interested in the probability of generating a tree branch of length $L$ in this random trellis. The selection of $\lambda$ should guarantee low probability of finding such repetitions due to random effects.

Each of the $N \times K$ nodes in the trellis has the potential to serve as a root and grow a new tree, if it is not taken by other trees yet. If one node can start a new tree, then in the random trellis, the probability that it can grow to the next step is:

$$\text{Prob}_g = 1 - \left( 1 - \frac{B}{N} \right)^K,$$

(4.6)

where $(1 - \frac{B}{N})^K$ is the probability that none of $K$ nodes in the next column can be its descendant. Thus the probability for a random tree branch to reach length $L$ (i.e. grows $L - 1$ steps) is:

$$\text{Prob}_L = \text{Prob}_g^{(L-1)}.$$

(4.7)

In the entire trellis, there are in total $(N - L + 1) \times K$ potential nodes that can lead to a tree of length $L$. Assuming these trees are independent of each other, the probability that
none of the \((N - L + 1) \times K\) nodes can generate a tree of length \(L\) is:

\[
Prob_T = (1 - Prob_L)^{(N-L+1)\times K} = \left[1 - \left(1 - \left(1 - \frac{B}{N}\right)^K\right)^{(L-1)}\right]\left(1 - \left(1 - \left(1 - \frac{B}{N}\right)^K\right)^{(N-L+1)\times K}\right).
\] (4.8)

To further investigate the relations between \(Prob_T\) and \(L\), we plot in Fig. 4.5 how \(Prob_T\) changes with respect to \(L\), with two sets of pre-specified \(B\) and \(N\). It is surprising that \(Prob_T\) is sensitive to a fixed threshold of \(L\) (e.g. around 10) under given \(B\) and \(N\). When \(L\) is longer than the threshold, it is highly likely \((Prob_T \to 1)\) that a continuous branch of length \(L\) will not appear in a random matching-trellis, while when \(L\) is smaller than the threshold, it is highly likely \((Prob_T \to 0)\) that a continuous branch will appear due to random effects.

During our forest-growing, we make sure that each discovered tree is of sufficient length \(|T_i| \geq \lambda\) under the specific parameters \(N\), \(K\) and \(B\) to rule out those formed at random.

4.4. Experiment 1: Recurring event discovery from news video

To evaluate the efficiency of our method in mining exact repetitions, we use broadcast news video as the test set, which contains highly repetitive content including commercials, special program lead-in and lead-out clips, and anchor person shots. Considering that commercials are exact repeats and appear frequently, we use them as the ground truth to evaluate the results of our algorithm. The video database contains 22 streams of half-hour ABC news videos collected from the TRECVID dataset [66]. All these half-hour segments are combined into one video sequence with a total length of 10.5 hours. The frame rate is 30 fps and the frame size is 352 \times 240 or 352 \times 264. For evaluation, we collect a set of 56 repetitive clips from the 10.5-hour video dataset as the benchmark data, including 55 commercials and 1 program.
lead-out clip. The lengths of the commercials vary from 15 to 60 seconds and the program lead-out clip is between 10 and 11 seconds. Each repetitive clip has 2 to 8 recurrences in the database and we manually label the ground truth. These 56 repetitive clips have in total 189 recurring instances in the 10.5-hour dataset.

In our implementation, each video primitive (VP) $S_i$ is of 4.8 seconds with a sampling interval of 0.4 seconds. Thus each VP has an overlap of 4.4 seconds with its temporal neighbors. In total, the 10.5-hour video database generates $N = 94,008$ VPs. We also set the temporal distance $\tilde{N} = 300$ VPs to filter neighboring video segments in the matching set. The minimum length of a valid repetition is set to $\lambda = 15$ VPs as suggested in Fig. 4.5. As a result, a valid repetition is at least $4.8 + (15 - 1) \times 0.4 = 10.4$ seconds. The experiment
is performed on a standard pentium-4 3.19 GHz PC with 1 GB RAM. The algorithm is implemented in C++.

4.4.1. Visual signature extraction

For exact repeat mining, it is desirable that the visual features are robust under video coding variations, such as compression rate, frame size and color format changes. Moreover, the visual features should also be unique enough to identify different videos. To this end, we follow the feature extraction in [56] [60] and use two types of compact visual signatures to characterize each video segment $S_i$: (1) color signature and (2) spatial signature. For each $S_i$, we concatenate its 3 color histograms and 3 spatial pattern histograms into a single normalized histogram $F$. Since each individual histogram is of 24-dimension, a VP is characterized by a feature vector: $F \in \mathbb{R}^d$ ($d = 24 \times 6 = 144$). As all of the signatures can be extracted from the MPEG compressed video data directly [55], they cost fewer CPU time.

4.4.2. Efficiency

Table 4.2 summarizes the computational costs of (1) building the matching-trellis and (2) growing the forest. Only CPU cost is counted while file I/O cost is not included. Overall, our method proves to be very efficient and can mine a 10.5-hour video in about 7.16 minutes. It is notable that the forest-growing step is extremely efficient and only takes 7.5 seconds to grow all the continuous paths. Therefore, the major computational cost comes from the overhead of building the matching-trellis, which takes 98.25% of the total CPU time, even with the help of LSH. However, compared with computing a full $N \times N$ similarity matrix, the overhead has already been largely reduced.
To estimate the parameters for $\epsilon$-NN query, we randomly select 1% VPs from the whole dataset. The estimated mean of pair-wise distances is $\mu = 0.269$ and the estimated standard variance is $\sigma = 0.060$. Considering that commercials are exact repeats, we set a strict matching criterion with $\epsilon = \mu - 3\sigma = 0.089$. The $\epsilon$-NN query is fast by applying LSH with a small $\epsilon = 0.089$. In our database of size $N = 94,008$, the average CPU time for each $\epsilon$-NN query is only 5 milliseconds. Since the average size of the matching set $M_S$ is 24, the size of the matching-trellis is approximately $K \times N = 24 \times 94,008$, which is a much more compact representation compared with the $N \times N$ self-similarity matrix, as $K/N = 0.000255$.

Table 4.2. Computational Cost with $\epsilon = \mu - 3\sigma$ and $B = 3$.

<table>
<thead>
<tr>
<th></th>
<th>Complexity</th>
<th>CPU Cost</th>
</tr>
</thead>
<tbody>
<tr>
<td>build the matching-trellis</td>
<td>$O(N^{1+\frac{1}{\alpha}})$</td>
<td>422.1 sec</td>
</tr>
<tr>
<td>forest-growing</td>
<td>$O(NK)$</td>
<td>7.5 sec</td>
</tr>
<tr>
<td>total cost</td>
<td>$O(N^{1+\frac{1}{\alpha}} + NK)$</td>
<td>429.6 sec</td>
</tr>
</tbody>
</table>

In table 4.3, we compare the improved forest-growing method with the basic one. By using an auxiliary array, the forest-growing procedure is largely accelerated. In terms of CPU cost, the improved forest-growing method is around 18 times faster than the basic one.

Table 4.3. Comparison of basic forest-growing and improved forest-growing ($K = 24$ and $N = 94008$).

<table>
<thead>
<tr>
<th></th>
<th>Basic ($B = 2$)</th>
<th>Imp. ($B = 2$)</th>
<th>Imp. ($B = 3$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Complexity</td>
<td>$O(NK^2)$</td>
<td>$O(2NK)$</td>
<td>$O(3NK)$</td>
</tr>
<tr>
<td>CPU cost</td>
<td>101.4 sec</td>
<td>5.5 sec</td>
<td>7.5 sec</td>
</tr>
</tbody>
</table>

4.4.3. Performance

To evaluate the performance of our forest-growing algorithm, we treat the 56 repetitive clip set as the benchmark set, which corresponds to 189 repetitive instances in total. We evaluate the performance of our method by the recall and precision. The recall score is defined as the
percentage of the total 189 instances retrieved. It is measured in terms of the video length, where the percentage is calculated by the number of frames that are correctly discovered by the algorithm. This recall score reflects how many of the 189 instances are finally detected. On the other hand, because we do not have the ground truth of all possible repetitions, which includes not only commercials, but also anchor person shots and still images, it is difficult to provide the accurate precision score. So instead we estimate the precision by randomly picking 20 discovered recurring instances and manually checking whether they are really repetitions. The precision score is the percentage of correct ones from the 20 discovered instances.

We compare different branching factors \((B = 2 \text{ and } B = 3)\) in table 4.4 to see how they influence the performance. As expected, a larger branching factor leads to more trees in the forest, as well as more valid branches. For example, when \(B = 2\), we get 2086 trees and 35084 valid branches, whose lengths are longer than the minimum valid length \(\lambda\). In comparison, when \(B = 3\), more trees (2905) and much more valid branches (94017) are returned. The average number of valid branches per tree almost doubled when we allow trees to grow more flexibly by increasing the branching factor from \(B = 2\) to \(B = 3\). Moreover, the results of the benchmark data (189 commercial instances) shows that the larger branching factor \(B = 3\) also leads to a higher recall compared with \(B = 2\) in mining video repetitions. This result validates our theoretical analysis in Fig. 4.4. On the other hand, the precision drops as the branching factor increases.

We further compare different selections of \(\epsilon\) for \(\epsilon\)-NN search, as shown in table 4.5. A larger \(\epsilon\) \((\epsilon = \mu - 2\sigma)\) for \(\epsilon\)-NN query results in a better recall. But it also brings a worse precision score than using a stricter criterion \(\epsilon = \mu - 3\sigma\). As expected, the computational
Table 4.4. Comparison of different branching factors ($\epsilon = \mu - 3\sigma$).

<table>
<thead>
<tr>
<th></th>
<th>$B = 2$</th>
<th>$B = 3$</th>
</tr>
</thead>
<tbody>
<tr>
<td># valid branches</td>
<td>35084</td>
<td>94017</td>
</tr>
<tr>
<td># trees</td>
<td>2086</td>
<td>2905</td>
</tr>
<tr>
<td>ave. # valid branches per tree</td>
<td>16.82</td>
<td>32.36</td>
</tr>
<tr>
<td># instances (merged trees)</td>
<td>939</td>
<td>926</td>
</tr>
<tr>
<td>recall of benchmark data</td>
<td>55.0%</td>
<td>84.3%</td>
</tr>
<tr>
<td>estimated precision</td>
<td>95.0 %</td>
<td>90.0 %</td>
</tr>
</tbody>
</table>

time is also much longer with larger $\epsilon$, because more NNs are retrieved. Hence a much larger matching-trellis is built and needs to discover.

Table 4.5. Comparison of different criteria in selection $\epsilon$ for NN-search ($3\sigma$-criterion v.s. $2\sigma$-criterion). Branching factor $B = 3$.

<table>
<thead>
<tr>
<th></th>
<th>$\epsilon = \mu - 3\sigma$</th>
<th>$\epsilon = \mu - 2\sigma$</th>
</tr>
</thead>
<tbody>
<tr>
<td>K</td>
<td>24</td>
<td>430</td>
</tr>
<tr>
<td>building trellis</td>
<td>422.1 sec</td>
<td>2741.9 sec</td>
</tr>
<tr>
<td>forest-growing</td>
<td>7.5 sec</td>
<td>191.0 sec</td>
</tr>
<tr>
<td># valid branches</td>
<td>94017</td>
<td>706462</td>
</tr>
<tr>
<td># trees</td>
<td>2905</td>
<td>18465</td>
</tr>
<tr>
<td># instances (merged trees)</td>
<td>926</td>
<td>3367</td>
</tr>
<tr>
<td>recall of benchmark data</td>
<td>84.3%</td>
<td>98.3%</td>
</tr>
<tr>
<td>estimated precision</td>
<td>90.0 %</td>
<td>70.0 %</td>
</tr>
</tbody>
</table>

4.5. Experiment 2: Mining Recurring Patterns from Human Motion

To validate that our algorithm can handle more general spatio-temporal variations, we further test our algorithm on motion captured human motion data to discover recurring motion patterns, which exhibit representative spatio-temporal dynamics in human motions. In the case of human dance data, they are typical dance moves that appear repetitively in a certain type of dance. Our test dataset consists of 32,260 frame human motion data from Carnegie Mellon University Graphics Lab mocap database [79]. It includes 15 motion sequences from 5 different motion categories: break dance, acrobatics, Indian dance, Michael
Jackson-style dance and salsa. These motions are captured in different mocap sessions and performed by different subjects. Since the same type of dance can be performed at different speeds and by different people, instances of the same motion event can vary significantly due to the differences in the skeletal/spatial configurations and speeds. Thus compared with mining exact repeats from videos, mining recurring motion patterns is a much more challenging problem.

We completely rely on our mining algorithm to discover motion patterns without providing any a priori knowledge of the underlying motion data. This experiment is performed on a standard Pentium-4 2 GHz PC with 2 GB RAM.

4.5.1. Feature extraction and similarity matching

Given a human motion sequence $\mathcal{V}$, we can represent it as a sequence of human poses $\mathcal{V} = \{\mathbf{S}_i\}_{i=1}^N$. Each $\mathbf{S}_i$ is an individual frame and the skeletal pose is characterized by its root position and joint orientations. By treating each pose as a primitive, we measure the similarity between two poses $\mathbf{S}_i$ and $\mathbf{S}_j$ as the weighted Euclidean distance between the quaternion representations of the two poses:

$$d(\mathbf{S}_i, \mathbf{S}_j) = \sum_{k=1}^{J} w_k \|q_{i,k} - q_{j,k}\|,$$  \hfill (4.9)

where $\|\cdot\|$ denotes the Euclidean distance; $w_k$ is the weight of joint $k$; $q_{i,k} \in S^3$ is the unit quaternion representation of the orientation of joint $k$ with respect to its parent joint in frame $i$; $J$ is the total number of joints in the skeletal representation ($J = 29$ in our test set). As in [64], $w_k$ is set to 1 for important joints like the shoulders, elbows, hips, knees,
pelvis, lower back and upper back, whereas $w_k$ is set to 0 for less important joints, like the toes and wrists.

### 4.5.2. Results

Considering the large variations that may exist in human motion data, we choose a less strict $2\sigma$ criterion for selecting $\epsilon$, which helps to retrieve similar but not necessarily identical poses given a query. The estimated mean and standard variance of pair-wise distances are $\mu = 3.23$ and $\sigma = 0.94$, respectively, thus $\epsilon = \mu - 2\sigma = 1.35$. The average size of the matching set is $K = 1163$. Branching factor is set to $B = 5$, in order to accommodate the possibly large temporal variations in human motions. When building the match trellis, we filter temporally nearby $\hat{N} = 300$ frames and set the minimum valid length $\lambda = 60$ to eliminate too short motion fragments.

It takes 241 seconds to build a matching-trellis of size $32260 \times 1163$, and 213 seconds to find all continuous paths using our forest-growing algorithm. In total, 2,527,496 valid branches are discovered, from which we obtain 6,944 raw paths by only picking the longest branch from each tree. Then we iteratively merge raw paths with temporal overlaps greater than 3/4 and finally obtain 186 recurring instances. The average, maximum and minimum length of the instances are 183, 839 and 60, respectively. These 186 recurring instances are then clustered into 30 event groups using normalized-cut based on the similarity matrix defined in Eq. 4.2. In Fig. 4.6, we show an event cluster containing 3 similar human dance sequences.

A person with animation expertise checks the effectiveness of our algorithm by visually identifying if instances in each cluster are indeed similar patterns (temporal-spatial variations in the motions are allowed). Table 4.6 presents the instance distribution in each individual
cluster, labeled by the animation expert. For a cluster containing instances from different motion categories, we take the majority as the ground truth and evaluate the error rate by the percentage of mis-clustered instances in this cluster. The final error rate is defined as

Table 4.6. Discovery of recurring motion patterns. The errors are highlighted. The total error rate for clustering is 11.29%.

<table>
<thead>
<tr>
<th>Acrobatics</th>
<th>M.J.</th>
<th>Salsa</th>
<th>Indian</th>
<th>Break</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>C1</td>
<td>0</td>
<td>11</td>
<td>0</td>
<td>0</td>
<td>11</td>
</tr>
<tr>
<td>C2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>C3</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>C4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>C5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>C6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>C7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>C8</td>
<td>0</td>
<td>3</td>
<td>1</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>C9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>C10</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>C11</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>C12</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>C13</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>5</td>
<td>1</td>
</tr>
<tr>
<td>C14</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>8</td>
</tr>
<tr>
<td>C15</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>16</td>
</tr>
<tr>
<td>C16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>C17</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>C18</td>
<td>0</td>
<td>0</td>
<td>6</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>C19</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>C20</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>C21</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>C22</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>10</td>
</tr>
<tr>
<td>C23</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6</td>
</tr>
<tr>
<td>C24</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>C25</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>C26</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>C27</td>
<td>0</td>
<td>4</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C28</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3</td>
</tr>
<tr>
<td>C29</td>
<td>4</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>C30</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td><strong>sum</strong></td>
<td><strong>7</strong></td>
<td><strong>25</strong></td>
<td><strong>16</strong></td>
<td><strong>48</strong></td>
<td><strong>90</strong></td>
</tr>
</tbody>
</table>
the weighted sum of the error rates of all 30 clusters, where the weights are based on the number of recurring instances in each cluster. Judged by the expert, 22 of the 30 event classes are 100% accurate, while 8 out of the 30 events contain instances from different motion categories. The resulting error rate is 11.29%.

Interestingly, in one of the erroneous clusters (C27 in Table 4.6), where the majority belong to Michael Jackson style dance (4 out of 6), the salsa instance and the break dance instance that are mis-clustered into this event actually share a full-body rotation pattern that is similar to the Michael Jackson dance instances.

4.6. Conclusions

Because recurring events may not be exact repeats, it poses extreme challenges to algorithms that try to discover and detect such events automatically. To overcome the large computational cost and to handle content and temporal variations, we translate the recurring pattern mining problem into finding continuous paths in the matching-trellis. The proposed forest-growing algorithm is efficient and of only sub-quadratic complexity with respect to the database size. By introducing a branching factor when growing the forest, it is able to handle content and temporal variations, as well as the misses caused by the approximate search using LSH, which is critical for reducing the computational cost of finding best matches.

The experiment with news video demonstrates the efficiency of our method, which can mine a 10.5 hour video within 8 minutes, upon feature extraction. The other experiment on human dance video validates that our method is capable of handling large content and temporal variations, including pose changes and non-uniform time warping in human motion
patterns. Our forest-growing algorithm is applicable to different underlying feature representations and therefore can be easily extended to other types of time-series data for mining recurring patterns.
The 1st recurring instance.

The 2nd recurring instance.

The 3rd recurring instance.

Figure 4.6. A discovered recurring event consists of three motion instances. Each instance is a segment in the long motion sequences; the figures are sampled every 10 frames. All of the three instances belong to the same event in break dance, but are of various length and dynamics. More results of the discovered recurring motion patterns can be seen at http://www.ece.northwestern.edu/~jyu410/motionmotifs/.
CHAPTER 5

Mining Compositional Features for Pattern Classification

Given a pool of decision stumps $\Omega = \{f_i\}$, where each $f_i$ is called a feature primitive, our goal is to discover strong enough compositional features $F = \{f_i\}$ for boosting (Fig. ??). Each compositional feature $F$ contains one or more feature primitives, and its decision is determined by the responses of feature primitives. To balance between generalization and discrimination abilities, we require $F$ to be both (1) descriptive features (i.e. high frequency in positive training data), and (2) discriminative features (i.e. high accuracy in prediction). Although feature primitives can be rather weak individually, we show that an appropriate composition of them can have guaranteed discriminative power with a guaranteed bound of training error. Compared with boosting decision stumps, boosting these higher-order rules can lead to faster convergence in training as well as better generalization if the decision boundary function is not in an additive form of original feature primitives [80].

To reduce the combinatorial cost in searching for compositional features [81], we apply data mining methods such as frequent itemset mining (FIM) for pattern discovery. Due to their computational efficiency, data mining methods are becoming popular in many vision applications, including visual object detection [82], classification [83] [84] [85] and image pattern discovery [16]. After the compositional features are discovered, we boost them by applying a multi-class AdaBoost method: stagewise additive modeling with exponential loss (SAMME) [86]. SAMME directly handles the $K$-class problem by building a single $K$-class classifier, instead of $K$ binary ones. The solution of SAMME is consistent with the Bayes
classification rule, thus it is optimal in minimizing the misclassification error. Experimental results of both simulated data and a challenging 10-class visual event recognition problem validate the advantages of boosting compositional features.

5.1. Induced Transactions for Data Mining

We consider a $K$-class classification problem. Suppose we are given a training dataset containing $N$ samples of $K$ classes: $\mathcal{D}_N = \{x_t, c_t\}_{t=1}^N$, where $x_t \in \mathbb{R}^P$ denotes the feature vector and $c_t \in \{1, 2, \ldots, K\}$ is the label of $x_t$. The task is to find a classifier $g(\cdot): x \rightarrow c$ from the training data, such that given a new query sample $x$, we can assign it a class label $c \in \{1, 2, \ldots, K\}$. Instead of using the raw features $x$ directly to estimate $c$, we consider a collection of induced binary features $\{f_1, f_2, \ldots, f_P\}$ where each $f_i: x \rightarrow \{0, 1\}$ is a feature primitive. For example, $f_i$ can be a decision stump:

$$f_i(x) = \begin{cases} f_i^+ & \text{if } x(i) \geq \theta_i \\ f_i^- & \text{if } x(i) < \theta_i \end{cases}, \quad (5.1)$$

or a decision stump when only positive response is considered:

$$f_i(x) = \begin{cases} f_i & \text{if } x(i) \geq \theta_i \\ \emptyset & \text{if } x(i) < \theta_i \end{cases}. \quad (5.2)$$

Here $x(i)$ is value of $x$ in the $i_{th}$ dimension, and $\theta_i \in \mathbb{R}$ is the quantization threshold for $f_i$. We call $f_i$ the feature item associated with the feature primitive $f_i$.

Without loss of generality, we use the decision stump considering positive response only (Eq. 5.2) for illustration. Given a collection of $P$ features, we have an item vocabulary $\Omega = \{f_1, f_2, \ldots, f_P\}$ containing $P$ items. As illustrated in Fig. 5.1, now a training sample $x \in \mathbb{R}^P$ can be transferred into a transaction.
\[ T(x) = \{f_1(x), f_2(x), \ldots, f_P(x)\} \subseteq \Omega, \]

according to the responses of \( P \) feature primitives. The induced transaction dataset \( T = \{T_t\}_{t=1}^N \) contains a collection of \( N \) training samples, where each \( T \) corresponds to a data sample \( x \). By transforming continuous features \( x \in \mathbb{R}^P \) into discrete transactions, we can perform traditional data mining algorithm, such as frequent itemset mining. In Sec. 5.2 and Sec. 5.3, we discuss how to take advantage of efficient data mining method to search for informative features for classification.

![Figure 5.1. Illustration of the induced transaction. By partitioning the feature space into sub-regions through decision stumps \( f_A \) and \( f_B \), we can index the training samples in terms of the sub-regions they are located. Only positive responses are considered. For example, a transaction of \( T(x) = \{f_A, f_B\} \) indicates that \( f_A(x) > \theta_A \) and \( f_B(x) > \theta_B \).](image-url)
5.2. Mining Compositional Features

For each feature primitive \( f_i \), we can use it to predict the class label. A \textit{primitive classification rule} is thus of the form:

\[ f_i(x) = f_i \implies \hat{c}(x) = k, \]

where \( k \in \{1, 2, ..., K\} \), and \( \hat{c}(x) \) is the predicted label of \( x \). Since a classification rule based on an individual \( f \) is usually of low accuracy, it is of our interests to find compositional feature \( F = \{f_i\} \subseteq \Omega \) which can be more accurate. Given a compositional feature \( F \), we define its classification rule as:

\[ F(x) = \begin{cases} k & \text{if } F \subseteq T(x) \\ 0 & \text{otherwise} \end{cases}, \]

where \( k \in \{1, 2, ..., K\} \) is the predicted class; \( F(x) = 0 \) implies that \( F \) cannot make decision on \( x \).

Following the terms in data mining literature, we call \( F \) as a \textit{feature item-set}. Given an itemset \( F \), the transaction \( T_t \) which includes \( F \) is called an \textit{occurrence} of \( F \), \textit{i.e.}, \( T_t \) is an occurrence of \( F \), if \( F \subseteq T(x_t) \). We denote by \( T(F) \) the set of all occurrences of \( F \) in \( T \), and the \textit{frequency} of an itemset \( F \) is denoted by:

\[ \text{frq}(F) = |T(F)| = |\{t : F \subseteq T(x_t)\}|. \]

Considering it is important to evaluate the quality of a compositional classification rule, we first give an analysis of the perfect rule.
Definition 6. perfect classification rule

A compositional classification rule \( \mathcal{F}^* \) is perfect if \( \exists k \in \{1, 2, \ldots, K\} \), such that

\[
\text{descriptive : } P(\mathcal{F}^*(x) = k | c(x) = k) = 1 \tag{5.4}
\]
\[
\text{discriminative : } P(c(x) = k | \mathcal{F}^*(x) = k) = 1 \tag{5.5}
\]

In Definition 6, we specify two conditions for the perfect rule \( \mathcal{F}^* \), where \( P(\mathcal{F}^*(x) = k | c(x) = k) = 1 \) is the descriptive ability, while \( P(c(x) = k | \mathcal{F}^*(x) = k) = 1 \) is the discriminative ability. Since its classification result is the same as the ground truth, \( \mathcal{F}^* \) is the best possible rule for class \( k \). However, exhaustive search for \( \mathcal{F}^* \) is computationally demanding due to the combinatorial complexity. Because each \( f_i \) can generate two possible outcomes: \( f_i \) or \( \emptyset \), the total number of all possible classification rules is \( 2^{|\Omega|} \). Thus efficient search methods are required to make the feature selection process computationally feasible. Even worse, such a perfect rule \( \mathcal{F}^* \) may not always exist in the case of noisy training data \([87]\), where positive and negative samples are not perfectly separable. In such a case, we need to sacrifice the strict conditions of selecting optimal \( \mathcal{F}^* \) for sub-optimal ones. In other words, instead of searching for perfect rule \( \mathcal{F}^* \), we search for a collection of weaker rules \( \Psi = \{\mathcal{F}_i\} \).

With its justification later, we define the sub-optimal compositional rule in Definition 7.

Definition 7. \((\lambda_1, \lambda_2)\)-compositional rule

A compositional feature \( \mathcal{F} \subset \Omega \) is called \((\lambda_1, \lambda_2)\)-compositional rule if \( \exists k \in \{1, 2, \ldots, K\} \), such that:

\[
\text{sup. : } P(\mathcal{F}(x) = k) \geq \lambda_1
\]
\[
\text{conf. : } P(c(x) = k | \mathcal{F}(x) = k) \geq \lambda_2 \times P(c(x) = k)
\]
The first condition requires that \( \frac{freq(F)}{N} \geq \lambda_1 \), which is the support requirement in mining frequent patterns [21]. A rule of low support covers few training samples. Such a classification rule has limited ability to generalize, even if it can predict accurately on few number of training samples. The second condition requires that the rule is accurate enough for prediction, such that most covered samples are correctly classified. This condition corresponds to the confidence of a rule in data mining literature [21]. Different from traditional data mining methods which usually set a fixed confidence threshold, we consider the class prior to handle imbalanced training data. A weak rule \( F \) that satisfies both conditions are viewed as useful rules for future use.

To further justify our criteria of \( (\lambda_1, \lambda_2) \)-compositional rule, we develop Definition 7 into two weak conditions:

\[
P(F(x) = k | c(x) = k) \geq \lambda_2 \times P(F(x) = k), \tag{5.6}
\]

\[
P(c(x) = k | F(x) = k) \geq \lambda_2 \times P(c(x) = k), \tag{5.7}
\]

where Eq. 5.6 is obtained because

\[
P(F(x) = k | c(x) = k) = \frac{P(F(x) = k, c(x) = k)}{P(c(x) = k)} \geq \frac{P(F(x) = k)\lambda_2 P(c(x) = k)}{P(c(x) = k)} = \lambda_2 P(F(x) = k).
\]

Comparing the conditions for perfect rule (Definition 6) with Eq. 5.6 and Eq. 5.7, we can see that weak rules in Definition 7 only need to satisfy weak descriptive and discriminative conditions, thus they are sub-optimal features compared with perfect feature \( F^* \).
We further notice that the two requirements in Eq. 5.6 and Eq. 5.7 are actually an equivalent one:

\[ \frac{P(F(x) = k, c(x) = k)}{P(c(x) = k)P(F(x) = k)} \geq \lambda_2, \]

given \( P(F(x) = k) \geq \lambda_1 \). When \( \frac{P(F(x) = k, c(x) = k)}{P(c(x) = k)P(F(x) = k)} = 1 \), it indicates independent events \( c(x) = k \) and \( F(x) = k \). In order to make sure \( F \) is informative for prediction (e.g. performing better than random guess), we require \( \lambda_2 > 1 \). The other parameter \( 0 < \lambda_1 \leq 1 \) controls the support of a rule, which influences the generalization ability of the rule.

Moreover, according to Eq. 5.6, we need \( \lambda_2 P(F(x) = k) \leq 1 \). Since \( P(F(x) = k) \geq \lambda_1 \), we have

\[ \lambda_1 \leq P(F(x) = k) \leq \frac{1}{\lambda_2}, \quad (5.8) \]

which indicates that qualified rules are those of mild-frequency. This actually explains why we need to discard the most common and uncommon words in the “bag-of-word” approach [15]. Here, it says that we should discard not only common and uncommon words, but also common and uncommon word combinations. As we can see in Eq. 5.6 and Eq. 5.7, such “word-combinations” of mild frequency are informative features for classification.

Based on Eq. 5.8, we further have \( \lambda_1 \lambda_2 \leq 1 \). Let \( r_k = P(c(x = k)) \), we also have \( \lambda_2 \leq \frac{1}{r_k} \) since we need \( \lambda_2 P(c(x) = k) \leq 1 \) in Eq. 5.7. Combining all results, we obtain the conditions for feasible parameters \( \lambda_1 \) and \( \lambda_2 \):

**Proposition 1. feasible parameters of data mining**

The following requirements must be satisfied to avoid mining non-informative or an empty set of \((\lambda_1, \lambda_2)\)-compositional rules according to Definition 7.

\[ 0 < \lambda_1 \leq \frac{1}{\lambda_2} < 1 < \lambda_2 \leq \min\left\{ \frac{1}{\lambda_1}, \frac{1}{r_k} \right\}. \quad (5.9) \]
Eq. 5.9 thus gives the guidance in selecting $\lambda_1$ and $\lambda_2$ for effective data mining, which avoids mining in vain for compositional features.

Based on Proposition 1, we present the major theoretical result in this paper in Theorem 2, where we show that $\lambda_1$ and $\lambda_2$ can bound the training error of the $(\lambda_1, \lambda_2)$-compositional rule $\mathcal{F}$.

**Theorem 2. training error bound of $(\lambda_1, \lambda_2)$-comp rule**

For a $(\lambda_1, \lambda_2)$-compositional rule $\mathcal{F}$ predicting for the $k_{th}$ class, its training error of class $k$ is upper bounded by:

$$\epsilon_{\mathcal{F}} \leq \frac{1}{\lambda_2} - \lambda_1 \lambda_2 r_k = B_{\epsilon_{\mathcal{F}}},$$

(5.10)

where $r_k = P(c(x) = k)$ denotes the prior of class $k$ and $\epsilon_{\mathcal{F}} = P(\mathcal{F}(x) \neq c(x) | \mathcal{D}_N)$ is the empirical error on training data $\mathcal{D}_N$. Specifically, the upper bound is tight, i.e. equality holds in Eq. 5.10, if (1) the two equalities hold in Definition 7 and (2) $\lambda_1 \lambda_2 = 1$. Moreover, when $\lambda_1 = \frac{1}{\lambda_2} = r_k$, we have $\epsilon_{\mathcal{F}} = 0$.

The prove of Theorem 2 is in the Appendix. As illustrated in Fig. 5.2, Theorem 2 states that given a frequent pattern $\mathcal{F}$, we can upper bound its training error on class $k$ by its support ($\lambda_1$) and confidence of the association rule regarding to class $k$ ($\lambda_2$). When $\lambda_1 \lambda_2 = 1$, the bound is tight.

In mining $(\lambda_1, \lambda_2)$-compositional rules in Definition 7, we perform a two-step method in Alg. 4. First, we perform through frequent itemset mining (FIM) algorithms to find compositional features with high support. After that we filter non-discriminative rules that do not satisfy the confidence condition. As each $(\lambda_1, \lambda_2)$-compositional feature can also be a discriminative classification rule according to Eq. 5.6 and Eq. 5.7, Alg. 4 finally ends up
Feasible and desired regions for data mining: \( r_k = 0.50 \)

Figure 5.2. Feasible and desired regions for data mining. We simulate the upper bound \( B_{\varepsilon_F} = \frac{1}{\lambda_2} - \lambda_1 \lambda_2 r_k \), with a specific choice of \( r_k = 0.5 \). The shading region corresponds to \( 0 \leq B_{\varepsilon_F} \leq 1 \). As shown in the color bar, the darker the shade, the smaller the upper bound of \( \varepsilon_F \) and thus are more favorable for data mining. The shading region above the red curve (\( \lambda_1 \lambda_2 > 1 \)) is infeasible for data mining since no rule exists there according to Proposition 1. The region below the red curve (\( \lambda_1 \lambda_2 \leq 1 \)) and between two red lines (\( 1 \leq \lambda_2 \leq \frac{1}{r_k} \)) is the feasible choice of data mining parameters \( \lambda_1 \) (support) and \( \lambda_2 \) (confidence). The region below the red curve while above the blue curve (\( B_{\varepsilon_F} = 0.25 \)) is the desired region where we can get rules with \( \varepsilon_F \leq 0.25 \). At the optimal point \( \lambda_1 = \frac{1}{\lambda_2} = r_k \), where the red curve (tight bound of training error) meets the boundary \( \varepsilon_F = 0 \), we get minimum training error and thus the perfect classification rule.

with a collection of sub-optimal classification rules whose training error is upper bounded according to Theorem 2.

Although it is impossible to search for perfect compositional features directly due to the combinatorial complexity, it is computationally efficient to perform such a two-step
method in Alg. 4, by taking advantage of the FIM algorithms. Specifically, FIM algorithms take advantage of the monotonic property of frequent itemsets (Apriori algorithm) or applying a prefix-tree structure storing compressed information of frequent itemsets (FP-growth algorithm), in order to find them efficiently [21]. In this paper we apply the FP-growth algorithm to implement closed-FIM [39] for discovering frequent patterns.

**Algorithm 4: Mining Compositional Rules**

| input | Training dataset $D = \{T_i, c_i\}_{i=1}^{N}$, where $T_i \subseteq \Omega$, $c_i \in \{1, 2, ..., K\}$ parameters: $\lambda_1, \lambda_2$ satisfying Eq. 5.9 |
| output | a pool of weak rules: $\Psi = \{F_i\}$ |

1 **Init**: $\Psi = \emptyset$;

2 **FIM**: Perform closed FIM on $T$ based on the support parameter $\lambda_1$, and obtain a set of compositional features $\Psi = \{F : frq(F) > \lfloor \lambda_1 \times N \rfloor\}$

3 **foreach** $F \in \Psi$ **do**

4 **if** $P(c(x) = k|F(x) = k) < \lambda_2 r_k, \forall k$ **then**

5 $\Psi \leftarrow \Psi \backslash F$

6 **Return** $\Psi$

5.3. Multi-class AdaBoost

After discovering $\Psi = \{F_i\}$, we need to boost them for a final classifier. Each mined rule $F_i \in \Psi$ is a weak classifier for a certain class $k \in \{1, 2, ..., K\}$. We follow the stagewise additive modeling with exponential loss (SAMME) formulation for multi-class AdaBoost in [86]. Given the training data $\{x_i, c_i\}$, our task is to find a regression function $g : x \rightarrow \mathbb{R}^K$, i.e., $g(x) = [g_1(x), ..., g_K(x)]^T$, to minimize the following objective function:
\[
\begin{align*}
\min_{g(x)} & \quad \sum_{i=1}^{N} L(y_i, g(x_i)) \\
\text{s.t.} & \quad g_1 + \ldots + g_K = 0,
\end{align*}
\]

where \( L(y, g) = \exp\left[-\frac{1}{K}(y_1 g_1 + \ldots + y_K g_K)\right] = \exp(-\frac{1}{K} y^T g) \) is the multi-class exponential loss function. \( y = (y_1, ..., y_K)^T \) is the \( K \)-dimensional vector associated with the output \( c \), where

\[
y_k = \begin{cases} 
1 & \text{if } c = k \\
-\frac{1}{K-1} & \text{if } c \neq k
\end{cases}.
\]

The symmetric constrain \( g_1 + \ldots + g_K = 0 \) is to guarantee the unique solution of \( g \), otherwise adding a constant to all \( g_k \) will give the same loss since \( \sum_{i=1}^{K} y_i = 0 \). It can be seen that the optimal solution of Eq. 5.12 is:

\[
g^*(x) = \arg\min_{g(x)} E_{y|x} \exp[-\frac{1}{K}(y_1 g_1 + \ldots + y_K g_K)] \\
\text{s.t.} \quad g_1 + \ldots + g_K = 0.
\]

It is notable that the solution of Eq. 5.14 is consistent with the Bayes classification rule in minimizing the misclassification error [86]:

\[
\arg\max_k g^*_k(x) = \arg\max_k \text{Prob}(c = k|x).
\]

Compared with AdaBoost.MH which needs to perform \( K \) one-against-all classifications, SAMME performs \( K \)-class classification directly. It only needs weak classifiers better than random guess (e.g. correct probability larger than \( 1/K \)), rather than better than \( 1/2 \) as two-class AdaBoost requires.
We modify original SAMME [86] in considering compositional features for boosting. By boosting compositional features, we actually consider a functional ANOVA decomposition [80] of \( g(x) \) by applying weak rules \( \mathcal{F} \) of any possible orders:

\[
g(x) = \sum_{m=1}^{M} \alpha^m \mathcal{F}^m(x) = \sum \mathcal{F}_i(x(i)) + \sum \mathcal{F}_{ij}(x(i), x(j)) + ...,\]

where \( \alpha^m \in \mathbb{R} \) are weight coefficients.

Our compositional boosting method is listed in Alg. 5. From Alg. 5, we can see one major difference between SAMME and two-class AdaBoost is in step 8, where an extra \( \log(K-1) \) is added to guarantee \( \alpha^m > 0 \) when \( \text{err}^m < \frac{K-1}{K} \). In the case of \( K = 2 \), it is equivalent to the original two-class AdaBoost because \( \log(K-1) = 0 \).
Algorithm 5: SAMME on Compositional Features

**input**: A pool of compositional rules $\Psi = \{\mathcal{F}_j\}$

Training dataset $D = \{x_i, c_i\}_{i=1}^N$

Iteration number, $M$

**output**: a strong classifiers: $g(\cdot) : x \rightarrow \{1, 2, ..., K\}$

1 **Init**: set the training sample weights $w_i = 1/N$, $i = 1, 2, ..., N$.

2 **for** $m = 1, 2, ..., M$ **do**

3 Select a classifier from the pool $\Psi$: $F^m(x) = \arg\min_{F_i \in \Psi} \sum_{i=1}^N w_i I(c_i \neq F^m(x_i))$.

4 $\Psi = \Psi \setminus \mathcal{F}$.

5 Compute weighted training error: $err^m = \frac{\sum_{i=1}^N w_i I(c_i \neq F^m(x_i))}{\sum_{i=1}^N w_i}$.

6 Compute: $\alpha^m = \log \frac{1-err^m}{err^m} + \log(K - 1)$.

7 Update sample weight: $w_i \leftarrow w_i \cdot \exp[\alpha^m I(c_i \neq F^m(x_i))]$.

8 Re-normalize $w_i$.

9 Return $g(x) = \arg\max_k \sum_{m=1}^M \alpha^m \cdot I(F^m(x) = k)$

Moreover, It is important to notice that each $\mathcal{F}$ is only a binary classifier for a specific class $k$. We transfer it into a $K$-class classifier by randomly guessing the rest $K - 1$ class label when $\mathcal{F}(x) = 0$. Specifically we estimate the weighted training error at step 4 as:

$$\sum_{i=1}^N w_i I(c_i \neq F^m(x))$$

$$= \sum_{c_i \neq k} w_i I(F^m(x) = k) + \frac{K - 1}{K} \sum_{i=1}^N w_i I(F^m(x) = 0),$$

where $I(\cdot)$ denotes the binary indicator function.
5.4. Experiments

5.4.1. UCI data sets

Figure 5.3. Comparison of real training error $\epsilon_F$ and its theoretical upper bound: (a) breast cancer data set, (b) wine data set, and (c) handwritten numeral data set. The closer the point to the 45 degree line, the tighter the upper bound.

To validate Theorem 2, we select 3 data sets from the UCI Machine Learning Repository for evaluation: (1) breast cancer Wisconsin (diagnostic) data set which consists of both malignant and benign samples, (2) wine data set which consists of 3 classes of data samples, and (3) multiple features data set which consists of handwritten numerals (‘0’–‘9’) extracted from a collection of Dutch utility maps. We apply different strategies to quantize the continuous features in 3 data sets. For the breast cancer and wine data sets, as there are only a few features (30 and 13 respectively), we select the mean value ($\theta_i = \mu_i = E[x(i)]$) at each individual feature $x_i$ for quantization and consider both positive and negative items in generating transactions (Eq. 5.1). For the handwritten numerals data set, each sample contains 649 features. To alleviate the computational cost of FIM, we apply $\theta_i = \mu_i + \sigma_i$.
for quantization and only consider the positive items \((x(i) \geq \theta_i)\) in generating transactions (Eq. 5.2), where \(\sigma_i\) is the standard variance of the \(i\)th feature \(x(i)\).

For each data set, we set \(\lambda_1 = \min_k r_k\) and \(\lambda_2 = \frac{1}{2\lambda_1}\), such that \(\lambda_1 \lambda_2 = 0.5 < 1\). The compositional feature discovery result is presented in Table 5.4.1. We discover in total 12597, 266 and 48452 qualified \((\lambda_1, \lambda_2)\)-compositional rules for the breast cancer, wine and handwritten numeral data set, respectively. The best class-specific rule discovered for these three data sets has training error 0.081, 0.022, and 0.017, respectively. All the discovered rules has smaller training error than the theoretical upper bound \(B_{\epsilon_F}\). To further compare the real training error \(\epsilon_F\) and the upper bound, we present all discovered rules in Fig. 5.3. Each point corresponds to a discovered compositional rule \(F\). The \(x\)-coordinate of a point gives its real classification error \(\epsilon_F\). The \(y\)-coordinate gives the upper bound training error calculated on its own support and confidence values according to Theorem 2). It is surprising to see that for most discovered rules in all data sets, the real classification error of a rule \(F\) is close to its own theoretical upper bound. We also notice the smaller the \(\epsilon_F\), the closer \(\epsilon_F\) to its own upper bound. These results show that the derived upper bound in Theorem 2 is quite tight for compositional rules.

Table 5.1. Compositional feature discovery in 3 UCI data sets.

<table>
<thead>
<tr>
<th></th>
<th>cancer</th>
<th>wine</th>
<th>numeral</th>
</tr>
</thead>
<tbody>
<tr>
<td># class</td>
<td>2</td>
<td>3</td>
<td>10</td>
</tr>
<tr>
<td># feature</td>
<td>30</td>
<td>13</td>
<td>649</td>
</tr>
<tr>
<td># sample</td>
<td>569</td>
<td>178</td>
<td>2000</td>
</tr>
<tr>
<td>(\theta_i)</td>
<td>(\mu_i)</td>
<td>(\mu_i)</td>
<td>(\mu_i + \sigma_i)</td>
</tr>
<tr>
<td>(\lambda_1)</td>
<td>0.37</td>
<td>0.27</td>
<td>0.10</td>
</tr>
<tr>
<td>(\lambda_2)</td>
<td>1.34</td>
<td>1.85</td>
<td>5.00</td>
</tr>
<tr>
<td># frq. itemsets</td>
<td>12,729</td>
<td>342</td>
<td>156,734</td>
</tr>
<tr>
<td># comp. feature</td>
<td>12,597</td>
<td>266</td>
<td>48,452</td>
</tr>
<tr>
<td>(B_{\epsilon_F})</td>
<td>0.561</td>
<td>0.406</td>
<td>0.150</td>
</tr>
<tr>
<td>max (\epsilon_F)</td>
<td>0.366</td>
<td>0.270</td>
<td>0.100</td>
</tr>
<tr>
<td>min (\epsilon_F)</td>
<td>0.081</td>
<td>0.022</td>
<td>0.017</td>
</tr>
<tr>
<td>aver. (\epsilon_F)</td>
<td>0.241</td>
<td>0.150</td>
<td>0.081</td>
</tr>
</tbody>
</table>
5.4.2. An event recognition problem

The real test is an event recognition problem. The goal is to recognize typical events from personal photo collections, where each event corresponds to a specific human activity taking place in a certain environment, and captured by a collection of images taken during the event: 

\[ E_i = \{I_j\}_{j=1}^{||E_i||}, \]

where \( I_j \) denotes an image. We chose 10 types of frequently occurring events with reasonably distinctive visual characteristics, inspired by the tag statistics revealed by Flickr.com: 

\[ C = \{ \text{Christmas activity, backyard activity, ball game, beach, birthday, city walk, hiking, road trip, skiing, wedding} \}. \]

They include both outdoor and indoor events. In general, event recognition is more challenging and complicated than scene recognition due to the higher semantics involved [88] - the visual content can vary dramatically from one instance to another (as shown later, boosting decision stumps did not perform well).

Table 5.2. A 10-class event recognition dataset. The first row describes event class name; the second and third row indicate the number of events and images in that class respectively.

<table>
<thead>
<tr>
<th>Chr</th>
<th>byd</th>
<th>bgm</th>
<th>bea</th>
<th>bir</th>
<th>cit</th>
<th>hik</th>
<th>rtp</th>
<th>ski</th>
<th>wed</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>8</td>
<td>7</td>
<td>8</td>
<td>10</td>
<td>9</td>
<td>10</td>
<td>9</td>
<td>7</td>
<td>10</td>
</tr>
<tr>
<td>234</td>
<td>127</td>
<td>229</td>
<td>197</td>
<td>212</td>
<td>182</td>
<td>634</td>
<td>922</td>
<td>149</td>
<td>567</td>
</tr>
</tbody>
</table>

For each event \( E \), it can be uniquely labeled with one of the 10 event classes: \( l(E_i) \in C \).

The experimental dataset contains 88 individual events, where each event contains a variable number of 7 to 108 images. There are 3453 images in total in the dataset. Due to the limited number of events, we perform leave-one-out test to report all results. Without extra mentioning, the support threshold is set as \( \lambda_1 = \frac{1}{11} \approx \min_k r_k \) and the confidence threshold is set as \( \lambda_2 = 4 \). Therefore for a specific class \( k \), its training error bound is: 

\[ \epsilon_k \leq \frac{1}{N^2} - \lambda_1 \lambda_2 r_k = \frac{1}{3} - \frac{4}{11} r_k \] (theorem 2). The quantization parameters \( \theta_i \) determine the transactions and thus have large influences on the mining and classification results. To carefully select \( \theta_i \), for each feature dimension \( x(i) \), we estimate its mean \( \mu_i = E[x(i)] \) and
variance $\sigma^2 = \text{Var}[x(i)]$. Then we set $\theta_i = \mu_i + \tau \times \sigma_i$, where $\tau > 0$ is a global parameter decided through leave-out-out cross validation.

**Visual vocabularies and feature primitives**

Visual vocabularies have proved to be an effective way of building visual recognition systems, e.g., for scene recognition [89]. An image is partitioned by a fixed grid and represented as an unordered set of image patches. Suitable descriptions are computed for such image patches and clustered into bins to form a “visual vocabulary”. In this study, we adopted the same methodology and extended it to consider both color and texture features for characterizing each image grid.

To extract color features, an image grid is further partitioned into $2 \times 2$ equal size sub-grids. Then for each sub-grid, we extract the mean $R$, $G$ and $B$ values to form a $4 \times 3 = 12$ feature vector which characterizes the color information of 4 sub-grids. To extract texture features, we apply a $2 \times 2$ array of histograms with 8 orientation bins in each sub-grid. Thus a $4 \times 8 = 32$-dimensional SIFT descriptor is applied to characterize the structure within each image grid, similar in spirit to [89] [90]. In our experiments, if an image is larger than 200k pixels, we first resize it to 200k. We then set image grid size of $16 \times 16$ with overlapping sampling interval $8 \times 8$. Typically, one image generates 117 such grids.

After extracting all the raw image features from image grids, we build separate color and texture vocabularies by clustering all the image grids in the training dataset through $k$-means clustering. In our experiments, we set both vocabularies of size 500. By accumulating all the grids in an event (a collection of images), we obtain two normalized histograms for an event, $h_c$ and $h_t$, corresponding to the word distribution of color and texture vocabularies, respectively. Concatenating $h_c$ and $h_t$, we end up with an normalized word histogram:
\[
\sum_{i=1}^{1000} h_i(E) = 1.
\]
Each bin in the histogram indicates the occurrence frequency of the corresponding word. We only consider the positive responses of \( f_i \) when the \( i \)th word appears frequently enough (i.e. \( h_i(E) > \theta_i \)) in the whole event. We have two types of visual vocabularies \( \Omega_c \) and \( \Omega_t \), where \( f_i \in \Omega_c \) is the color primitive, whereas \( f_i \in \Omega_t \) is the texture primitive. Denoting the complete vocabulary as \( \Omega = \Omega_c \cup \Omega_t \), we discover compositional rule \( F \subset \Omega \) which can contain integrated information of both color and texture.

**Event recognition results**

The leave-one-out test result is showed in Table 5.3, when \( \tau = 1.4 \) (\( \theta_i = \mu_i + 1.4\sigma_i \)). The iteration number of boosting is 400. The main confusion comes from the indoor social events, such as *birthday*, *Christmas* and *wedding*. We also notice confusion between *hiking* and *backyard activities*, possibly due to their visual similarity.

<table>
<thead>
<tr>
<th></th>
<th>Ch</th>
<th>by</th>
<th>bg</th>
<th>be</th>
<th>bi</th>
<th>ci</th>
<th>hi</th>
<th>rt</th>
<th>sk</th>
<th>we</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chr</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>byd</td>
<td>0</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bgm</td>
<td>1</td>
<td>1</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bea</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>bir</td>
<td>2</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>cit</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>hik</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>10</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>rtp</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>7</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>ski</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>wed</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>9</td>
</tr>
</tbody>
</table>

The composition of the mined feature pool \( \Psi \) is listed in table 5.4. Among the 400 weak classifiers finally selected through boosting, 42% of them are compositional features of high-orders and the rest are decision stumps.
Table 5.4. Order distribution of the mined weak classifier pool $\Psi$. The values are averaged by all the leave-one-out tests.

| $|\mathcal{F}|$ | 1    | 2    | 3    | 4    | $\geq 5$ | total |
|----------------|------|------|------|------|--------|-------|
| mined #        | 271.6| 120.1| 62.9 | 31.2 | 40.2   | 525.9 |
| used #         | 232.3| 86.1 | 37.6 | 16.8 | 27.1   | 400.0 |

We compare the results of boosting compositional features with conventional decision stumps in Fig. 5.4. Due to the high redundancy among compositional features because of sharing primitive features, we do not allow a same compositional feature to be selected again during boosting (see step 5 in Alg. 5). In comparison, we allow re-used stumps to follow conventional boosting. After feature mining, we obtain a pool of 526 compositional features. On the other hand, the pool of decision stumps contains 1000 features, which correspond to 2000 types of weak rules since we consider both positive and negative items of an individual feature. The quantization parameters are the same as those of compositional features. From Fig. 5.4, we notice that training error decreases faster in our method (iteration 150, error 1.3%) than in conventional decision stumps (iteration 400, error 6.7%). Better generalization ability is also obtained. Despite that more complex weak features are applied in our method, we do not observe over-fitting in the first 450 iterations, whereas boosting decision stumps surprisingly does. This validates the regularization of the discovered rules by their support.

In summary, while this event recognition problem is indeed challenging for conventional decision stumps (70.0%), the proposed mining and boosting approach is able to achieve significantly improved accuracy (80.7%).
Figure 5.4. Comparison of error between boosting decision stumps and boosting compositional features. The error is calculated through averaging all leave-one-out tests.

5.5. Conclusions

We present a data mining-driven approach to discovering compositional features for multi-class boosting, which is successfully applied to a challenging 10-class event recognition problem. Compared with boosting decision stumps, we achieve faster error decreasing in training as well as better generalization in testing. We show how the data mining parameters (*i.e.* support $\lambda_1$ and confidence $\lambda_2$) influence the descriptive and discriminative abilities of the mined features $\mathcal{F}$, and obtain the upper bound of the training error. This gives us the theoretical guidance in selecting appropriate parameters for data mining. Finally, by applying the SAMME algorithm in boosting, our method naturally handles the multi-class problem by combining the mined binary compositional rules.
CHAPTER 6

Conclusion

In this thesis, we systematically study the image and video data mining. I extended the work in mining structured data (e.g., transaction data) and semi-structured data (e.g., text) to mining image and video data that have much more complex structure. Several data mining algorithms are developed for discovering interesting and \textit{a priori} unknown patterns from image and video data. Compared with traditional data mining methods which mainly focus on discrete and structure data, such as texts and transactions, our methods are specifically designed for image and videos which are usually characterized by high-dimensional features and can exhibit large pattern variations.

In image pattern discovery, two fundamental challenges are addressed. Above all, unlike transaction and text data that are composed of discrete elements without ambiguity (i.e., predefined items and vocabularies), visual patterns generally exhibit large variations in their visual appearances. Secondly, visual patterns have more complex structures than transaction and text patterns. In our approach, both content and structure variations are handled in mining common visual patterns. For recurring pattern mining in video data, I proposed an efficient mining method that can also tolerate temporal pattern variations such as non-uniform scaling. The discovery of frequent visual patterns helps to interpret and model visual concepts. The results will lead to significant improvement in the quality of content-based and object-level multimedia retrieval, will greatly benefit visual recognition that requires large
datasets for training and evaluation, and will be very useful in intelligent video surveillance applications.

Finally, to justify the important of the discovered frequent patterns, I study how the discovered patterns can help to classification. The sufficient conditions that frequent patterns can be discriminative features is proposed, where the upper bounded empirical training error is provided. A corresponding data mining-driven approach has been proposed for searching optimal discriminative rules. By taking advantage of classic data mining methods, such as frequent itemset mining, the proposed method can discovered qualified compositional features efficiently. This result bridged the gap between research in frequent pattern mining in data mining and pattern classification in machine learning. We have demonstrated the advantages and potential of using them for visual event categorization in consumer photos.
CHAPTER 7

Appendix

7.1. Appendix:A

We prove theorem 1 here. Given two pixels $i$ and $j$, let both $\{x_i^k\}_{k=1}^K$ and $\{x_j^k\}_{k=1}^K$ be sequences of i.i.d. Bernoulli random variables indicating whether the corresponding pixel can receive a vote at the partition $k$. Assuming $x_i^k$ and $x_j^k$ are independent given $i$ and $j$, we have:

$$X_i^K - X_j^K = \sum_{k=1}^K x_i^k - \sum_{k=1}^K x_j^k = \sum_{k=1}^K (x_i^k - x_j^k),$$

(7.1)

where elements $\{(x_i^k - x_j^k)\}_{k=1}^K$ are also i.i.d. random variables. It is easy to see that $E(x_i^k - x_j^k) = p_i - p_j$. Thus according to the weak law of large numbers, we have $\forall \epsilon > 0$

$$\lim_{K \to \infty} \Pr(|X_i^K - X_j^K - p_i + p_j| < \epsilon) = 1.$$  

(7.2)

Now let $\epsilon = \frac{p_i - p_j}{2} > 0$, we have

$$\lim_{K \to \infty} \Pr\left(\frac{X_i^K - X_j^K}{K} > \frac{p_i - p_j}{2}\right) = 1.$$ 

(7.3)

Since $p_i - p_j > 0$, it follows

$$\lim_{K \to \infty} \Pr(X_i^K - X_j^K > 0) = 1.$$ 

(7.4)

7.2. Appendix:B

Pair-wise Dependency Test.

If $W_i, W_j \in \Omega$ are independent, then the process of randomly generating the pair $\{W_i, W_j\}$
in a transaction \( T_i \) is a \((0/1)\) Bernoulli trial with probability \( P(W_i, W_j) = P(W_i)P(W_j) \). According to the central limit theory, as the number of trials (transaction number \( N \)) is large, the Bernoulli distribution can be approximated by the Gaussian random variable \( x \), with mean \( \mu_x = P(W_i)P(W_j) \). At the same time, we can measure the average frequency of \{\( W_i, W_j \)\} by counting its real instance number in \( T \), such that \( P(W_i, W_j) = \hat{f}rq(W_i, W_j)/\hat{N} \).

In order to verify if the observation \( P(W_i, W_j) \) is drawn from the Gaussian distribution \( x \) with mean \( \mu_x \), the following T-score is calculated; \( S^2 \) is the estimation of variance from the observation data.

\[
t(\{W_i, W_j\}) = \frac{P(W_i, W_j) - \mu_x}{\sqrt{\frac{S^2}{\hat{N}}}} \tag{7.5}
\]

\[
= \frac{P(W_i, W_j) - P(W_i)P(W_j)}{\sqrt{\frac{P((W_i,W_j))(1-P(W_i,W_j))}{\hat{N}}}} \tag{7.6}
\]

\[
\approx \frac{\hat{f}rq(\{W_i, W_j\}) - \frac{1}{\hat{N}}\hat{f}rq(W_i)\hat{f}rq(W_j)}{\sqrt{\hat{f}rq(\{W_i, W_j\})}}. \tag{7.7}
\]

### 7.3. Appendix:C

We prove theorem 2 here. Given a \((\lambda_1, \lambda_2)\)-weak compositional rule \( F \in \Psi \), we have \( P(c(x) = k|F(x) = k) \geq \lambda_2 r_k \) and \( P(F(x) = k) \geq \lambda_1 \) according to Def. 7. Then we can upper bound the incorrect prediction of positive training samples:

\[
P(F(x) \neq k, c(x) = k)
\]

\[
= P(c(x) = k) - P(c(x) = k|F(x) = k)P(F(x) = k)
\]

\[
\leq r_k - \lambda_1 \lambda_2 r_k. \tag{7.8}
\]

Furthermore, the incorrect prediction of negative samples can be upper bounded as:
\[ P(\mathcal{F}(x) = k, c(x) \neq k) \]
\[ = [1 - P(c(x) = k | \mathcal{F}(x) = k)] P(\mathcal{F}(x) = k) \]
\[ \leq (1 - \lambda_2 r_k) \frac{1}{\lambda_2} \]
\[ = \frac{1}{\lambda_2} - r_k, \]  \hspace{1cm} (7.9)

where we apply \( P(c(x) = k | \mathcal{F}(x) = k) \geq \lambda_2 r_k \) and \( P(\mathcal{F}(x) = k) \leq \frac{1}{\lambda_2} \) (Eq. 5.8) to derive Eq. 7.9. Finally, the error probability bound is

\[ \epsilon_{\mathcal{F}} = P(\mathcal{F}(x) \neq k, c(x) = k) + P(\mathcal{F}(x) = k, c(x) \neq k) \]
\[ \leq \frac{1}{\lambda_2} - \lambda_1 \lambda_2 r_k. \]  \hspace{1cm} (7.11)

The above bound is tight, i.e., the equality of Eq. 7.11 holds, if both equalities hold in Eq. 7.8 and Eq. 7.9. It can be seen that if the equality holds for both conditions in Def. 7, i.e. \( P(\mathcal{F}(x) = k) = \lambda_1 \) and \( P(c(x) = k | \mathcal{F}(x) = k) = \lambda_2 P(c(x) = k) \), the equality of Eq. 7.8 holds. Moreover, if \( P(\mathcal{F}(x) = k) = \frac{1}{\lambda_2} \), the equality of Eq. 7.9 holds. In such a case, we have \( P(\mathcal{F}(x) = k) = \lambda_1 = \frac{1}{\lambda_2} \), which requires \( \lambda_1 \lambda_2 = 1 \).
References


