



HAL
open science

Automatic tag correction in videos : an approach based on frequent pattern mining

Hoang Tung Tran

► **To cite this version:**

Hoang Tung Tran. Automatic tag correction in videos : an approach based on frequent pattern mining. Data Structures and Algorithms [cs.DS]. Université Jean Monnet - Saint-Etienne, 2014. English. NNT : 2014STET4028 . tel-01623441

HAL Id: tel-01623441

<https://theses.hal.science/tel-01623441>

Submitted on 25 Oct 2017

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



THÈSE

pour obtenir le grade de

Docteur en Informatique

par

Hoang Tung TRAN

et intitulée

Correction automatique d'annotations de vidéos :
une approche à base de fouille de motifs fréquents

préparée au laboratoire Hubert Curien

la commission d'examen du 17 juillet 2014:

- Rapporteurs :* Bruno CRÉMILLEUX - Professeur, Université de Basse-Normandie, Caen
Céline ROBARDET - Maître de conférences (HDR), INSA, Lyon
- Examinatrice :* Christine SOLNON - Professeur, INSA, Lyon
- Directeur :* François JACQUENET - Professeur, Université Jean Monnet, St-Étienne
- Co-directeur :* Baptiste JEUDY - Maître de conférences, Université Jean Monnet, St-Étienne
- Co-directrice :* Elisa FROMONT - Maître de conférences, Université Jean Monnet, St-Étienne



PHD THESIS

to obtain the title of

PhD of Computer Science

Defended by

Hoang Tung TRAN

Entitled

Automatic tag correction in videos:
an approach based on frequent pattern mining

prepared at Hubert Curien Laboratory

Jury 2014/07/17:

<i>Reviewers :</i>	Bruno CRÉMILLEUX	-	Professor, Basse-Normandie University, Caen
	Céline ROBARDET	-	Assistant professor (HDR), INSA, Lyon
<i>Examiner :</i>	Christine SOLNON	-	Professor, INSA, Lyon
<i>Supervisor :</i>	François JACQUENET	-	Professor, Jean Monnet university, St-Étienne
<i>Co-supervisor :</i>	Baptiste JEUDY	-	Assistant professor, Jean Monnet university, St-Étienne
<i>Co-supervisor :</i>	Elisa FROMONT	-	Assistant professor, Jean Monnet university, St-Étienne

Acknowledgements

First and foremost, I would like to thank to my PhD advisors, Professors Elisa Fromont, Baptisete Jeudy and Francois Jacquenet. It has been an honor to be one of Elisa's first Ph.D. students. She has taught me, both consciously and unconsciously, how a good researcher should be. I appreciate all her contributions of time and ideas to make my Ph.D. experience productive and stimulating. I hope that I could be as lively, enthusiastic, and energetic as her and to someday be able to do research as well as she can. Baptiste has always been patient and understanding with my mathematical struggles. Francois has been supportive and has given me the freedom to pursue various directions without objection. He has also provided many insightful discussions and suggestions as well as scientific advices. This work would not have been possible without their guidance, support and encouragement.

I gratefully acknowledge the members of my Ph.D. committee, Prof. Bruno Cremilleux, Celine Robardet and Christine Solnon, for their time and valuable feedback on a preliminary version of this thesis. I am very thankful to Prof. Marc Sebban and Amaury Habrard for their valuable suggestions concerning statistical theory.

I would like to thank my friends in the computer science department at Hubert Curien Laboratory for all the great times that we have shared. Specifically, I would like to acknowledge Jean-Philippe Peyrache for proposing so many sport activities; Chahrazed Bouhini for being the only girl in our office; Aurélien Bellet for lots of valuable advice; being-in-love guy (Rahul Mourya) for showing your warm feeling everyday; funny guy (Mohammad Motasem Nawaf) for his intentional and unintentional jokes; family guy (M.A. Hasnat) for always reminding me about family value; David Combe for always standing by my side. I would also like to extend my thanks to the technicians of the laboratory of Hubert Curine Laboratory for their help in offering me the resources in running the program.

I take this opportunity to sincerely acknowledge the Vietnam International Education Development (VIED), Government of Vietnam, Hanoi, for financial support during last 3 years. I would like to acknowledge the Caisse Allocations Familiales (CAF), French Government, Saint Etienne for providing financial assistance which buttressed me to have a comfortable life in France.

I am deeply thankful to my family for their love, support, and sacrifices. Without them, this thesis would never have been written. This last word of acknowledgment I have saved for my dear wife Hong Thinh and my beloved Bao Truc, who have been with me all these years and have made them the best years of my life.

TRAN Hoang Tung

Saint Etienne, France

July 2014

Abstract

This thesis presents a new system for video auto tagging which aims at correcting the tags provided by users for videos uploaded on the Internet. Most existing auto-tagging systems rely mainly on the textual information and learn a great number of classifiers (one per possible tag) to tag new videos. However, the existing user-provided video annotations are often incorrect and incomplete. Indeed, users uploading videos might often want to rapidly increase their video's number-of-view by tagging them with popular tags which are irrelevant to the video. They can also forget an obvious tag which might greatly help an indexing process. In this thesis, we limit the use of this questionable textual information and do not build a supervised model to perform the tag propagation. We propose to compare directly the visual content of the videos described by different sets of features such as SIFT-based Bag-Of-visual-Words or frequent patterns built from them. We then propose an original tag correction strategy based on the frequency of the tags in the visual neighborhood of the videos. We have also introduced a number of strategies and datasets to evaluate our system. The experiments show that our method can effectively improve the existing tags and that frequent patterns built from Bag-Of-visual-Words are useful to construct accurate visual features.

Résumé

Nous présentons dans cette thèse un système de correction automatique d'annotations (tags) fournies par des utilisateurs qui téléversent des vidéos sur des sites de partage de documents multimédia sur Internet. La plupart des systèmes d'annotation automatique existants se servent principalement de l'information textuelle fournie en plus de la vidéo par les utilisateurs et apprennent un grand nombre de "classifieurs" (potentiellement un par tag) pour étiqueter une nouvelle vidéo. Cependant, les annotations fournies par les utilisateurs sont souvent incomplètes et incorrectes. En effet, un utilisateur peut vouloir augmenter artificiellement le nombre de "vues" d'une vidéo en rajoutant des tags non pertinents ou peut simplement oublier une annotation évidente qui aurait pu être utilisée efficacement par un système d'indexation. Dans cette thèse, nous limitons l'utilisation de cette information textuelle contestable et nous n'apprenons pas de modèle pour propager des annotations entre vidéos. Nous proposons de comparer directement le contenu visuel des vidéos par différents ensembles d'attributs comme les sacs de mots visuels basés sur des descripteurs SIFT ou des motifs fréquents construits à partir de ces sacs. Nous proposons ensuite une stratégie originale de correction des annotations basées sur la fréquence des annotations des vidéos visuellement proches de la vidéo que nous cherchons à corriger. Nous avons également proposé des stratégies d'évaluation et des jeux de données pour évaluer notre approche. Nos expériences montrent que notre système peut effectivement améliorer la qualité des annotations fournies et que les motifs fréquents construits à partir des sacs de motifs fréquents sont des attributs visuels pertinents.

Contents

Contents	v
List of Figures	viii
Introduction	1
1 State-of-the-Art	5
1.1 Transformation Step	6
1.1.1 Video Features	6
1.1.1.1 Feature Classification	7
1.1.1.2 Feature Extraction	9
1.1.1.3 SIFT Descriptor	10
1.1.1.4 Bag-of-Word Construction	12
1.1.2 Structure Analysis	13
1.1.2.1 Shot Boundary Detection	14
1.1.2.2 Keyframe Extraction	15
1.2 Pattern Mining	16
1.2.1 Overview	16
1.2.1.1 Generalities over Pattern Mining Datasets	17
1.2.1.2 Frequency and Apriori Principle	18
1.2.1.3 Itemset Mining Algorithms	20
1.2.1.4 Sequence Mining Algorithms	21
1.2.2 KRIMP and SLIM Heuristic Algorithms	22
1.2.2.1 Minimum Description Length (MDL)	22
1.2.2.2 KRIMP Algorithm	26
1.2.2.3 SLIM Algorithm	28
1.2.3 Converting Image Descriptors into Binary Vectors	30

1.2.4	Pattern Mining applied to Computer Vision	32
1.3	Current Tagging Methods	33
1.3.1	Comparing Videos	33
1.3.1.1	Averaging	34
1.3.1.2	Pairwise	34
1.3.2	Automatic Tagging	34
1.3.2.1	Automatic Tagging Benchmarks for Videos	35
1.3.2.2	Model-based methods	36
1.3.2.3	Nearest neighbors-based methods	37
1.3.3	Tag Processing	37
2	Our Tagging Framework	39
2.1	Parameters Evaluation	40
2.1.1	Image Dataset	40
2.2	Step 2: Feature Construction	41
2.2.1	Encoding Images Using Frequent Patterns	42
2.2.2	Classification Using Supervised Mined Patterns (FLH Patterns)	43
2.2.3	Unsupervised Mined Patterns of <i>Oxford-Flower17</i>	44
2.2.3.1	Global Histogram	45
2.2.3.2	Local Histograms	45
2.2.4	Pattern Reduction Techniques	46
2.2.4.1	PCA (for supervised case)	46
2.2.4.2	MDL-based Reduction (for unsupervised cases)	48
2.2.5	Distance Functions	50
2.3	Step 3: Video Comparison	51
2.4	Step 4: Tag Propagation	52
2.5	Conclusion	54
3	Datasets and Tag Propagation Experiment Designs	55
3.1	A First Real Dataset (51 videos)	56
3.1.1	Dataset Description	56
3.1.2	Encoding Procedure	56
3.1.3	Evaluation Procedure	59
3.1.4	Results	61
3.2	Synthetic 182 Videos Dataset	61
3.2.1	Dataset Design	62

3.2.2	Evaluation Procedure	63
3.2.3	Results	64
3.3	A second real dataset (668 videos)	65
3.3.1	Building Dataset	65
3.3.2	Evaluation Procedure	65
3.3.3	Experiment Settings	67
3.3.3.1	Finding Patterns Without Tag Information	69
3.3.3.2	Finding Patterns With Tag Information	71
3.3.4	Results	72
3.4	Conclusion	73
4	Conclusions and Future Works	75
4.1	Contributions & Conclusions	75
4.2	Future Works	77
	References	79

List of Figures

1.1	Knowledge Discovery in Databases (KDD) Process	5
1.2	An example of color histogram in RGB space. (source: [34])	7
1.3	Interest points (left) and dense sampling points (right).	10
1.4	Gradient magnitude and orientation of a patch.	11
1.5	8-bin orientation.	11
1.6	128-D vector.	11
1.7	Building bag of visual word.	12
1.8	Building local BOW features.	13
1.9	An example of apriori principle based pruning. (source: [69])	19
1.10	Breadth-first search vs Depth-first search approaches. (source: [69])	20
1.11	Code table example (from [96]). The widths of the codes indicate lengths. The <i>usage</i> column is here only to demonstrate the more a code is used, the shorter it is.	23
1.12	A database example with its cover and encoded length derived from the code table shown in Figure 1.11. (source: [96])	24
1.13	A example of Standard Code Table with its cover and encoded length derived from the database shown in Figure 1.12. (source: [96])	25
1.14	Examples of different binarization techniques. From top to bottom: SBin, multiple uniform intervals, non-uniform intervals and B2S.	31
1.15	A video is represented as an average of all its keyframes.	34
1.16	The distance between two videos equals to the distance between two most similar keyframes (the red line in this example).	35
2.1	The Overall Framework	39
2.2	Class examples of <i>Oxford-Flower17</i>	41
2.3	Image representation from global histogram (global BoW) using frequent patterns	42

2.4	Image representation from local histograms (local BoW) using frequent patterns	43
2.5	Explained Variance according to the number of Principal Components .	47
2.6	Accuracy given by the 1-NN algorithm according to the number of Principal Components	48
2.7	A dense line is a distance from the one keyframe on the left to the most similar keyframe of the right video. The distance from the left video to the right video is the average of three dense lines.	52
2.8	An example of tag propagation: tag <i>Harry Potter</i> should be removed because it does not appear in the neighborhood, and tag <i>football</i> should be propagated because it appears in 5 over 6 neighbors.	53
3.1	Several example keyframes of the 51-videos dataset.	58
3.2	Result of our tag correction algorithm on the small real video dataset. Our algorithm uses only bag of word features (solid line) or histograms of both words and frequent patterns (dotted line).	61
3.3	Several examples of keyframes of the synthetic 182 video dataset along with the tags associated with the video the keyframe belongs to.	62
3.4	Percentage of wrong tags in the videos after one propagation step according to the percentage of noise introduced. The blue and green lines give the results of our method which compare histograms of SIFT-BOW using a L1 distance function. The red line gives the result of the propagation step starting from an ideal visual comparison.	63
3.5	Percentage of wrong tags in the videos after one propagation step according to the percentage of noise introduced computed on a synthetic video dataset. Our algorithm uses only SIFT-BOW features, frequent patterns or both.	64
3.6	Distribution of the 150 kept tags over the 668 video dataset (top), and, for each video in the dataset, over all its 30 nearest neighbors videos (bottom). The nearest neighbors are computed using the SIFT-BOW baseline and the L1 distance function.	67
3.7	Several examples of keyframes of the real 668 videos dataset along with parts of the original tags and the complete pre-processed tags associated with the video the keyframe belongs to. The result of the tag propagation using our system and the tag ground truth is also given for these keyframes.	68

3.8	Number of nearest neighbor videos that contain (plain line) or should contain the tag “amanda” to trigger the propagation step (dashed line) according to the number of nearest neighbors for 4 different videos. A video is described using an average of the histograms representing all its keyframes. The keyframes are described using one global SIFT bag-of-words. The videos are compared using a L1 distance function.	69
3.9	Four different strategies to obtain a discriminant set of frequent patterns. The two first ones use first the LCM algorithm on the dataset and compute the maximal or the closed patterns. Then the patterns are post-processed to keep the emerging ones (using the tag information) and further post-processed using the SLIM algorithm. In the third and fourth strategies, the tag information is not used.	70
3.10	Percentage of good corrections according to the number of neighbors after one propagation step using a SIFT-BOW video encoding (baseline) and an encoding based on SLIM patterns derived directly from SIFT-BOW. The dataset is the 50 ground-truth sample taken from the 668 real video preprocessed dataset.	71
3.11	Percentage of good corrections according to the number of neighbors in 50 videos represented with SIFT-BOW (baseline) and frequent patterns obtained LCM(closed and max)+SLIM and LCM(closed and max)+SLIM+ tag_information.	72
3.12	Percentage of good corrections according to the number of neighbors for a video dataset represented with 3 different features: SIFT-BOW (baseline) and frequent patterns obtained with LCM(closed) + SLIM + tag_info, and the concatenation of both feature vectors.	73
3.13	Percentage of good corrections when counting only the addition (dashed line with +), the deletion (dashed line with o) and both according to the number of neighbors for a SIFT-BOW (baseline), LCM(closed) + SLIM + tag_info and the concatenation of both feature vectors.	74
3.14	Percentage of good corrections to the number of neighbors for a video dataset represented by frequent patterns obtained with LCM(closed) + SLIM + tag_information using the asymmetrical similarity measure and simply averaging the keyframes features and computing an histogram intersection.	74

Introduction

It's now commonplace to say that the digital universe is rapidly growing. In fact, according to a market survey performed by IDC¹, between 2009 and 2020 the amount of digital information will grow by a factor of 44. In the case of digital multimedia content, that trend is confirmed by the fact that video-sharing websites are growing very quickly. If we consider for example the particular case of YouTube, the statistics provided by that company² say that "over 6 billion hours of video are watched each month on YouTube — that's almost an hour for every person on Earth, and 100 hours of video are uploaded to YouTube every minute". Concerning the users of such services, YouTube says more than 1 billion unique users visit their servers each month, that shows there is a huge demand for such resources.

How can users retrieve interesting videos on the Web among this huge amount of data? This is a more and more difficult issue to handle. Multimedia content indexing has been a very active field of research in recent years as the benefits of such a research will have a strong impact on the usage which may be done by ordinary people of the internet tomorrow. The famous survey on visual content-based video indexing and retrieval from Weiming Hu [46] reflects the intensive research in this field. The difficulty of the task may be easily understood. Working on videos means dealing with huge quantities of keyframes (generally at least 24 per second) made up of hundreds of thousands pixels. Even if image indexing techniques begin to be efficient, there is still work to do to get such results with videos. In fact, textual document indexing is a far more mature domain and efficient algorithms have been provided for years in the field of information retrieval that were dedicated primarily to textual document retrieval [76]. The most efficient tool available for ordinary people is without no doubt Google that allows to instantly find relevant documents from a huge set of documents stored

¹J. Gantz and D. Reinsel, "The Digital Universe Decade, Are You Ready?," 2011. http://www.emc.com/digital_universe

²<http://www.youtube.com/yt/press/statistics.html>

on Google’s servers. Even if many efforts are still needed to improve search engines such as Google to better search for textual documents, textual document indexing is far more advanced than video indexing at the moment.

Thus, the approach based on combining textual information and video resources has begun to provide promising results [77] and is widely used in most commercial video-sharing websites such as YouTube, DailyMotion, Wat TV, etc. The users of such websites generally have to manually annotate the videos while uploading them providing a small textual description or a set of keywords that are generally called tags. Those tags are then used to index the videos and make them available for the other users.

However, to allow the search engines underlying these websites to accurately index the extensive resource of online videos, each video has to be carefully annotated. Unfortunately these annotations are often incorrect. Indeed, users uploading videos might often want to rapidly increase their video’s number-of-view by tagging them with popular tags such as “Harry Potter”, even if those videos have nothing to do with this famous book series. Moreover the set of tags attached to a video is also generally incomplete that is, it is not sufficient to correctly characterize the video. The poor quality of the tags leads to a poor quality of video indexing and many videos are hidden to people just because their tags are not accurate.

To solve this problem, video-sharing websites might consider recruiting people whose job would be to annotate videos all day long. Given the amount of data, it would be very expensive, but this is not even the main issue. In fact, in the market survey performed by IDC presented previously, it was not only given facts about the growth of the digital information between 2009 and 2020, but also that, in the same time, the staffing and investment to manage it will grow by a factor of only 1.4. Given the growth of digital storage, there would never be enough human resources to annotate the uploaded videos and the resulting cost.

In fact, a major issue to make professional people annotate videos is that the annotation task is in essence subjective. This may result in a tag quality that may be dependent of the tagger, of his personal knowledge about the videos he has to tag, of the moment of the day he performs his task, etc. To solve this problem we may think about an automatic process to help people tagging videos. The first step would consist in manually tagging a not so huge quantity of videos, and the second step would be done by the machine that could learn (with supervised machine learning techniques) a classifier for each tag that has been used. Nevertheless this approach is not realistic

because the number of tags used in websites like youtube may reach millions. Training millions of classifiers is not a solution.

In this thesis, in order to address these problems, we focus on improving the annotation of videos provided by users of video-sharing websites. There have been many efforts to design algorithms to automatically annotate videos (see for example [77] and [62]). However, most of the current approaches are based on supervised learning techniques which is not acceptable for scalability reasons.

Thus, we want to propose an unsupervised data mining approach based on the comparison of the visual content of the videos to propagate the tags of the most similar videos. There are two motivations for applying data mining techniques to design an unsupervised video tagging system. The first one is that despite the drawbacks of the user-provided annotations we previously talked about, these millions of tags result from the annotation of millions of videos by millions of users all around the world and it would be a pity not to use such a great source of information. Indeed, adequately processing the (eventually noisy) tags will certainly help a lot the video indexing task. The idea we want to implement is that if people upload videos about a topic T , most people will tag that video with a tag "T". Then, if a new video about T is uploaded without that tag, and if we may find many similar videos with tag "T", that new video should automatically be tagged with "T".

The second motivation for applying data mining techniques in this thesis is that many success stories have been told about data mining techniques in the area of business and science. Many papers have also been published about using data mining techniques in the context of image/video processing in order to extract discriminative patterns. Applying data mining techniques in our context is just a natural step since we have large amount of data and we want to discover interesting information in it, that is knowledge about the way particular videos are associated to particular tags.

Of course, the problem we want to tackle is not simple because it confronts us with a number of challenges. The first one is related to the classical problem of computational cost while mining huge datasets. That cost becomes even higher with video datasets. Indeed, as any video segment can be a candidate for a "good pattern", performing exhaustive search on all possible candidates at various locations (space) and lengths (time) will involve extremely huge computational cost. Of course, a second challenge is related to the fact that (pre-)processing videos is far more difficult than (pre-)processing

transactional data stored into a relational database. Another important challenge is the classical problem of overwhelming amount of patterns resulting from the data mining step. As we said previously, other researches about automatic tagging of videos often rely on supervised approaches thus they can use supervised information (the class of the video) to post-process the extracted patterns and only keep those one that are discriminative for a given class. However, in our case, there is no such information, thus post-processing becomes a very important issue. Finally, another challenge is that we have some video datasets without any proper ground truth. This makes our approach difficult to compare with other state-of-the-art approaches.

Several contributions are presented in this thesis. First we look at the various features that could be used to design an efficient comparison function between videos. Besides, we evaluate our approach and compare it with state-of-the-art techniques. Then we propose a simple but effective tag propagation process and finally evaluate the entire system.

The remaining of the thesis is organized as follows. Chapter 1 describes the state of the art techniques that are useful for our research. It first describes the basic concepts in the image processing field and more specifically goes into more details about a classical visual feature called SIFT-BoW. It then presents some popular data mining techniques as well as how to apply them in the context of videos. The last section is about the tagging process and presents some current techniques. Chapter 2 presents an overview of our framework that aims to automatically tag videos. In that context, there are many parameters that need to be tuned, thus, extensive experiments with image datasets are performed to evaluate the impact of each one. Chapter 3 presents some experiments on various datasets in order to demonstrate the effectiveness of our approach. The two first datasets are quite simple with a limited number of videos or tags, the third one has nearly 700 videos with 150 tags. Finally we will conclude and propose some future developments.

Chapter 1

State-of-the-Art

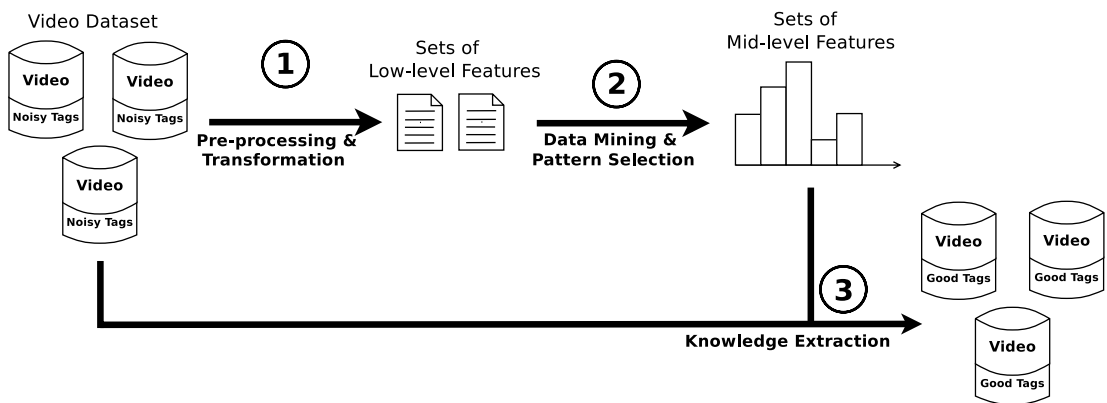


Figure 1.1: Knowledge Discovery in Databases (KDD) Process

As explain in the introduction, our aim is to automatically correct and complete tags on large video datasets such as, for example, the Youtube dataset. To tackle this problem, we will follow a process very similar to the traditional knowledge discovery from databases (KDD) process as shown in Figure 1.1. In the first step, the video dataset undergoes a series of pre-processing substeps: *selection* and *cleaning*. For example the cleaning step could consist in filtering out the videos with a resolution too low to be processed, in cropping all videos to get a dataset of videos of the same size, in dividing the dataset into batches of manageable size, etc. These steps are not developed in this chapter and we will assume that the dataset we are using can undergo the subsequent processes (more details about our specific pre-processing steps are given in Chapter 4). Once done these first substeps, one has to convert the video dataset into a suitable format for the data mining process, this is the *transformation step*. In our

case, the *data mining step* is then used to produce a set of Mid-level features that are, hopefully, more suitable than the low-level features that can be directly extracted by image processing algorithms. Then, a *knowledge extraction step* is performed. In our case, it is a tag propagation step based on the comparison of the videos described by the newly found mid-level features. This chapter focuses on the these three last steps.

1.1 Transformation Step

We call *transformation step* the process of extracting sets of features from a clean video dataset. To do that, one first needs to analyze the structure of the video and then, extract some features to describe it. The structure analysis consists in splitting the video into multiple *frames*. On the other hand, the feature extraction process aims at representing a frame (i.e an image) as a vector of visual features. However, since the structure analysis is also performed based on some features extracted from the frames, we start this section with a review of the different common features used in image and video analysis and we will then explain how the structure analysis is performed.

1.1.1 Video Features

This section will cover some visual features that are often used for video analysis. Auditory features and text features are not considered here although they are combined with other features in some articles related to automatic tagging of videos [86]. Different features are usually best suited for different tasks so the current trend is to concatenate all kinds of low level features and obtain a very high dimensional vector that will be used for video comparison. Another very popular technique is to construct a Bag-of-visual words [102] (BOW) from the original low-level feature vectors. In this document, we will mainly rely on local features called SIFT (for Scale Invariant Feature Transform) [55] and SIFT-Bag-of-words. The name SIFT applies both to an interest point detector method and a local feature used in image processing. It is robust and can be used to identify the same features in different images, even with large changes in scale and rotation. In this section, we will first classify the different group of video features then briefly explain how they can be extracted. We will focus on the particular case of the SIFT descriptors and SIFT bag-of-words since they are the ones used in the remainder of this document.

1.1.1.1 Feature Classification

Since a video is composed of frames (i.e. images), most of the traditional image features can also be used for video analysis. Features can be roughly classified as color-based, texture-based and shaped-based. For the specific case of videos, we add the motion features.

Color-Based Features. They include color histograms [6], color moment [100], color correlograms, etc. These features depend on color spaces such as RGB (Red-Green-Blue), HSV (Hue-Saturation-Value, which is a representation of points in an RGB color model), YCbCr (this is another way of encoding RGB information: Y is luminance, Cb and Cr represent the chrominance i.e. differences between the luminance and the blue (resp. red) color channels), etc. The choice of color spaces depends on applications. Color features can be extracted from the whole frame or from an image block. They are very simple but still efficient. Figure 1.2 shows an example of a color histogram. In the specific context of automatic tagging, color moment (together with other features) is used in [63], background features (8x8 Hue-Saturation) are used in [86] and in [62], frame features include LAB and HSV global color histograms.

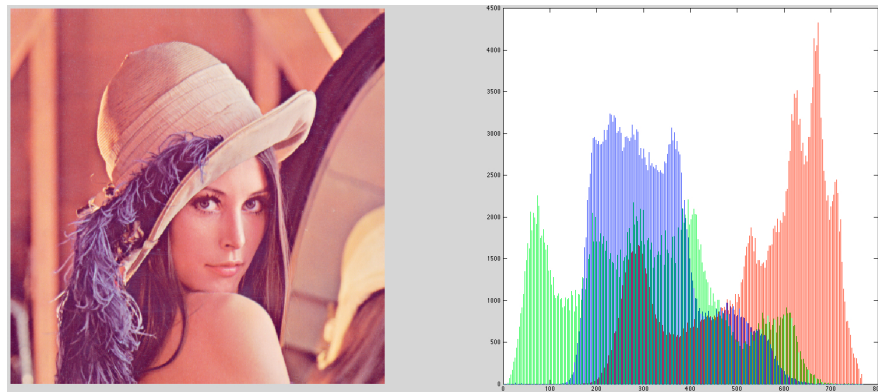


Figure 1.2: An example of color histogram in RGB space. (source: [34])

Texture-Based Features. Similar to color, texture is an important part of human visual perception and is another essential feature. Unlike color, texture spreads over a region instead of at a point. It is normally defined solely by grey levels and thus is orthogonal to color. Texture features contain important information about object surfaces as well as their correlations with the surrounding area. The most common

texture features include Tamura feature [84], co-occurrence matrix [40], and wavelet transformation-based features [88]. For example, Amir *et al.* [6] use Tamura and co-occurrence features for a video retrieval task in TRECVID-2003, Hauptmann *et al.* [44] use Gabor wavelet filters for a video search engine. In the specific context of tagging, Haar and Gabor wavelets are, for example used in [62] and in [86].

Shape-Based Features. Shape-based features describe object shapes, and they can be extracted from object contours. A common approach is to detect edges in keyframes, and then use a histogram to describe the edges distribution. For example, Hauptmann *et al.* [44] use edge histogram descriptor in TRECVID-2005 for the video search task. At local level, Foley *et al.* [31] divide an image into 4 x 4 blocks, and then extract a edge histogram for each block. Shape-based features are much more difficult to extract than color and texture-based features. Yang *et al.* [103] provide a survey about shape-based features for interested readers. SIFT (for Scale Invariant Feature Transform)[55] is a very popular descriptor considered as a shape-based feature. This descriptor, firstly proposed by Lowe [55], is scale rotation, illumination and viewpoint invariant. It is robust and can be used to identify the same features in different images, even with large changes in scale and rotation. It is presented in more details in the following subsection. In the specific context of tag propagation, edge distribution histograms are used in [63] whereas Yang and Toderici [104] use HOG (Histograms of Oriented Gradient)[20].

Motion Features. The previous features do not take into account the spatial-temporal relationship between video frames, which is the crucial property that distinguish videos from still images. Motion features fill this gap by presenting the temporal information. In general, there are two types of motions: background motion caused by the camera movement, and foreground motion caused by moving objects. Thus motion features can be divided into 2 categories: camera-based and object-based. Within camera-based features, there are different camera motions such as “zooming in and out”, “tilting up and down” etc. These features can not capture the motion of the important objects.

Object-based motion features are more interesting and thus more studied. A first approach consist in trying to model the distribution of global or local motions in videos. For instance, Ma and Zhang [57] convert the classic motion vectors into a number of directional slices, these slices in turn are used to create a multidimensional vector which is called motion texture. This approach has low computational complexity, but it lacks an accuracy representation for object actions. The second approach is modeling the

motion trajectories of objects in videos [21]. In [9], each trajectory is represented as temporal orderings of sub trajectories which are represented by their principle component analysis (PCA) coefficients. Su *et al.* [81] construct motion flow from motion vectors to create continual motion information in the form of a trajectory. With a given query as a trajectory, the system retrieves trajectories that are similar to it. Even this method can describe object actions, it depends on many difficult factors: correct object tracking, automatic recording of trajectories... The third approach describes spatial relationship between objects. Nakanishi *et al.* [64] construct the spatio-temporal relationships between objects by putting objects' traces across a time line. The limitation of this approach is the difficulty of labeling each objects and its position.

Features Fusion Fusion is a step of combining different objects, for example, concatenating visual features is called *early fusion*. Combing multiple features is not simply put one feature next to another, for example, if the former ranges $[0, 1]$ while the latter ranges within $[0, 1000]$, a normalization term is required before concatenating them.

1.1.1.2 Feature Extraction

Local features are usually extracted by Interest Point detectors. An interest point is a point in the image which, in general, is rich in terms of local information contents. Corner detectors (for example, [61] or [41]) and blob detectors are the two most common types of interest point detectors. In 2004, Lowe [55] proposed the SIFT interest point detector, to identify potential interest points by selecting the local extrema of an image filtered with differences of Gaussian. The output of a sift detector is a point, a scale which generates a patch around the point (depending on the scale of the image) and an orientation which corresponds to the dominant gradient in the patch. The orientation will provide rotation-invariance properties to the descriptors.

The key-points extracted from these detectors are then described (with some descriptors) and used as input for any image/video analysis tools. However, extracting a large number of interest points can be expensive and, not necessarily reliable if the images/videos are especially diverse. In 2006, [66] showed that for description purposes, one could use a simpler strategy that obtain comparable performance as detecting a set of interest points. This strategy is called *dense sampling*. It consists in uniformly selecting points in the image, for example on a regular grid, instead of focusing on particularly interesting ones. An example of these two methods are shown in figure 1.3. In the special case were the targeted descriptors are SIFT features, [89] proposed

a very efficient dense sampling implementation written to avoid the computations of overlapping regions.



Figure 1.3: Interest points (left) and dense sampling points (right).

1.1.1.3 SIFT Descriptor

We will now focus on the particular case of the SIFT descriptors as they are the main features used in this thesis. A SIFT descriptor is a 128-Dimensional vector which describes weighted gradient information around a given point. To construct such a descriptor, one has to identify a neighborhood region (a patch) around a point given by a scale parameter and an orientation for this patch. These parameters are provided by a sift detector and should be chosen arbitrarily when using a dense sampling method [43].

To compute the SIFT descriptor, the patch is encapsulated inside a 16x16 window (as in Figure 1.6) broken into sixteen 4x4 windows. Within each small window, gradient magnitudes and orientations are calculated. The gradient magnitude and orientation of a point (x, y) (see Figure 1.4) can be computed by:

$$m(x, y) = \sqrt{(L(x + 1, y) - L(x - 1, y))^2 + (L(x, y + 1) - L(x, y - 1))^2}$$

$$\theta(x, y) = \tan^{-1} \frac{L(x, y + 1) - L(x, y - 1)}{L(x + 1, y) - L(x - 1, y)}$$

These gradients are computed for all points (x, y) within the window. Then, a histogram is created for the orientations. By default, the 360 degrees are divided into 36 bins (each bin has 10 degrees): the first bin is from 0 to 9 degrees, the second bin

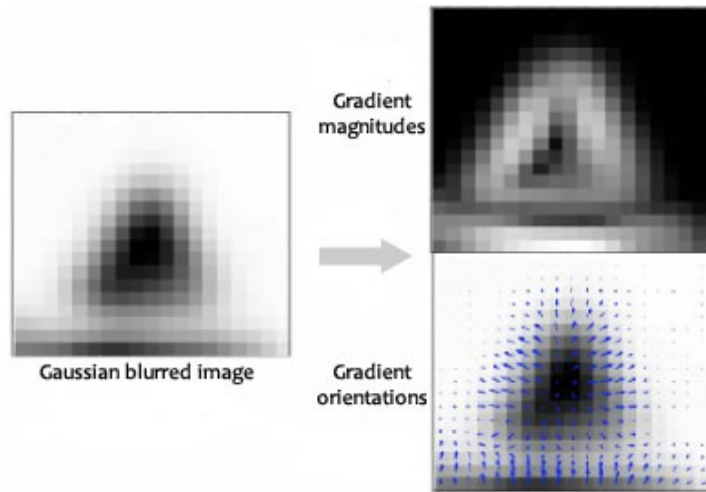


Figure 1.4: Gradient magnitude and orientation of a patch.

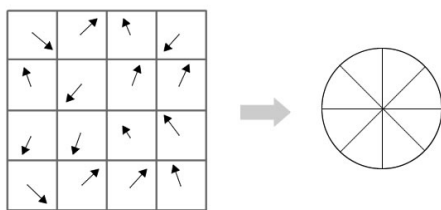


Figure 1.5: 8-bin orientation.

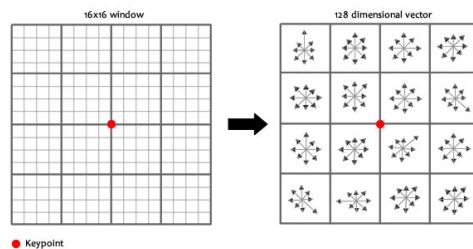


Figure 1.6: 128-D vector.

is from 10 to 19 degrees, and so on. A weight is associated to each bin. This weight is proportional to the magnitude of gradient at that point, and to a Gaussian window or a Gaussian weighting function which encodes the distance towards the original interest point. Then, these orientations are again split into an 8-bin histogram (instead of the original 36 bins): the first bin is from 0 to 44 degrees, the second bin is from 45 to 89 degrees, and so on (see Figure 1.5). The weights computed before guarantee that gradients far away from the interest point will add smaller values to the histogram.

As a result, one 4x4 window is equivalent to a 8-bin histogram or a 8-dimensional vector. The whole 16x16 window can now be presented as a $(4 \times 4 \times 8 =)$ 128 dimensional vector. To make this feature rotation independent, the interest point's prominent rotation (the original orientation) is subtracted from each and every orientation (i.e. all 128 values). A normalization step is done by dividing each bin by the root of the sum of the squared vector.

1.1.1.4 Bag-of-Word Construction

The bag-of-words model is a representation originally used in natural language processing [42]. In this model, a text (such as a sentence or a document) is represented as an unordered collection of words, disregarding grammar and word order. Recently, the bag-of-words model has also been adopted for computer vision. Thus some prefer calling this Bag-of-Visual-Word (BOW) instead, to emphasize that this technique is used in image processing, and the “word” is a “visual word” instead of a “text word”.

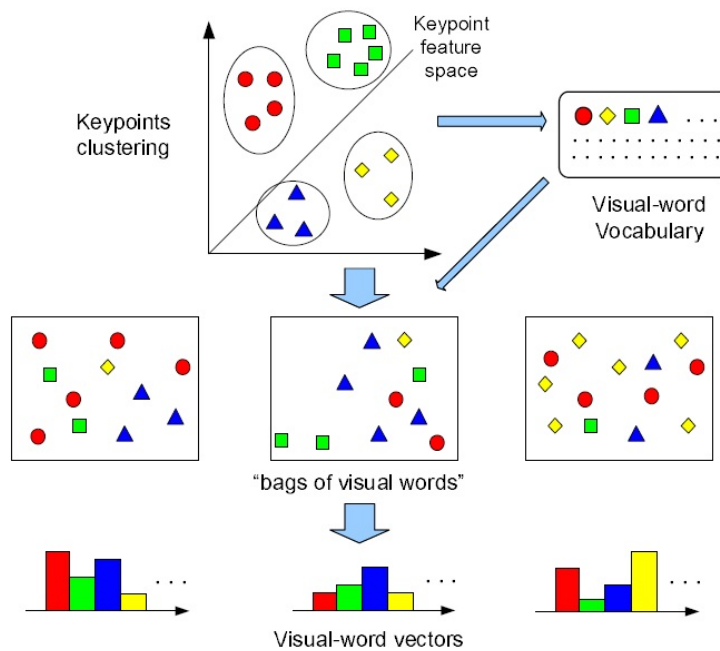


Figure 1.7: Building bag of visual word².

We will now illustrate the construction of BOW from SIFT features. At this stage, each interest point is represented by a 128-D vector, or a point in a 128-D space (called key point feature space in figure 1.7). A dataset consists of several hundred images, and an image may also consist of several hundreds interest points, thus the whole data set can be described as a huge set of points in this 128-D space. In this space, a large amount of those points might be close and could be described as a unique “point” (i.e. a unique visual word) to obtain a description less sensitive to small variations.

Specifically, a clustering algorithm (e.g. K-means) is used to cluster the total amount of descriptors described in the SIFT 128-D space. The number of clusters

²<http://www.ifp.illinois.edu/~yuhuang/samsung/bagofwords.jpg>

determines the size of the *visual vocabulary* (i.e. the number of visual words), which can vary from hundreds to over tens of thousands. Each cluster has a prototype (typically its center points) called the *visual word*. All the points that belong to this cluster will be mapped to this prototype.

Finally, within an image, replacing all interest points with their corresponding visual words produces a set of visual words called the bag of visual words. This BOW representation can be converted into a visual-word vector by counting the number of appearance of each visual word within the image (or within the bag).

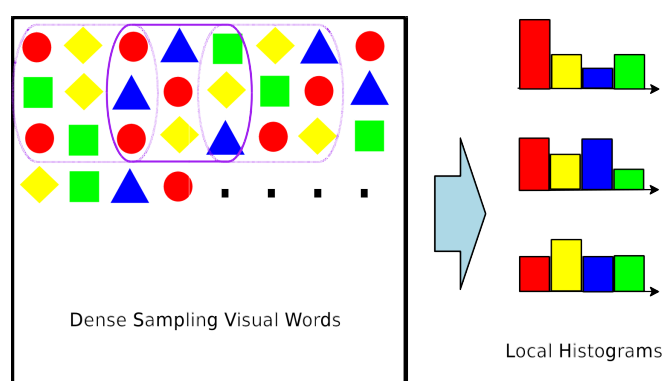


Figure 1.8: Building local BOW features.

This popular BOW construction has been originally used at a global scale, i.e. one image is usually represented as a single histogram of visual words. There is another approach called *local BOW* in which histograms are built from multiple small patches within the image (Figure 1.8). Each patch is the spatial neighborhood of a certain interest point. Consequently, applying local BOW, one image is represented as multiple histograms. An advantage of this approach lies within the local information captured by local BOW [10].

1.1.2 Structure Analysis

The building blocks of a *video*, at a primitive level, are *frames* which are still images. At the intermediate level, there are 3 main notions : *shot*, *scene* and *clip*. A *shot* is a series of frames taken by a single camera without breaks. A *scene* is a shot or series of shots constituting a continuous action (for example, a scene of a dialog between two people might contain many shots of talkers). *Clip* is used to mentioned a short video, usually part of a longer recording. When dealing with videos, it is uncommon to process all the frames (there are about 1500 frames per minute), thus researchers have decided to

work with a smaller portion of frames, called *keyframes* which are still discriminative enough to represent videos. In the following we focus on state-of-the-arts methods to perform this structure analysis and, in particular shot boundary detection, keyframe extraction, and scene segmentation. In the rest of this document, we will rely on some of these methods (we do not make contributions on them) to tackle our problem.

1.1.2.1 Shot Boundary Detection

A shot is defined as a consecutive sequence of frames from the start to the end of a recording. It often displays a continuous action, thus there are strong correlations between frames within a shot. The start and the end of a shot are called shot boundaries. The task is then to detect these shot boundaries in the original video. Generally, shot boundaries can be classified into two types: discontinuous and continuous. If the boundary shows a sudden change between two successive shots, it is an instantaneous boundary and often called a “cut”. Vice versa, if the change is gradual and stretches over several frames, it is the continuous one, and often includes fade, dissolve, wipe, etc. Detecting a cut is typically easier than detecting a gradual boundary. There are different approaches to solve this problem [108] but they all contain these three steps: extracting the visual features from the frames, measuring the similarities between them, and detecting boundaries based on the dissimilarities.

Since the number of frames is often large (at least 24 frames per second) within a shot, the low-level features extracted should be simple and efficiently computable. They often include color histogram [38], [45], motion vectors [72], etc. Some authors even tried with more complex features such as SIFT features [14] or other descriptors extracted with an interest points detector [32]. However, in general, all these more complicated features can not outperform the basic color histogram [108].

The similarities between the frame features can be measured using for example [11, 18, 56]:

- the classic Euclidean distance:

$$d(\mathbf{q}, \mathbf{p}) = \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$$

- the intersection kernel:

$$k(\mathbf{q}, \mathbf{p}) = \sum_{i=1}^n \min(q_i, p_i)$$

-
- the chi-squared metric:

$$d(\mathbf{q}, \mathbf{p}) = \sum_{i=1}^n \frac{(q_i - p_i)^2}{2(q_i + p_i)}$$

The techniques include pair-wise measuring (compare successive frames) and window measuring (compare frames within a window). The window-based one is more effective but also more computational than the pair-wise one.

Then, once computed the features and their similarities, one has to detect the shot boundaries. Current approaches can be roughly divided into two types: threshold-based and learning-based approaches.

- **Threshold-based Approach:** the threshold can be global [13], adaptive [99], or combination of both [75]. The global threshold is used for the entire video, and the adaptive threshold is calculated locally with a sliding window. The adaptive threshold often returns better performance compared to the global one. However, estimating the parameters for the adaptive threshold is more difficult.
- **Learning-based Approach:** the boundary detection problem is converted into a binary classification task in which frames are classified as boundary or not. Both supervised and unsupervised learning are used. Support Vector Machine (SVM) [15], [118] and Adaboost [117] are the two most commonly used supervised classifiers.

1.1.2.2 Keyframe Extraction

The next step after detecting shots is trying to succinctly represent each shot as one or several most informative frames. These frames are called keyframes. According to [87], there are many categories of keyframe extraction techniques.

The simplest technique consists in extracting the keyframes by evenly sampling the video (two keyframes a second, or two keyframes a shot [62], for instance).

Another commonly used technique consists in iteratively comparing each and every frame with the most recent keyframe until finding a frame that is very different. That frame then is selected as the next keyframe. Simple feature such as color histogram can be used with this technique [116]. Another similar technique is constructing a reference frame for each shot, it can be an average histogram [29], or a maximum occurrence frame [83]. Then, a distance is calculated between each frame in the shot and the constructed/reference frame. Keyframes are extracted using the distance curve (at the peaks, for example). Some researchers even consider both keyframe extraction and

object detection to make sure that the keyframes will contain information about objects [48]. Clustering-based approach [119] is another method to extract keyframes. After all frames are clustered, one representative frame is then extracted from each cluster to become the keyframe. The similarity function of clustering algorithm is calculated as the overlapping of two color histograms (i.e. using an intersection kernel).

Since the definition of a keyframe is subjective, there is no consistent method to evaluate the quality of these keyframes. However, the error rate and the video compression ratio are two popular measures that can be used as evaluations. In general, given a video and a set of keyframes, a compromise between low error rate and high compression ratio is often taken.

1.2 Pattern Mining

The features presented before were the core of the research in computer vision for many years. However, many recent papers [2, 50, 82] have shown that global features (such as a single histogram per images), very local features such as SIFT descriptors or even BOW are not discriminative enough to tackle very challenging image or video analysis problems such as the ones provided for example in the PASCAL VOC challenges [27] and in the TRECVID competition [79]. In particular, there has been a very recent focus on making use of pattern mining techniques to obtain more discriminative features from the BOW representation of images and videos (see Section 1.2.4).

In this section, we will start with a general introduction on pattern mining. Then, we will focus on two recent heuristic pattern mining algorithms based on the “minimum description length” principle: KRIMP and SLIM. The latter is the main algorithm that we will use in the following chapters. Finally, we will conclude with a bibliography on the pattern mining methods currently used in computer vision.

1.2.1 Overview

Pattern mining is a subfield of data mining which aims at finding regularities (*patterns*) in some given data. Depending on the type of data, these patterns can be sets (*itemsets*), sequences or graphs. We first provide the basic vocabulary used in the pattern mining field and we present in more details different itemset mining algorithms and their characteristics. Three of them are extensively used in the rest of this thesis.

Transaction ID	Attributes				
	Beer	Bread	Soda	Diaper	Milk
1	0	1	1	0	1
2	1	1	0	0	0
3	1	0	1	1	1
4	1	1	0	1	1
5	0	0	1	1	1

Table 1.1: A market basket record dataset.

1.2.1.1 Generalities over Pattern Mining Datasets

A *dataset* can be considered as a collection of *data objects*. There are many other names for *data object*, depending on the context: *record*, *point*, *vector*, *pattern*, *event*, *sample*, *observation* to name a few. Each *data object*, in turn, is described by multiple attributes such as the mass, or the availability of an object. *Attributes* are also named *variables*, *characteristics*, *features* or *dimensions*. Datasets can be roughly divided into three categories: record data, graph-based data, and ordered data.

Attributes An attribute is a property of an object that may vary from one object to another, or from one time to another. It can have a small number of possible values (*categorical attributes*) or unlimited ones. For example, the attribute “eye color” has a limited number of possible values that change from person to person, and the attribute “temperature” of an object has unlimited possible values that change over time. With the same dataset, there are many options to choose appropriate attributes. For example, in table 1.2, the attributes are “the first item in the basket” etc. and the possible values are “Beer, Bread, Soda. . .” However, that dataset can be represented as in table 1.1, the attributes then are “whether Beer is bought or not”, “whether Bread is bought or not” . . . and the corresponding values are binary: “1” means *yes* and “0” means *no*.

Record and Transactional Data In the standard setting, the data set is a collection of records, each record contains a fix number of attributes. The data can thus be represented as $m \times n$ matrix where m is the set of objects or examples and n is the set of attributes. Figure 1.1 gives an example of such record data in the special case

Transaction ID	ITEMS
1	Bread, Soda, Milk
2	Beer, Bread
3	Beer, Soda, Diaper, Milk
4	Beer, Bread, Diaper, Milk
5	Soda, Diaper, Milk

Table 1.2: The transactional dataset corresponding to the record dataset shown in Table 1.1.

where the data contains information about some customer purchases in a supermarket (we call it *market basket data*).

In the case of *market basket data*, the number of possible *items* that a customer can purchase (the attributes of the table) is often much larger than the actual number of items in his/her basket. A *Transaction* table is a more “compact” representation of the record table where only the non-zero values are kept. In our market basket example, the products purchased by a customer constitutes a transaction, while each individual product is an item. Table 1.2 shows a sample transaction data set. Each row represents the purchases of a certain customer at a certain time. Note that to make it even more compact, the attributes name can be re-encoded using numbers.

1.2.1.2 Frequency and Apriori Principle

In the seminal SIGMOD paper of Agrawal et.al. ([3] which introduced the first pattern mining algorithm, the patterns were used to create association rules. This first algorithm mined singleton patterns called *itemsets* (i.e. sets of *items*). An association rule is a rule $I \rightarrow J$ where both I and J are itemsets and which means that there is an “association” between the occurrence of an itemset I and of an itemset J in the database. The number of transactions that include a particular itemset in a database is called the *support* of this itemset (written $sup(I)$). The *frequency* of an itemset ($freq(I)$) is its support divided by the number of transactions in the database. In the original paper, the authors were interested in mining association of products that were often bought together in a market basket database. They were thus interested in *frequent association rules*. The frequency of an association rule is defined as $freq(I \cup J)$. The association rule search problem boils down to finding the set of patterns $I \cup J$ that are frequent enough (according to a *minsup* or *minfreq* threshold). Note that the *size* of

an itemset is the number of items it contains: a 1-length itemset is a single item.

A brute-force approach to find the frequent patterns (and then the association rules) would be to enumerate them all and test their frequency over the given dataset. However, this approach is exponential in the number of possible items (i.e. attributes) in the database: with only 20 unique items, there are $2^{20} \approx 10^6$ possible candidates frequent patterns for which a support must be computed. Besides introducing the first frequent itemset mining algorithm called *APRIORI*, the authors of [3] also used a simple anti-monotonic principle over the frequency of the itemsets which has been used subsequently in all other pattern mining algorithm. This principle (also known as the *Apriori principle*) states that *if an itemset is frequent, then all of its subsets must also be frequent*. In other words, if an itemset is found infrequent, all its supersets can not be frequent. This principle, illustrated in Figure 1.9 in the case of APRIORI, is used to drastically reduce the number of frequent candidates during a frequent pattern search and thus reduce the computation time of all algorithms. Besides, if the dataset is divided into several partitions, an itemset can be frequent only if it is frequent in at least one partition. This property opens the door to divide-and-conquer algorithms [39, 114] for finding frequent patterns.

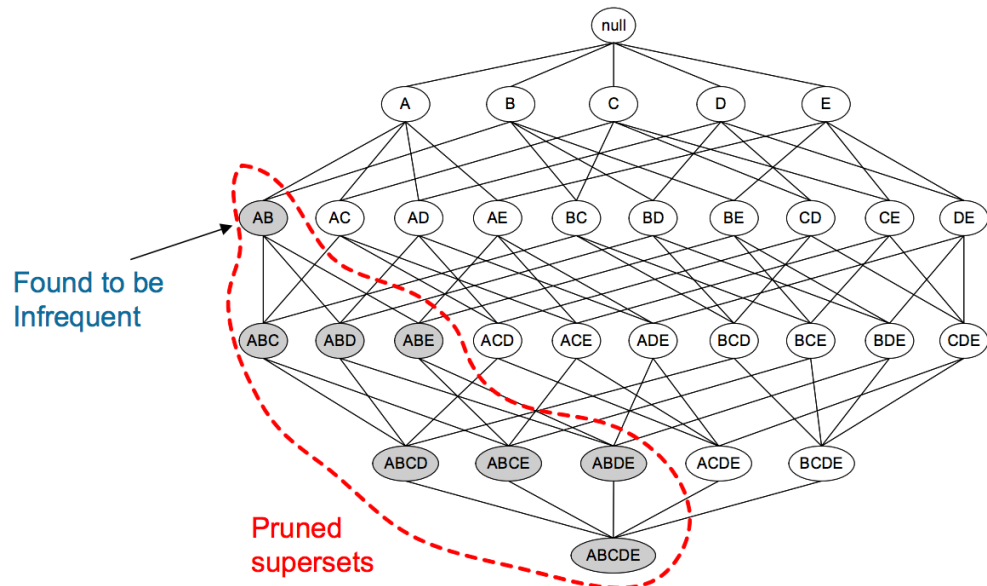


Figure 1.9: An example of apriori principle based pruning. (source: [69])

1.2.1.3 Itemset Mining Algorithms

In general, pattern mining search algorithms can be roughly categorized into two approaches: *breadth-first search* and *depth-first search* (see Figure 1.10). The most popular example for breadth-first approach is the Apriori algorithm [3], and FP-Growth [39] is a representation for depth-first approach. Besides the search strategies, these algorithms also differ in the way they compute the support of an itemset.

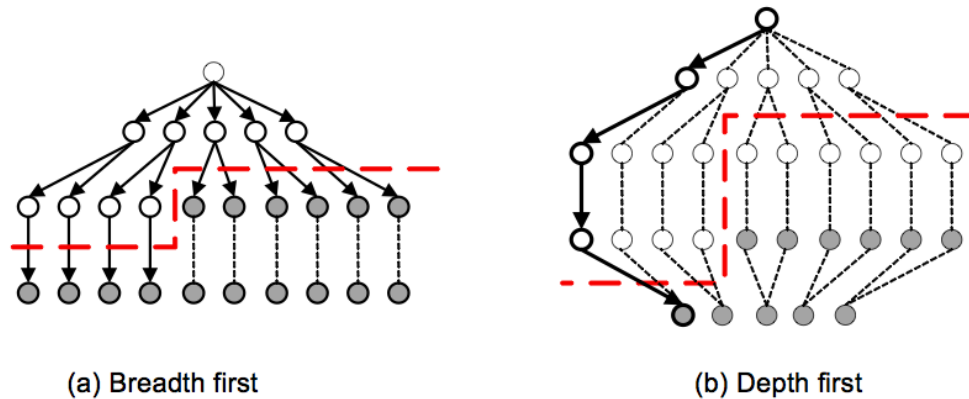


Figure 1.10: Breadth-first search vs Depth-first search approaches. (source: [69])

The breadth-first Apriori algorithm [4] starts by considering all individual items (which can be viewed as itemsets with length 1) and taking out 1-length frequent patterns. The 2-length frequent patterns are chosen from a set of candidates, which are obtained by combining every item to each 1-length frequent pattern. Let D_k the set of frequent patterns of size k . Apriori algorithm computes candidates for D_{k+1} from D_k using the same technique: combining each k -length frequent patterns. Then the Apriori principle is used to ensure that all candidates containing subsets of length k that are infrequent are pruned. Counting the support of each remaining candidate by scanning through the whole dataset, and eliminating candidates that are infrequent result in the frequent patterns of length $k + 1$. The progress continues iteratively until $D_k = \emptyset$ and stops at such k (notice that if $D_k = \emptyset$, then $D_{k'} = \emptyset$ for any $k' > k$). Apriori uses a hash structure to reduce the number of comparison when scanning dataset, thus reducing the complexity of checking supports for candidates.

Apriori generates candidates and tests if they are frequent: both steps are expensive. If there are 10^4 frequent 1-itemsets, the Apriori algorithm will need to generate more than 10^7 2-itemsets and test their frequencies. FP-Growth [39] is proposed with these

difficulties in mind: it allows discovering frequent patterns without explicit candidate generation. The crux of FP-Growth lies within a special representation of the dataset called a FP-Tree (for Frequent Pattern Tree). Once an FP-tree has been constructed, the algorithm uses a recursive divide-and-conquer approach to mine the frequent patterns.

The standard frequent pattern algorithms find all frequent patterns. Besides, if there is a frequent pattern of size k , then according to the Apriori principle, all its $2^k - 2$ subsets are frequent. In many cases, only the frequent patterns that have the maximum number of items are needed, thus an exponential number of subsets are wastefully generated. Such frequent patterns are called *maximal frequent patterns*. By definition, a pattern is maximal frequent if none of its supersets is frequent. The one downside of a maximal frequent pattern is that, even though all the subsets are frequent, the actual support of those are not available. That is how the *closed frequent patterns* came into picture: a pattern is closed if none of its supersets has the same support as the itself. Closed frequent patterns are more widely used than maximal ones because they contain the support of the subsets, so no additional information is needed to find the support of all frequent patterns.

LCM [91] stands for *Linear time Closed itemset Miner* is one of the fastest implementations to mine closed frequent patterns. There are also *LCMfreq* and *LCMmax* implementations for mining normal frequent patterns and maximal frequent patterns, respectively. For the sake of simplicity, the term LCM is used to indicate all these three algorithms. Theoretically, the complexity of LCM is bounded by a linear function with respect to the number of closed frequent patterns, thus it is called *Linear time Miner*. *LCM* uses a *prefix preserving closure extension* to completely enumerate closed itemsets. This allows counting the support of an itemset efficiently during the mining process. It includes a strong pruning method to further reduce the computation time when the number of large frequent itemsets is small. It also generates closed itemsets with no duplication. For all these reasons, we will use this very efficient and fast implementation when mining itemsets during this thesis.

1.2.1.4 Sequence Mining Algorithms

Subsequences are another very popular type of pattern. The sequential pattern mining problem was first stated by Agrawal and Srikant [5]. Each transaction is a set of items, and each sequence is a list of ordered transactions. A frequent sequential pattern is a sequence whose appearance in the dataset is greater than a pre-defined threshold. There

are many applications involving sequence data such as temporal market basket analysis, telephone calling patterns, events in networking environment, biological sequences, time series analysis etc. A number of algorithms to tackle the problem of extracting the complete set of frequent sequences in a dataset has been developed over the years. For example, SPADE (Sequential PAttern Discovery using Equivalence classes) [113] is based on the principles developed for the ECLAT algorithm [112] for frequent itemset mining. It adopts a divide and conquer strategy using tidlist for support counts. SPAM [7], PrefixSpan [70, 71] and Clospan (Closed Sequential pattern mining) [101] are other well known depth-first search sequential mining algorithms. They all try to implement smart strategies to enumerate the candidate subsequences while storing as few information as possible into main memory. The last one even limits the output of the algorithm to the closed sequences which are the sequences with no super-sequence with the same support. More information on those algorithms can be found in [60]. The problem of mining sequential patterns using a heuristic such as the compression-based principle described in the next section has been tackle in [85]. However, to the best of our knowledge, this method is limited to a restricted case of episodes (sequences of items) and it cannot tackle sequences of itemsets.

1.2.2 KRIMP and SLIM Heuristic Algorithms

The output of standard frequent pattern miners is often a very large number of highly redundant patterns. This leads to one of the major problems with the classic frequent pattern mining: it is difficult to find a small but good and informative set of patterns from such a huge number of output. The MDL principle is a criterion to select a good subset of the complete set of patterns. MDL stands for “minimum description length”, and its principle is that “the best set of frequent patterns is that set that compresses the database best”.

1.2.2.1 Minimum Description Length (MDL)

MDL [35] is a practical version of Kolmogorov Complexity [51]. They all follow the *Induction by Compression* philosophy. The principle of MDL can be described roughly as the following:

Given a set of model \mathcal{M} , the best model $M \in \mathcal{M}$ is the one that minimizes

$$L(M) + L(\mathcal{D}|M) \tag{1.1}$$

in which $L(M)$ is the length of the description of M , and $L(\mathcal{D}|M)$ is the length of

the description of the data \mathcal{D} when encoded with M . Both are measured in bits.

In order to apply MDL to the pattern mining scenario, three questions need to be clarified: what the models \mathcal{M} are (and in particular a model $M \in \mathcal{M}$), how can M describe a database, and how can M and \mathcal{D} be encoded in bits. These 3 questions can be paraphrased as: what is M , what is $\mathcal{D}|M$, and how are $L(M)$, $L(\mathcal{D}|M)$ calculated. Note that with MDL, the main concern is the code lengths, not the semantics of the code words.

The key idea of the MDL-based pattern mining approach proposed by Siebes et. al. [96] is the use of a *code table*. A code table is a simple two-column dictionary with item sets on the left-hand side and corresponding codes on the right-hand side (see Figure 1.11). The itemsets in the code table are ordered first descending on length, second descending on support, and third following alphabet order. And the code table must contain at least all single items constructing the whole database. As mentioned before, the actual codes in the right-hand side are not important, their lengths are. To explain how these lengths are computed, the second question must be answered first: how is a database encoded by using the code table?




<i>Itemset</i>	<i>Code</i>	<i>Usage</i>
A B C		5
A B		1
A		1
B		1
C	—	0

Figure 1.11: Code table example (from [96]). The widths of the codes indicate lengths. The *usage* column is here only to demonstrate the more a code is used, the shorter it is.

Take an arbitrary transaction t as an example to be encoded by a code table. The first step is searching for the first item X in the code table such that $X \subseteq t$, the search order is the order of itemsets of the code table. The code of X then becomes a part of the encoding of t . If $t \setminus X \neq \emptyset$, the search is repeated to encode $t \setminus X$. The process keeps going on until $t \setminus X = \emptyset$, and the set of itemsets that are used to encode transaction t is called its *cover*. Since the code table contains all singletons, this algorithm always guarantees returning an encoding to any transaction. Figure 1.12 gives an example of a database and its corresponding cover as well as encoded length.

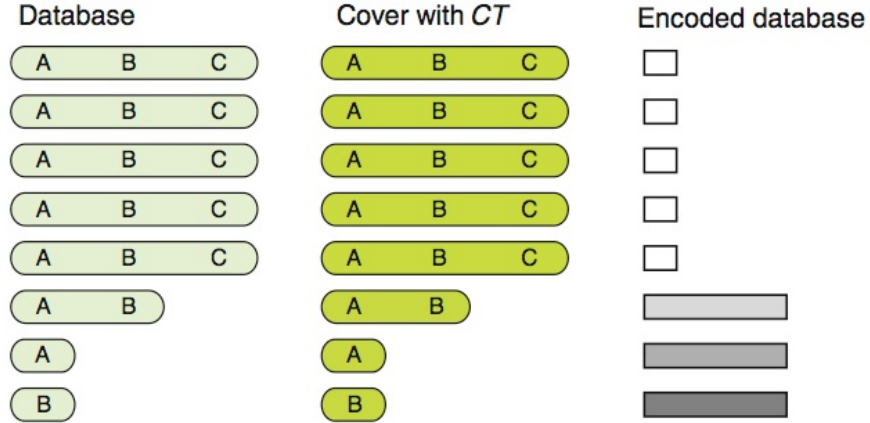


Figure 1.12: A database example with its cover and encoded length derived from the code table shown in Figure 1.11. (source: [96])

With this encoding knowledge is at hand, we can cover all transactions in the database, and computing code lengths becomes a feasible task. The concept is that: the more often a code is used, the smaller its length should be.

The *usage* of an itemset X in the code table CT is the total number of transactions which contain X in the encoding process. Thus the relative usage of X is the probability that X is used to encode an arbitrary transaction t . Borrowing from information theory field, the Shannon entropy [19] provides the length of optimal code for X :

$$L(X|\mathcal{D}) = -\log Pr(X|\mathcal{D})$$

where

$$Pr(X|\mathcal{D}) = \frac{usage(X)}{\sum_{Y \in CT} usage(Y)}$$

These optimal values help forming the right-hand side column of the code table, thus we can use $L(X|CT)$ and $L(X|\mathcal{D})$ interchangeably. Note that the above model M is now become the code table CT . From this information, the length of the encoding of a transaction can be easily calculated as the sum:

$$L(t|CT) = \sum_{X \in t} L(X|CT)$$

And thus, the second element of equation 1.1, or the size of the encoded database

is:

$$L(\mathcal{D}|CT) = \sum_{t \in \mathcal{D}} L(t|CT) \quad (1.2)$$

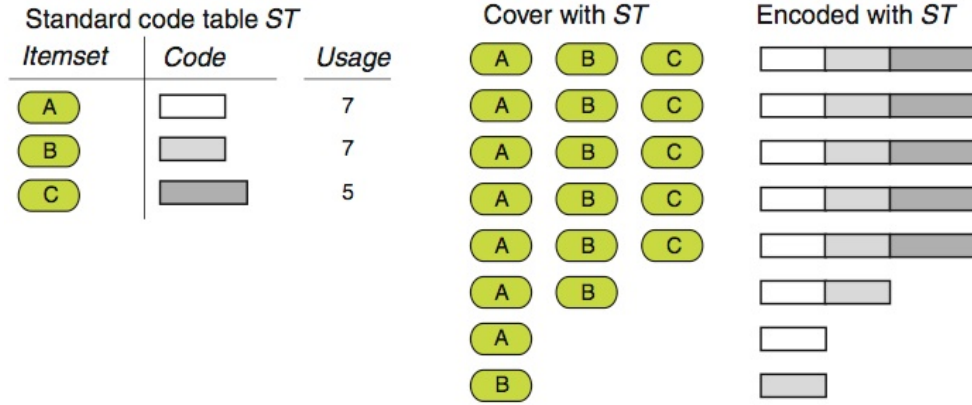


Figure 1.13: A example of **Standard Code Table** with its cover and encoded length derived from the database shown in Figure 1.12. (source: [96])

To use the MDL principle, we still need to know what $L(CT)$ is, i.e. the encoded size of a code table. Recall that a code table is a two-column table consisting of itemsets and codes. The simplest code table is the one that contains only the single items, and it is called **Standard Code Table**, or *ST* for short. Figure 1.13 demonstrate an example of *ST* in action. This definition is needed to encode the first column of the code table: all itemsets of the left-hand side column are encoded by the code table *ST*. Taking the sum of all these encodings results in the size of the first column of the code table. The size of the second column is just the sum of all the code lengths. Basically, the size of the code table *CT* is given by:

$$L(CT|\mathcal{D}) = \sum_{X \in CT} (L(X|ST) + L(X|CT)) \quad (1.3)$$

With results from equation 1.2 and 1.3 the total size of the encoded database can be calculated: it is simply the size of the encoded database plus the size of the code table. That is, the criteria 1.1 can be rewritten as:

$$L(\mathcal{D}, CT) = L(CT|\mathcal{D}) + L(\mathcal{D}|CT)$$

The optimal set of item is the one that its corresponding code table minimize above criteria.

However, searching for this optimal set of items is not a simple task. First of all, the search space is too large to be considered exhaustively. Secondly, there is not a simple way to know what the effect will be on the compression before adding an element to a code table [96]. Thus, making use of heuristics is a realistic option. Two following sections will devote to two prominent heuristic approaches to solve MDL problem: KRIMP and SLIM. KRIMP is Dutch for *to shrink*, and SLIM means *smart*.

1.2.2.2 KRIMP Algorithm

The KRIMP algorithm [96] needs a set of frequent patterns to be able to start with. The main task of KRIMP is filtering a pre-mined set of candidates, trying to select the best set of itemsets which compress the original database well. The pre-mined set can be a set of classic frequent patterns, like closed frequent patterns or maximal frequent patterns.

The basic greedy heuristic strategy is following:

- Start with the standard code table ST which consists of all the single itemsets.
- Add the itemsets from the pre-defined set one by one into the code table. With each itemset, if the newly added code table produce a better compression, the itemset is kept. Otherwise, discard the itemset.

Algorithm 1 The KRIMP Algorithm

Input: Database \mathcal{D} , candidate set \mathcal{F}

Output: A heuristic optimal code table CT

```

1:  $CT \leftarrow \mathbf{ST}$ 
2: for each candidate  $F \in \mathcal{F}$  do
3:    $CT_c \leftarrow (CT \oplus F)$ 
4:   if  $L(\mathcal{D}, CT_c) < L(\mathcal{D}, CT)$  then
5:      $CT \leftarrow \text{post-prune}(CT_c)$ 
6:   end if
7: end for
8: return  $CT$ 

```

The pseudo code of KRIMP is shown in Algorithm 1. It takes as input the database \mathcal{D} and a set of pre-mined candidates \mathcal{F} at a given support *minsup*, and the expected output is a heuristically optimal code table. The first step is assigning the **Standard Code Table** ST , which contains only singletons, to the code table (line 1). Then, each candidate F is considered one by one (line 2) by putting it into the code table CT

(line 3). Two questions are raised here: in which order the candidate $F \in \mathcal{F}$ should be considered, and in which order the candidate F should be put in the code table CT ?

To answer the first question: the candidates are ordered first descending on support, second descending on length, and third in alphabet order (for line 2). The reasoning for it is that: long, frequently occurring itemsets should be given high priority. Itemsets with the highest support, those with potentially the shortest ones, end up at the top of the list. Of those, the longest sets are preferred, as they will be able to replace as many items as possible.

About the order in code table CT , the elements are ordered first descending on length, second descending on support, and third in alphabet order (for line 3). The reasoning is as follows. To obtain a good compression ratio it is desired to replace as many individual items as possible, using as few as possible codes. The above order gives priority to long itemsets, as these can replace as many as possible items by just one code. Also we prefer those itemsets that occur frequently in the database to be used as often as possible.

The next step is calculating the new compressed size, and the candidate is only accepted if the compression improves (line 4). Lastly, all elements $X \in CT_c$ are reconsidered to remove elements which are not improve the total compression (line 5).

Pruning itemsets from the code table CT is an important step. If the total encoded size decreases by pruning an itemset, that itemset is permanently removed from the code table. It help controlling the size of the code table, and retaining only the best set of items by removing not-useful itemsets. Its net result is a speed-up as code table is kept smaller and the whole process thus needs to consider fewer itemsets. Since the pruning process only apply if the candidate F is accepted, and only few candidates are, this effectively reduces the pruning search space. Also, not all subsets of CT is considered for pruning, only itemset whose *usage* decreases is considered for removal. The rationale is that for itemsets whose code lengths have increased, there is a high chance that these sets now harm the compression.

The algorithm is simple and straightforward. It provides a solution to the well-known explosion in the size of frequent patterns in pattern mining field. It reduces the huge number of highly redundant frequent patterns to only a small number of high-quality patterns.

However, KRIMP has some drawbacks originated from its simplicity. The first one is its input: the pre-mined set of candidates. This candidate set is often very large (millions of patterns), thus merely mining, sorting, storing them takes lots of time and effort, even before KRIMP can start its duty. Secondly, almost all of patterns

will be rejected, very few patterns (out of millions) will be selected. Thus spending time checking a pattern while knowing the pattern is rejected with 99% probability is somehow irony. Lastly, KRIMP is a rough greedy heuristic approach, it considers each candidate only once, in a static order thus it sometime rejects candidates that could be useful later.

1.2.2.3 SLIM Algorithm

Unlike KRIMP, SLIM [80] does not need to start with a set of itemsets. Instead, SLIM adopts a bottom-up approach: it slowly add new itemset, which is a combination of current itemsets of the code table.

Algorithm 2 The SLIM Algorithm

Input: A database \mathcal{D}

Output: A heuristic optimal code table CT

```

1:  $CT \leftarrow \mathbf{ST}$ 
2: for each candidate  $F \in \{X \cup Y : X, Y \in CT\}$  do
3:    $CT_c \leftarrow (CT \oplus F)$ 
4:   if  $L(\mathcal{D}, CT_c) < L(\mathcal{D}, CT)$  then
5:      $CT \leftarrow \text{post-prune}(CT_c)$ 
6:   end if
7: end for
8: return  $CT$ 

```

The pseudo code of SLIM is shown in Algorithm 2. It is very similar to the pseudo code of KRIMP in Algorithm 1 with some slight differences. It takes only one input: the database \mathcal{D} (it does not need candidate set), and the expected output is a heuristically optimal code table. SLIM start by assigning the **Standard Code Table** \mathbf{ST} , which contains only singletons, to the code table (line 1). Then, every iteration, all pairwise combination of $X, Y \in CT$ is considered as candidate (line 2) by putting it into the code table CT (line 3). The next step is calculating the new compressed size, and the candidate is only accepted if the compression improves (line 4). Lastly, all elements $X \in CT_c$ are reconsidered to remove elements which are not improve the total compression (line 5).

The order of inserting candidate into the code table CT is exactly the same as with KRIMP case: first descending on length, second descending on support, and third in alphabet order. But the order of considered candidates is different, since there is not set of candidates in SLIM algorithm. This order is determined based on the gain in bits of the newly created code table: whatever pair $X, Y \in CT$ that gives the highest

gain when putting in the code table will be considered first. The most straightforward tactic is to calculate all gains for every possible candidates, and just select the best one. However, this approach is computationally expensive, since to compute the gain for a certain candidate, the whole database must be scanned.

One approach to partly solve this difficulty is to *roughly estimate* all the gains of all candidates first, and only calculate the exact gains for top-k candidates. The priority is finding an easily calculable estimate. Let X and Y be itemsets in CT , CT' be the code table after adding X and Y (i.e. $CT' = CT \oplus X \cup Y$), ΔL be the difference in encoded size between CT and CT' :

$$\begin{aligned}
\Delta L &= \Delta L(CT \oplus X \cup Y, \mathcal{D}) \\
&= L(CT, \mathcal{D}) - L(CT \oplus X \cup Y, \mathcal{D}) \\
&= [L(CT|\mathcal{D}) + L(\mathcal{D}|CT)] - [L(CT \oplus X \cup Y|\mathcal{D}) + L(\mathcal{D}|CT \oplus X \cup Y)] \\
&= \Delta L(CT \oplus X \cup Y|\mathcal{D}) + \Delta L(\mathcal{D}|CT \oplus X \cup Y)
\end{aligned}$$

Before developing this equation into details, several brief notations need to be defined for the sake of simplicity. Lower case, x , of an item X is used to indicates $usage(X)$ in the code table CT , x' is $usage(X)$ in the code table $CT' (= CT \oplus X \cup Y)$. s is the sum of all usages in CT , i.e. $s = \sum_{X \in CT} x$. Similarly, s' is the sum of all usages in the code table CT' .

Applying equation 1.3 into the first element, and performing suitable transformations, we end up with:

$$\begin{aligned}
\Delta L(CT \oplus X \cup Y|\mathcal{D}) &= \log xy' - L(X \cup Y|ST) + \sum_{c \in CT} \log \frac{c}{c'} \\
&\quad + |CT| \log s - |CT'| \log s'
\end{aligned}$$

Likewise, applying equation 1.2 into the second element, we have:

$$\begin{aligned}
\Delta L(\mathcal{D}|CT \oplus X \cup Y) &= s \log s - s' \log s' + xy' \log xy' \\
&\quad - \sum_{c \in CT} (c \log c - c' \log c')
\end{aligned}$$

Calculating exact values for these equations is not an easy task. Besides, the current essential task is finding an effective technique to *estimate* values. The authors of SLIM then made an assumption that only the *usage* of X , Y , and $X \cup Y$ will change. Consequently, many elements in two above equations will eliminated themselves, the

rest can easily be shorten using following relations: $x' = x - xy'$, $y' = y - xy'$, and $s' = s - xy'$. In the end, a very easily calculable estimate is obtained by making use of a simple assumption.

The most attractive feature of SLIM is that it doesn't need a pre-mined set of candidates. This feature alone saves lots of time for mining, storing... patterns. Considering the same dataset, the number of candidates examined by SLIM is often much smaller than the size of the candidate set of KRIMP. Sometimes, SLIM might scan more candidates than KRIMP, and the reason usually is SLIM considers itemsets at lower support than KRIMP can handle.

SLIM does not work well with sparse database. Both KRIMP and SLIM prefer dense data, in which finding structure seems easier.

1.2.3 Converting Image Descriptors into Binary Vectors

As explained before, our aim is to use existing pattern mining techniques on video datasets. Our first problem is thus to convert the original image representation into a suitable transactional or binary dataset usable by a pattern mining algorithm. In the rest of this document, we will mainly use SIFT-BOW histograms to describe the frames of our videos. We thus focus on the methods to convert such histograms into binary datasets. A SIFT-BOW histogram can be seen as a vector where the dimension is the number of visual words chosen during the BOW construction (see 1.1.1.4) and the value of each vector element (or *bin*) is the number of occurrences of the corresponding visual word in the considered image region.

The most naive way to do this conversion is to turn all different-from-0 values into ones, and keep the "0" values untouched. This discretization technique is denoted Simple-Binarization (SBin) in the rest of this manuscript. One variant is fixing the number of "1" by converting only top- K values into "1", and the remaining values into "0" [95]. However, we can notice that this procedure remove a lot of information. There are several ways to overcome this limitation. The first one is to improve this simple method by discretizing the values in each histogram bin into multiple intervals (also called bins). For example, if the number of interval chosen for the discretization process is "4", one can represent each interval by one binary code: 00, 01, 10, 11. The discretization thresholds (that are used to create the bounds of the intervals) can be uniformly chosen (for example, $[0 - 2]$, $[2 - 4]$, $[4 - 6]$, $[6 - 8]$) if, for example, the values in the original bins can get up to 8 or depend on the distribution of the values that fall into each bin.

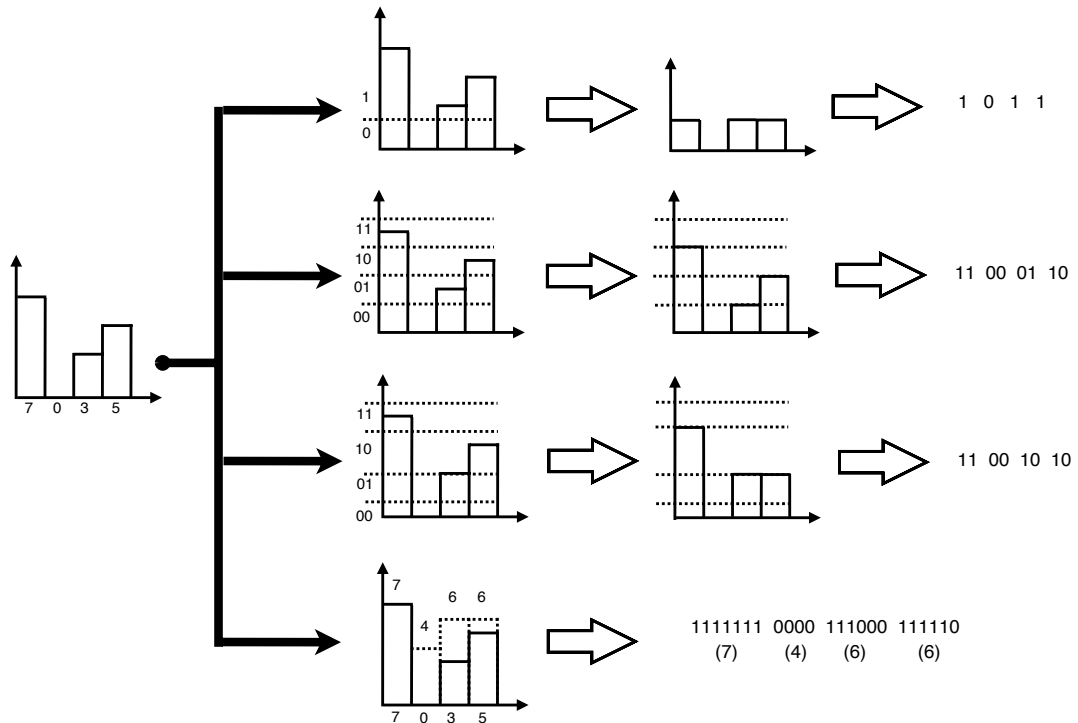


Figure 1.14: Examples of different binarization techniques. From top to bottom: SBin, multiple uniform intervals, non-uniform intervals and B2S.

The B2S (Bag-to-Set) representation [49], in which all the frequency information of the histograms is kept, is another approach. Its name comes from the initial idea: converting a histogram-based bag representation (i.e. bag-of-word original principle) into a set representation. For instance, we want to convert the following histogram into a binary form: $A = [3 \ 1 \ 4 \ 0 \ 2]$. And suppose that $C = [3 \ 2 \ 4 \ 3 \ 4]$ are the maximum values at each position over all histograms. Each value of A is replaced by consecutive values “1” with “0” ending (if applicable, so that the total number of binary digits is equal to the value of C at the corresponding location), for example, $A[5] = 2$ and $C[5] = 4$ then $A[5]_{binary} = [1100]$. Applying this step for all values, we have $A_{B2S} = [111 \ 10 \ 1111 \ 000 \ 1100]$.

In [30] the authors chose another approach that we call Sparse-B2S (or SB2S) which is quite similar to B2S, but instead of using consecutive values “1” with “0” endings, they use only one value “1” and the rests are “0” for one bin. With the same example as above, the corresponding binary code is $[001 \ 10 \ 0001 \ 000 \ 0100]$. Moreover, if, for a given location, a value is never used in any histogram, the corresponding digit will always be a 0. It can thus be removed. For instance, if there is no histogram with value

3 at position 3, then the binary code is finally $A_{SB2S} = [001\ 10\ 001\ 000\ 0100]$.

The different approaches are shown in Figure 1.14.

1.2.4 Pattern Mining applied to Computer Vision

Even though frequent itemset mining techniques and variants thereof are well-established in the data-mining community [91], they are, to date, not commonly used in state-of-the-art image classification methods. This is surprising, since it has been shown that these mining methods allow the construction of high-level sets of compound features which can, in many cases, capture more discriminative information [17].

Frequent pattern mining techniques have been used to tackle a variety of computer vision problems, including image classification [65, 110, 111], action recognition [33, 74], scene understanding [107], object recognition and object-part detection [73].

In the context of pattern mining-based image classification, local bag-of-words are usually preferred (e.g. in [49, 74, 78]), since they result in sparse representations, a better signal-to-noise ratio, an increased robustness to image clutter and some low level spatial information (proximity). Spatial configuration mining based on local BOW was first shown by Quack et al. [73]. More structured patterns such as sequences and graphs capturing the spatial distribution of visual words have been used by [65], while [109] uses boosting on top of binary sets of visual words discovered by pattern mining. Voravuthikunchai et al. [94] make use of closed frequent patterns to detect duplicate images in a very large dataset of one million images. Gilbert et al. [33] have applied itemset mining to action recognition using rather primitive features like corners, while in [111] high level features such as attributes [28] are successfully used with mining techniques. In [107], Yao et al. present a structured image representation called group-lets. To find discriminative group-lets, they mine for class-based association rules [3]. Recently, [95] has used random projection to produce transactions from BoW histograms, and proposed an image presentation based on pattern sets instead of individual patterns. This HoPS (Histograms of Pattern Sets) representation is efficiently used for image classification and objection recognition tasks. Jain and Jawahar [47] represent a shot as a transaction of frames and perform frequent pattern mining to find frequent frames. Based on that information, they are able to extract characteristic scenes from movies.

1.3 Current Tagging Methods

As explained in the introduction, our aim in this thesis is to automatically complete and correct tags provided by users for a given video uploaded on the Internet (e.g. on Youtube). According to [98], automatic tagging methods in which a computer provides or helps a user to provide tags for some given multimedia content can be referred to as *assistive tagging* methods. The authors of [98] divide the tagging tasks into 3 categories: 1) tagging with data selection and organization, which consists in organizing the data in such a way that only a small subset of it should be manually tagged to get some global information; (2) tag recommendation, which consists in suggesting relevant tags to a user while he is tagging his/her own video; and (3) tag processing, which consists in refining human-provided tags or adding more information to them. Our tagging problem relates to this last category.

Besides, the authors of the previously mentioned survey ([98]) are covering several types of multimedia content ranging from images to videos through music and speech. In this thesis, we focus on the video tagging problem. A video in itself (as opposed to images) contains multiple sources of information such as its metadata (i.e. texts embedded in videos which generally consist of a title, a list of authors, a copyright, etc.), an audio channel, sometimes a transcript (which can be obtained by speech recognition or by optical character recognition from caption texts) and the visual content included in the frames themselves which is temporally structured. In this thesis, we will only rely on the visual content of the frames and on the tags provided with the video. In this context, algorithms used on images and on videos mostly differ in the feature extraction and representation stages since both videos and images are often ultimately represented as feature vectors. Consequently, most tagging techniques in images processing can be safely applied to video tagging.

Assistive tagging methods are at the crossroad between automatic tagging and manual tagging methods. In the following, we thus first focus on automatic tagging methods and then on specific assistive ones in Section 1.3.3.

1.3.1 Comparing Videos

Even though a video is considered as a sequence of images, video comparison is a completely different problem from the image comparison one. Because each video has a different length, and also because the length of each scene/shot is different, the representations of different videos are usually not comparable. For example, when one wants to classify videos using a SVM classifier or cluster them using e.g. a k-means

clustering algorithm, the inputs of the algorithms must have the same dimensions. Whenever one wants to process videos, comparing videos is a mandatory step. Taking the average of the characteristics of the keyframes, comparing pairs of keyframes and make use of the identical frames are the three main trends to compare videos.

1.3.1.1 Averaging



Figure 1.15: A video is represented as an average of all its keyframes.

The first naive method to compare videos is to average the descriptors of each keyframe extracted from the video (see Section 1.1.2 for the keyframe extraction) to produce a single representation for the whole video. This method is used, for example, in [86] and [104] where a single bag-of-words histogram is produced and thresholded to remove potential noise in the video. In all cases, classical distance function such as the $L1$ or $L2$ distances can be used to estimate the similarity between videos. Even if this method is computationally effective, one loses a lot of the available information by averaging all the frames as can be seen in Figure 1.15. We nonetheless tried this method in our framework.

1.3.1.2 Pairwise

The second most used method consist in measuring the similarity between two videos as the maximum of the pairwise similarity between two key-frames of those two videos (see e.g. [63]). In other words, this method uses the similarity between the two most similar frames of two videos to represent the similarity between the videos themselves. Again, the comparison of the two videos is made using a unique pair of frame and no sequential information is taken into account.

1.3.2 Automatic Tagging

Video annotation (and in particular tagging) and *video classification* are two very popular problems in video analysis. They are related to each other (so they are often described interchangeably) even if the video annotation problem can be considered more



Figure 1.16: The distance between two videos equals to the distance between two most similar keyframes (the red line in this example).

general than the second one. The video classification problem consists in assigning to videos some pre-defined categories or classes (e.g. *news* or *football*). Video annotation consists in assigning to shots or video segments some pre-defined more general semantic concepts, such as *person*, *sky*, *people running*. Both problems share similar solving approaches: first, one needs to extract low-level features and then train some classifiers to map the features to the categories/concepts. The two problems of *video indexing* and *video retrieval* are also related to the above problems. In the video retrieval task, a query is given by a user and the retrieval system must effectively and accurately return the videos that match the query. To get this effective retrieval, a good indexing is needed which makes the two tasks dependent on one another. Besides, to match the results with the query, the retrieval system needs to compare the query with the meta-data of the already stored videos. This comparison is often made using distance functions which are also at the core of the machine learning methods used to solve the first two problems mentions. The methods presented in this section will thus cover all these different applications.

1.3.2.1 Automatic Tagging Benchmarks for Videos

The most common benchmarks used to evaluate automatic tagging methods (for video annotation, classification, retrieval or indexing) come from the TRECVID (*Text Re-*

trieval Conference specialized on *Video Retrieval Evaluation*) competition [79]. The TRECVID campaign, sponsored by NIST (*National Institute of Standards and Technology*) since 2001, focuses on research related to videos. During each year competition, a large number of video collections is provided together with a number of tasks. These tasks are proposed by both NIST and participants and they are gradually changing over time depending on the needs of community: tasks with very few participants are eliminated, and tasks attracted to video researchers are added. TRECVID 2013 [68] had 5 main tasks: semantic indexing, interactive surveillance event detection, instance search, multimedia event detection, multimedia event recounting. As explained before, most algorithms used to tackle these tasks are relevant for our auto-tagging problem.

Note that there are many more benchmarks available in the context of image classification and retrieval but, as they do not consider videos, they are not specifically presented in this chapter.

1.3.2.2 Model-based methods

One of the most widely applied method for automatic tagging consists in learning a model for each tag (or learn a multi-class classifier) based on a trustworthy training set and use these models to predict the tags of new data. It is thus seen as a binary classification problem. The model can be learned using any learning algorithm. However, most state-of-the-art papers claim a better accuracy using the well known *Support Vector Machines* (SVM) [26] learning algorithm. For example, Amir et .al [6] train SVM classifiers by using various primitive visual features (color histogram, color correlogram, edge histogram...). Beo et .al [8] use a special technique called *spatial pyramid matching* to represent low-level visual features like SIFT. These visual features are then provided to SVM classifiers for classifying concepts in TRECVID *Multimedia Event Detection* MED dataset. Sun and Nevatia [82] use Fisher Vector instead of BOW model to encode multiple visual features (such as histogram of gradients (HoG), histogram of optical flow (HoF)), then construct SVM classifiers to carry out experiments on the TRECVID MED dataset.

Recent methods [105, 106] have also studied how information learned from images (that are often easier to label) could be transferred to videos using a transfer learning algorithm. Using the same approaches, one can also detect *events* (i.e. series of concepts) in videos [25, 90]. This problem is called *Concept-based event detection*. In the same context, Merler et .al [59] introduced *semantic model vectors* which is an intermediate semantic representation between low-level features (SIFT) and high-level

events (playing football). The authors propose to train hundreds of concept detectors from a collection of thousands of labeled web images represented by a concept vector. A linear SVM classifier is then trained to recognize events from these concept vectors. [37] focuses on the relationship between concepts and events, it studies to find a way to create an effective vocabulary (list of concepts) for event detection task.

Note that these methods suffer for one essential drawback: the number and the name of each tag has to be known in advance and one has to obtain a sufficient amount of tagged videos to train a classifier with each of these tag.

1.3.2.3 Nearest neighbors-based methods

Nearest neighbors-based methods predict the tags of a new example by comparing it to its nearest neighbors (NN) in the training sample. For example, Makadia *et .al* [58] pioneered this approach by constructing a NN-based baseline for image classification task. They introduced a simple method to propagate n tags to a test image from its visual neighbors: taking almost all tags from the closest image, and if there is still not enough n tags, they rank tags of the remaining neighbors and select top-ranked tags. Later, Guillaumin *et .al* [36] learned a weighted nearest neighbor model where a weight π_{ij} denotes the relevance of image j for predicting the tags of image i . These weights can be rank-based (π_{ij} depends on the order of image j in neighborhood of image i) or distance-based (π_{ij} depends on the distance between image j and i).

Again, all aforementioned methods need a training set with a sound ground truth i.e. the tags are not questioned and thus do not need any refinement. The main advantage of these approaches is that one does not need to know the tags in advance nor learn a particular model for each tag.

1.3.3 Tag Processing

As explained before, this section is devoted to techniques that are at the crossroad between automatic tagging and manual tagging methods. Our work relies on the fact that even if tags are often associate with web videos, or large-scale video collections, they are still often incomplete and incorrect. However, by refining noisy tags, adding new ones and removing irrelevant ones, the performance of tag-based search can be greatly improved according to [54]. For example, associating a tag with some information about the shot it refers to in the video (and thus have sequential information over the tags) can be of great value to improve retrieval tasks as shown in [97].

Another strategy consists in applying the automatic-tagging strategies mentioned

above without relying entirely on the existing labels of the images/videos. For example, in [16], each SVM classifier is trained for each tag from the loosely labeled samples. These classifiers return an initial relevance score of the tags with respect to a picture. The score is later refined based on the similarity between images and semantic relations of tags. For example, Liu *et al* [52] use a nearest neighbor algorithm (voting system) in the visual neighborhood of an image to predict the relevance scores of some tags. They later extend their work to multiple visual feature spaces (i.e. global features, local features. . .) in [53]. Another example can be found in [92] where the author evaluates multiple nearest-neighbor approaches for tag refinement tasks.

Our aim is to take advantage of the nearest neighbor-based method and of the tags manually provided by users to improve the relevance of the tags in a video dataset. The method proposed in the following chapters can be seen as a nearest neighbors-based method on the visual content of the videos which take into account the uncertainty on the tags to propagate tags from the neighborhood videos.

Chapter 2

Our Tagging Framework

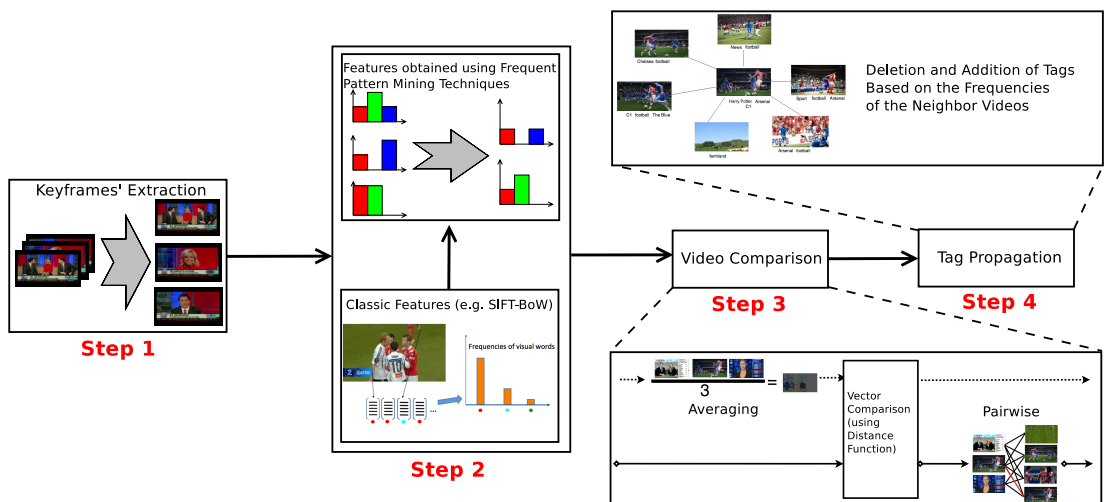


Figure 2.1: The Overall Framework

In this chapter we detail the tagging framework that we propose to tackle our automatic video tag correction task. The overall scheme can be divided into four steps as shown in Figure 2.1. The first step consists in extracting relevant keyframes from our target videos to be able to analyze them. The number of keyframes that should be extracted generally depends on the length and on the movement speed of the objects in each video. This has been discussed in details in Section 1.1.2.1. Since we do not present any contribution on this particular step, we rely on some existing keyframe extraction algorithms (e.g. [22]) or on the data provided by the authors for some benchmark datasets. In the second step, some classical visual features (in our case we rely on SIFT-BoW) are computed to describe the keyframes. We then propose

to build more discriminant mid-level features from the SIFT-BOW using some itemset mining techniques. In step 3, the new mid-level features are used to compare the videos for the auto-tagging problem. The most relevant distance functions are discussed in this chapter. Finally, in step 4, tags are propagated from one video to others based on the visual similarity computed between the videos and the overall distributions of the tags in the dataset.

2.1 Parameters Evaluation

As explained in the previous chapter, one of the difficult part of this thesis is the lack of ground truth labels for the video datasets. This is especially true when the aim is not to classify a video with a single label but to tag it, i.e. to give all the tags that are entirely relevant for the video. To carry on with the entire video tagging process we thus choose to validate a number of hypotheses on a simpler, well defined and well studied problem: the image classification problem. In particular we would like to assess if:

1. (in step 2) the frequent patterns (FP) are truly interesting features compared to simple bag-of-words if they are carefully chosen;
2. (in step 2) a standard Principle Component Analysis (PCA) can help reducing the number of frequent patterns used to describe each images (which is often huge if no post-processing step is carried on) without damaging the classification accuracy;
3. (in step 3) we can already choose a good distance function to compare two high dimensional vectors describing a video (or some video frames);

2.1.1 Image Dataset

To validate our hypotheses, we use two well-known image datasets for which supervised label information is provided: the *GRAZ* dataset [67] and the *Oxford-Flower17* [65] (some examples are shown in Figure 2.2). The *GRAZ* dataset is used to solve two binary classification problems: one to classify images which contain a *bike* (with 200 positive and 200 negative images) and one to classify images which contain a *person* (with the same number of images). For this dataset, 200 visual words are built on SIFT descriptors (this is what is commonly used for this dataset in the literature). The *Oxford-Flower17* is more challenging with 17 classes of flowers (each class has 80



Figure 2.2: Class examples of *Oxford-Flower17*.

images). Example images for 3 classes of this dataset are shown in Figure 2.2. For this one, 100 visual words are built from color descriptors, more precisely ColorName [93] (CN) descriptors, again because this is a common practice in the computer vision literature for this dataset.

2.2 Step 2: Feature Construction

As explained above, this step aims at constructing mid-level features from SIFT-BoW (see Section 1.1.1.3 for a description of SIFT and Section 1.1.1.4 for a description of local and global BoW). To be able to use itemset mining techniques on these histograms, one has to convert the histograms into a transactional or a binary dataset. We used three

binarization methods (see Section 1.2.3) for experiments: Simple-Binarization (SBin), B2S and Sparse-B2S. As explained in Section 1.2, the main problem with pattern mining algorithms is the size of the output. Since our plan is to encode an image using those patterns, the number of kept patterns should not be too large not to suffer from the curse of dimensionality [24]. In the following, we will first show how to use patterns to represent images. After that, we rely on an existing method to chose some good patterns to answer the questions 1 and 3 asked in the previous section. This method is not usable for our target application since we cannot completely rely on the tag to post-process the patterns. We thus propose some unsupervised alternatives to do this post-processing step.

2.2.1 Encoding Images Using Frequent Patterns

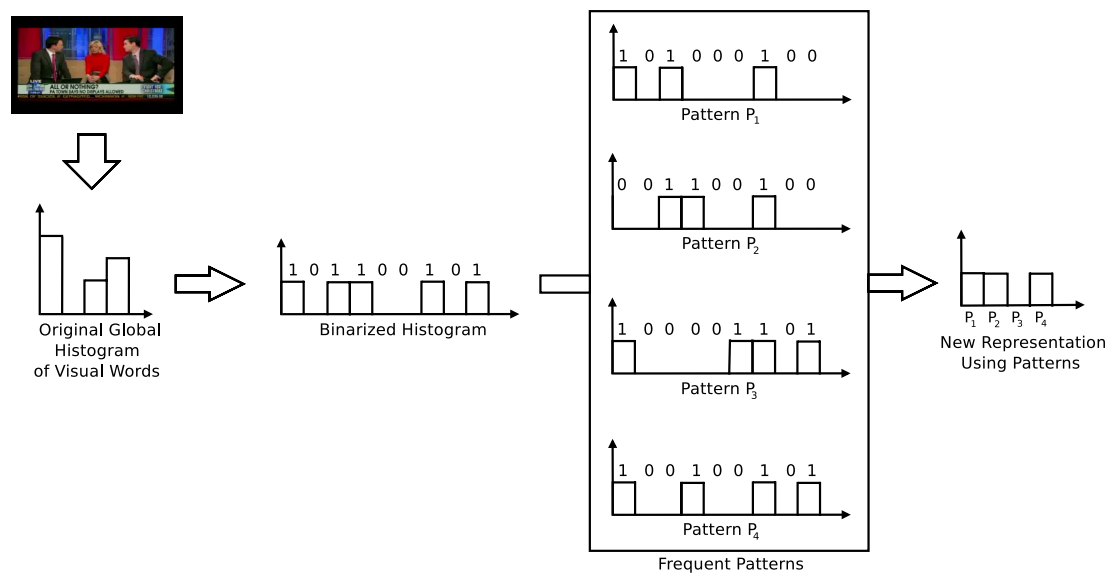


Figure 2.3: Image representation from global histogram (global BoW) using frequent patterns

Figure 2.3 shows the encoding process of an image with frequent patterns starting from a representation with a global histogram of visual words. After converting the histograms for all images into binary formats, a set of frequent patterns is mined from the set of all histograms (this process will be discussed in details later). At the end of the process, each image is encoded with a histogram of patterns. Notice that starting from a global histogram, this final representation is a binary vector (a pattern either appears or not in a given image).

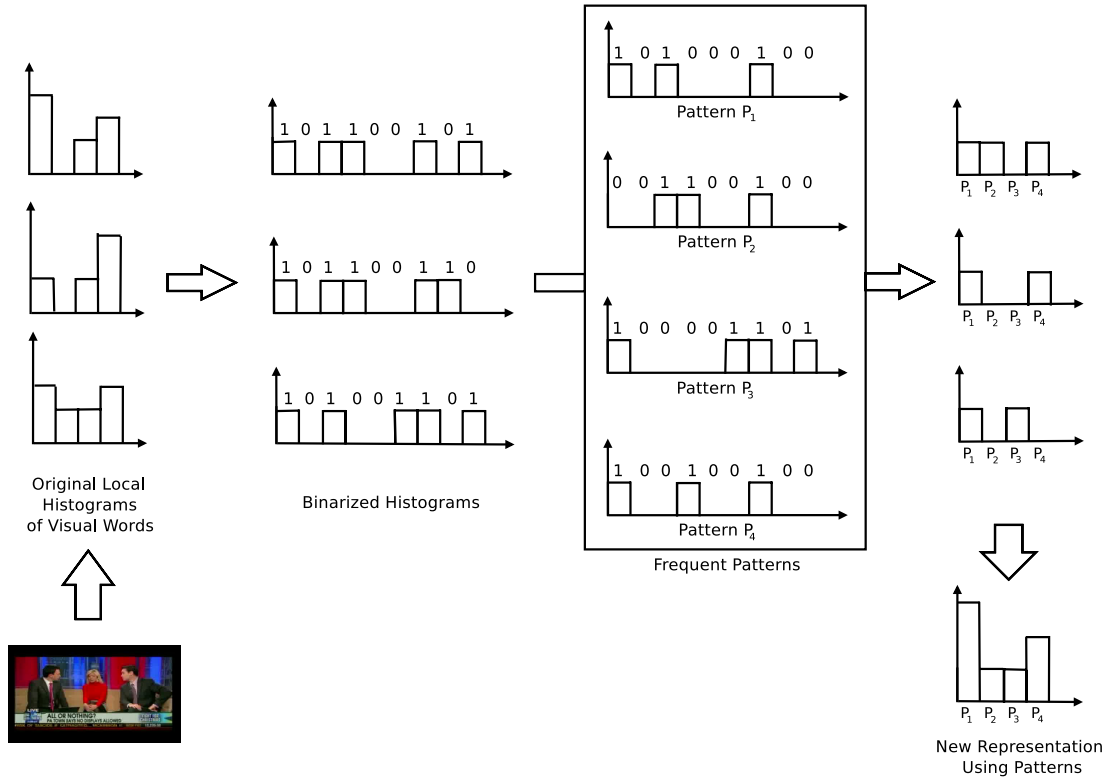


Figure 2.4: Image representation from local histograms (local BoW) using frequent patterns

The process is slightly more complex using local histograms (also called local BoW). In this case, one image consists of multiple local histograms (see Figure 2.4). As in the previous case, each local histogram is also converted into a binary format and all histograms are used during the mining phase. However, in this case, an image is represented as a non binary histogram of frequent patterns since one pattern can happen multiple time in the local histograms representing the original image.

2.2.2 Classification Using Supervised Mined Patterns (FLH Patterns)

When dealing with an image classification problem, we can use the class information (for e.g. the class entropy) associated to the training examples to select the most meaningful patterns. Similarly to what has been done in [30], we use the exhaustive LCM algorithm [91] (see Section 1.2.1.3) to compute closed frequent patterns from the set of local BoW (binarized using the Sparse-B2S technique). We then post-process with a greedy method the patterns extracted using the relevance criteria proposed in

	Dict. Size	SVM with Intersection Kernel		1-NN with L1 Dist. Function	
		Local BoW	FLH	Local BoW	FLH
GRAZ-Person	200	79.4 ± 1.1	83.5 ± 1.8	79.7 ± 3.2	88.0 ± 2.1
GRAZ-Bike	200	76.8 ± 2.5	81.3 ± 2.4	82.0 ± 1.0	80.5 ± 2.8
Flower-CN	100	59.4 ± 1.7	69.5 ± 2.5	50.8 ± 12.2	56.4 ± 14.1

Table 2.1: Accuracy of the SVM and the 1-NN classifiers on the image datasets.

[30]. At the end of the process 6000 patterns are kept to describe the images of the *GRAZ* dataset and 10000 for the *Oxford-Flower17* one. These final patterns are called FLH (*Frequent Local Histograms*).

We evaluate the accuracy of our method using a simple 1-Nearest-Neighbor (1-NN) algorithm with a *L1* distance function over the proposed feature vectors (local histograms and FLH). Other distances were also tried and discussed later on. As it was reported to be successful for those two datasets, we also used a SVM with an intersection kernel to assess the results of the KNN classifier.

Table 2.1 reports the mean accuracy over all images in the 2 datasets using the *L1* distance function. It shows that our simple setting is competitive with the SVM classifier for both datasets (better for *GRAZ* and a little bit worse for the *Oxford-Flower17* dataset). The Local BOW and FLH seems to bring different information since the accuracy is sometimes better with the FLH (88 vs 79.7 for the *GRAZ*-Person dataset) and sometimes worse (82.0 vs 80.5 for the *GRAZ*-Bike dataset). This confirms that the frequent patterns (here FLH) can be interesting features compared to simple bag-of-words if they are carefully chosen. Also the overall good results compared to the SVM classifier confirm the relevance of our features and distance choices. Note that this supervised post-processing step is crucial to select patterns of high quality but it cannot be used in our video tag-correction problem since such class information is not available. In the following, we will thus show unsupervised processing methods to reduce the number of patterns.

2.2.3 Unsupervised Mined Patterns of *Oxford-Flower17*

Inspired by these positive results, we decided to perform more experiments with the *Oxford-Flower17* dataset only using SIFT features since we also plan to use them as a baseline for our experiments on the video datasets. In these experiments, we did not use a SVM classifier nor the FLH patterns to be as close as possible to our video

setting. Instead, we used a 1NN approach with a L1 distance function to evaluate the performance of the closed patterns computed directly by the LCM algorithm from both global and local histograms obtained using different binarization techniques. We started by densely sampling points over all images with overlapping patches of 16x16 pixels, and using a step size of 8 pixels (the distance between the centers of two successive patches is 8 pixels). Standard SIFT descriptors are calculated for each patch. A dictionary of 200 visual words is then constructed.

2.2.3.1 Global Histogram

At global scale, one image is a histogram of 200 visual words. The first step before finding frequent patterns consists in transforming the image histograms into transactions.

Because the value at each bin of each histogram is often significantly larger than 0, using the Simple-Binarization (0 becomes 0, larger than 0 becomes 1) is not recommended. Consequently, we decided to use the B2S and Sparse-B2S binarization approaches detailed in Section 1.2.3. However, we simplified this process by discretizing the histogram values for each visual word into 10 evenly distributed bins to make those values more homogeneous. One image is then transformed from a 200-D vector (a histogram of 200 visual words) into a 2000-D binary vector.

To compute the patterns, we first set the lower support of the LCM algorithm to 10 (patterns that appear in less than 10 images out of 1360 images of *Oxford-Flower17* might not contain enough information to be considered “good”) and the upper support at half the number of images. The intuitive explanation is that if a pattern appears in more than 50% of the images, it is too frequent and thus too general to be a “good” candidate. Using the B2S binarization technique, LCM did not return any frequent patterns after one week running. On the other hand, with the Sparse-B2S binarization technique it returned too many patterns (billions). We try unsuccessfully to lower the upper frequency from 50% to less than 10% but LCM still returned millions of closed frequent patterns after several hours of running. Without any supervised information to post-process the patterns, these numbers were deemed unreasonable to describe an image.

2.2.3.2 Local Histograms

We then created local histograms as described in Section 1.1.1.4 using 5 nearest neighbors (the patch itself and 4 neighbors). On average, each image is represented using

Discretization	No of patterns
Simple-Binarization	133233
Sparse-B2S	141022
B2S	160180

Table 2.2: Number of closed frequent patterns found with the support thresholds [80 – 80000] for the *Oxford-Flower17* dataset using local BoW and different discretization techniques.

about 5000 local histograms (images have different resolutions, thus consist of different numbers of local histograms). These histograms were also converted into transactions (binary formats) using the three techniques explained in Section 1.2.3 The *Oxford-Flower17* dataset of 1360 images is converted into 6507035 transactions.

Using only LCM, we expectedly encountered the same problem as with the global histograms. We arbitrary set the threshold for the upper support to 80000, which means that any selected patterns must appear in less than 80000 transactions (out of about 6 millions). If a pattern appears in more than 80000 transactions and if, on average, it appears 100 times in any images, 800 images would contain that pattern. Since the dataset has 1360 images in total with 80 images per class, a pattern that is available in 800 images is not a good candidate for discriminate features. We also chose to set the lower threshold to 80 for the same reasons. With these two bounds, LCM returns less patterns than the above global case (there were millions), but these numbers are still too big to be effectively dealt with (see Table 2.2): encoding an image with hundred of thousands patterns leads to extremely sparse representations.

2.2.4 Pattern Reduction Techniques

2.2.4.1 PCA (for supervised case)

To overcome this pattern explosion problem, we first decided to use a Principal Component Analysis (PCA) [1]. Our aim is to reduce the number of patterns as well as (hopefully) the redundancy among them. PCA involves a mathematical procedure that converts a number of possibly *correlated* variables into a smaller number of *uncorrelated* variables called *principal components*. The first principal component explains as much of the variability in the data as possible, and each other component explains as much of the remaining variability as possible. The mathematical technique used in PCA is called eigen analysis: it finds eigenvalues and eigenvectors of a square covariance ma-

trix. The eigenvector which has the largest eigenvalue has the same direction as the first principal component. The eigenvector having the second largest eigenvalue determines the direction of the second principal component. This mathematically costly operation prevents us to use the method when the number of patterns is too important. We thus only use it in the supervised case.

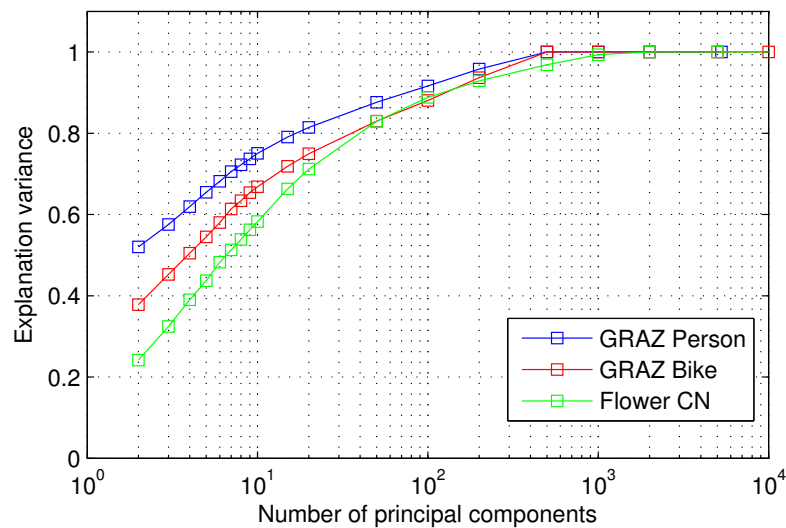


Figure 2.5: Explained Variance according to the number of Principal Components

The original size of the feature vectors is reduced to a given number of Principal Components (PC). Different number of PC were tried: from 2 to 10, 15, 20, 50, 100, 200, 500, 1000, 2000, 5000 when applicable. The number of chosen PC is usually correlated with the explained variance of the dataset which thus can be used to set a reasonable number of PC without using a validation set. When 90% of the variance is explained by the components, one can stop adding new components. Figures 2.5 and 2.6 describe the evolution of the explained variance and of the accuracy with various number of PC. We can see that 100 PC seems a reasonable value to chose. It corresponds approximately to the number of original visual words. In Table 2.1, we chose this particular number of PC value to reduce the number of frequent patterns.

However, using PCA does not improve the final outcome, the only benefit is reducing the computational expense. Thus, if the original sets of frequent patterns are not good enough, applying PCA is an unnecessary step.

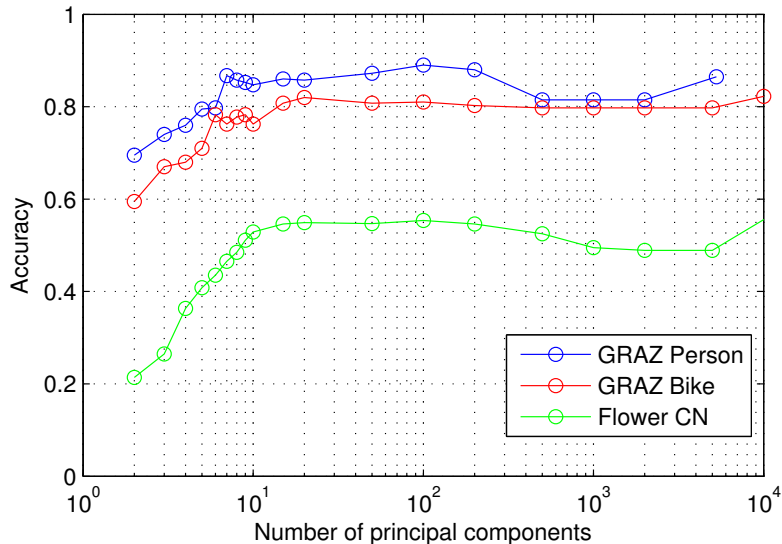


Figure 2.6: Accuracy given by the 1-NN algorithm according to the number of Principal Components

2.2.4.2 MDL-based Reduction (for unsupervised cases)

Global encoding for the *Oxford-Flower17* dataset As discussed in the first chapter, MDL-based algorithms such as SLIM can help to reduce the number of output patterns without supervised information. When used directly on our histogram-based transactional databases, SLIM returns a manageable number of patterns: 2513 patterns with the B2S binarization technique and 5770 with the Sparse-B2S one (see Table 2.3). Even if these values are larger than the original 200-D BOWs, they are much smaller than when using only the LCM algorithm. Although the SLIM algorithm is designed to keep improving the set of patterns over time, we observed that after running 2 hours, the number of output patterns changes at a slow speed. We thus stop SLIM after three days (one day might be already enough) to have enough relevant patterns.

We further tried to improve the quality of the SLIM output by removing all patterns that are too frequent or too infrequent. The lower threshold is arbitrary selected at one-hundredth, and the upper bound at one-third of the images in the dataset. Applying these two bounds, the number of SLIM patterns obtained with the Sparse-B2S binarization process is reduced from 5770 to 1651. With the B2S binarization process, it is reduced from 2513 to 415.

As explained before, we encode each image with a binary vector of patterns and evaluate the quality of our pattern encoding using a L1 based 1-Nearest-Neighbor al-

Visual Features	Size	Accuracy of 1NN with L1
SLIM patterns using SB2S	5770	23.8 ± 11.4
filtered SLIM patterns using SB2S	1651	25.7 ± 12.7
SLIM patterns using B2S	2513	32.8 ± 15.9
filtered SLIM patterns using B2S	415	33.2 ± 15.9
global SIFT-BOW	200	36.4 ± 18.0

Table 2.3: Different features at global scale of *Oxford-Flower17* dataset. The filtered rows correspond to the number of pattern remaining after removing the too frequent or infrequent ones.

Visual Features	Size	Accuracy of 1NN with L1
SLIM patterns using SBin	3206	40.5 ± 17.6
SLIM patterns using SB2S	2734	40.9 ± 18.2
SLIM patterns using B2S	2837	39.5 ± 18.3
Local SIFT-BOW	200	36.5 ± 18.0

Table 2.4: Different features at local scale of *Oxford-Flower17* dataset.

gorithm. Notice that we did not try to obtain the best possible classification results or we would had use a SVN classifier. Instead, we were trying to compare different image encodings so a simple 1NN model is enough for the proof of concept. All results are shown in Table 2.3. This table shows that it is better using SLIM patterns to set a frequency interval for all the patterns: it helps reducing the number of patterns greatly (around 4 times), and it improves the final performance. It also demonstrates that the redundancy between the SLIM patterns is still important. That might explain the fact that, unfortunately, the baseline performance with SIFT-BOW is still better than the SLIM patterns for this global encoding.

Local encoding for the *Oxford-Flower17* dataset Using the SLIM algorithm on the 6 millions local histograms returns about 3000 patterns after 2 days with each binarization technique as shown in table 2.4. Again as explained before, we encode the entire image with a non-binary vector of patterns and evaluate their quality in classification using a 1NN algorithm with a $L1$ distance function. All results are shown in table 2.4. The table shows that, in this case, the results are better using the patterns than using only the SIFT-BOW baseline. We also performed experiments using multiple

K	SBin	Sparse B2S	B2S	SIFT-BOW
1	40.5 ± 17.6	40.1 ± 18.2	39.5 ± 18.3	36.5 ± 18.0
3	36.5 ± 18.1	35.6 ± 18.1	31.8 ± 18.3	32.1 ± 18.6
5	36.4 ± 20.7	36.0 ± 20.2	32.9 ± 17.6	31.3 ± 18.6
10	36.0 ± 21.6	36.8 ± 22.3	32.9 ± 19.9	33.6 ± 20.1
30	36.8 ± 23.8	35.7 ± 24.1	33.9 ± 20.5	33.4 ± 22.6
50	35.4 ± 24.1	34.9 ± 23.6	32.6 ± 20.1	32.6 ± 21.7
100	34.6 ± 24.4	33.2 ± 24.5	32.9 ± 21.0	30.4 ± 21.8

Table 2.5: Accuracy results for the KNN algorithm using different values of K (nearest neighbors) and with a $L1$ distance function at local scale on the *Oxford-Flower17* dataset.

neighbors and show the results in Table 2.5. The results show better performance for this image dataset using only one neighbor. These encouraging results demonstrate the relevance of the patterns obtained with the SLIM algorithm from local histograms as new mid-level features to describe an image.

2.2.5 Distance Functions

We evaluated the quality of our patterns using a KNN algorithm with a $L1$ (Manhattan) distance function. In this section, we question this distance function and compare this distance in classification with the Euclidean distance and the Square root intersection similarity measure between histograms. Various other similarity measures such as the χ^2 , $L_{\frac{1}{2}}$ and $L_{\frac{2}{3}}$ were also tried but the performance were not satisfactory and are not reported here.

The Manhattan distance between two vectors $\mathbf{p} = [p_1, p_2, \dots, p_n]^T$ and $\mathbf{q} = [q_1, q_2, \dots, q_n]^T$ is defined as

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = |q_1 - p_1| + |q_2 - p_2| + \dots + |q_n - p_n| = \sum_{i=1}^n |q_i - p_i|$$

- The Euclidean distance function (or $L2$ distance) between two vectors $\mathbf{p} = [p_1, p_2, \dots, p_n]^T$ and $\mathbf{q} = [q_1, q_2, \dots, q_n]^T$ is defined as

$$d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) = \sqrt{(q_1 - p_1)^2 + (q_2 - p_2)^2 + \dots + (q_n - p_n)^2} = \sqrt{\sum_{i=1}^n (q_i - p_i)^2}$$

Dist. Function	SBin	Sparse B2S	B2S	SIFT-BOW
L1	40.5 ± 17.6	40.1 ± 18.2	39.5 ± 18.3	36.5 ± 18.0
L2	35.9 ± 19.1	37.6 ± 18.6	33.2 ± 18.0	32.2 ± 17.3
Intersection	24.9 ± 20.0	25.5 ± 19.5	29.7 ± 19.7	24.3 ± 13.8

Table 2.6: Different distance functions with 1-NN at local scale of *Oxford-Flower17* dataset.

- The square root intersection between two histograms is a dissimilarity measure. The dissimilarity between two vectors $\mathbf{p} = [p_1, p_2, \dots, p_n]^T$ and $\mathbf{q} = [q_1, q_2, \dots, q_n]^T$ is defined as

$$\begin{aligned}
 d(\mathbf{p}, \mathbf{q}) = d(\mathbf{q}, \mathbf{p}) &= \frac{1}{\sqrt{\min(q_1, p_1)} + \sqrt{\min(q_2, p_2)} + \dots + \sqrt{\min(q_n, p_n)}} \\
 &= \frac{1}{\sum_{i=1}^n \sqrt{\min(q_i, p_i)}}
 \end{aligned}$$

The results are shown in Table 2.6. They show that with a KNN algorithm, the L1 distance seems to be the best strategy to compare vectors of patterns whatever the encoding. This is thus the strategy chosen in the next chapter.

2.3 Step 3: Video Comparison

As explained in Section 1.3.1, there are two popular approaches for comparing videos: either taking the average between all the keyframes or doing a pairwise comparison between all the keyframes of the two videos. In this section, we propose an asymmetrical similarity comparison inspired by the video pairwise comparison techniques to increase the relevance of the video comparison. The first step consists in calculating all the pairwise similarities between all the keyframes of two videos. After that, instead of taking the optimum value of all the pairwise similarity scores, we propose to take the average of all maximum similarities corresponding to one video. In other words, for each keyframe of a video A , we search in all the keyframes of video B for the highest pairwise matching score and we keep this value. Then, we take the average of all the computed values for all the keyframes of the video A to return the similarity score of video A towards video B . If we denote $A(i)$ the i^{th} keyframe of A and $|A|$ the number

of keyframes in A , then

$$\text{sim}(A, B) = \frac{1}{|A|} \sum_i \max_j \text{sim}(A(i), B(j)).$$

The similarity $\text{sim}(A(i), B(j))$ between frames is just the inverse of a distance between the vectors representing the frames. When the two frames are identical, this asymmetrical similarity is set to a maximal value.



Figure 2.7: A dense line is a distance from the one keyframe on the left to the most similar keyframe of the right video. The distance from the left video to the right video is the average of three dense lines.

The relevance of this asymmetrical similarity measure compared to the two baseline proposed in the literature will be assessed in the next chapter together with the tag-propagation process.

2.4 Step 4: Tag Propagation

For each video V , a list of possible-relevant tags is obtained from the k most similar videos. After that, a score function is applied for each tag to estimate the relevance of that tag according to the query video V . This score function depends on the tag frequency (the higher the frequency, the higher the score), the number of tags associated to a video (the higher the number, the smaller the score), and the video similarity (the higher the similarity, the higher the score). Finally, all scores that are larger than a

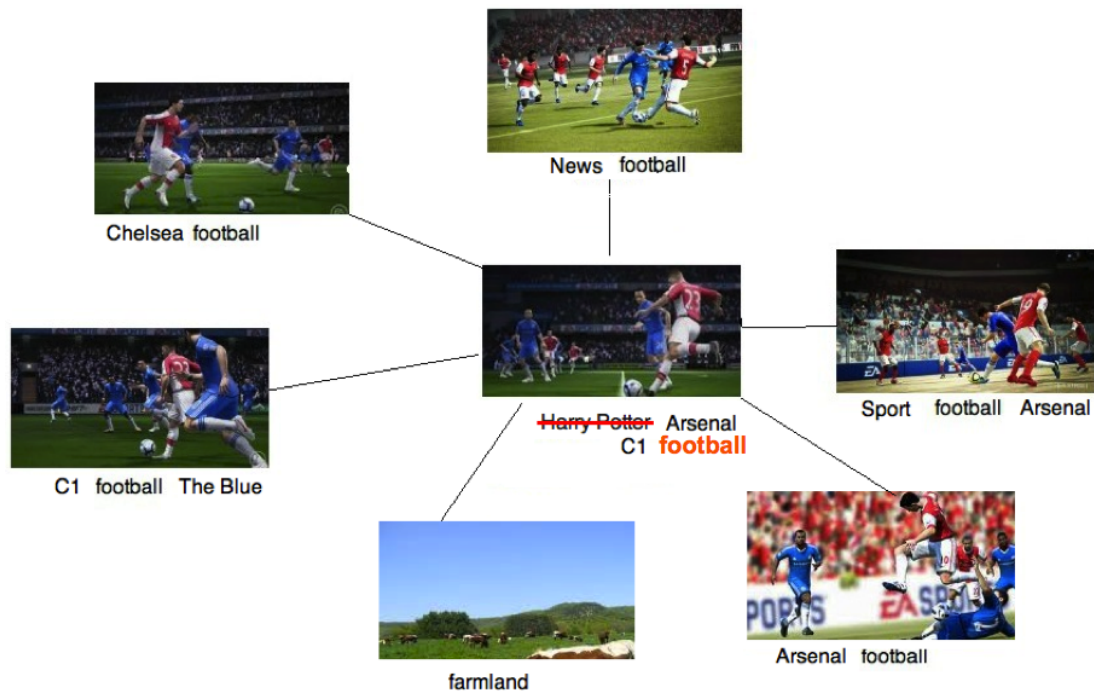


Figure 2.8: An example of tag propagation: tag *Harry Potter* should be removed because it does not appear in the neighborhood, and tag *football* should be propagated because it appears in 5 over 6 neighbors.

predefined threshold will be considered as suitable tags for the video V . Others tags (with smaller scores) will be deleted if they are available in the original tag list.

For instance, given an arbitrary video and an arbitrary number of neighbors set to 30, we first compute the similarities between this videos and all the videos in the dataset and keep only the 30 most similar ones. If most of these 30 neighbor videos contain a tag that is not present in the original video, then the tag should be added in the video (as it might, with high probability, describe the same content as in the video). On the other hand, if that video includes a tag that does not appear in any of the 30 videos, then that tag might have been intentionally mis-added and should be deleted.

To tag a given video $v \in V$, we rely on the tags $t \in T$ of the k most similar videos in its neighborhood. To propagate a given tag t to v , one need to set a threshold on the number of times t should appear in the neighbors. However, given the very different distribution of each tag, we decided to use two comparison statistical tests between the distribution of a tag in the entire dataset and its distribution in the k nearest neighbors. The first one states that the probability of a given tag should be significantly greater

than 0 in the entire dataset to be propagated and the second one states that it should be significantly more present in the neighbors than in the entire dataset. Formally:

- (Global scale) A tag can be propagated if:

$$\hat{p} \geq u_{\frac{\alpha}{2}} \sqrt{\frac{\hat{p}(1-\hat{p})}{N}}$$

where \hat{p} is the proportion of a tag over the whole dataset, N is the total number of videos, $u_{\alpha/2}$ is the $\frac{1-\alpha}{2}$ percentile of a standard normal distribution.

- (Local scale) A tag is propagated if:

$$\hat{p}_1 \geq \hat{p} + u_{\frac{\alpha}{2}} \sqrt{\frac{\hat{p}_1(1-\hat{p}_1)}{k} + \frac{\hat{p}(1-\hat{p})}{N}}$$

where \hat{p}_1 is the proportion of a tag in the k neighbors.

We arbitrarily decide to remove a tag from a video if it is never present in its neighbors. Note that the central limit theorem applies whenever $k \geq 30$ so without any background information on a dataset, this is the value used in most of the experiments.

2.5 Conclusion

In this chapter we presented the framework we use to tackle our video tag-correction problem. To set this framework, we made a number of hypotheses that were validated. In particular, we showed that frequent patterns are good mid-level features to describe images (and hopefully frames of videos). We also show that a simple vector comparison using a L1 distance function can give good results in image classification and thus hopefully to compute a visual similarity between two videos. We have proposed an asymmetrical similarity measure and two tag propagation criteria. These two last contributions will be evaluated in the next chapter after defining our experimental set up on videos.

Chapter 3

Datasets and Tag Propagation Experiment Designs

Our framework is described in details in the previous chapter with some justifications about the parameters we use. In this chapter, we apply this framework to three different datasets with different characteristics. Note that, as explained in the first chapter, there are no dataset with a ground truth available for tag propagation problems (contrarily to classification problems) so, we had to create them manually and design our own evaluation procedure. This chapter presents our attempts in chronological order. The first dataset is a small dataset with 51 videos with few tags. The first idea was to assess if videos sharing the same topic would be visually close to each other and if the tag propagation would be beneficial in this case. However the results were not entirely satisfactory in both cases probably due to the small size of the dataset and the subjectiveness of the tags that were selected. We thus introduced a second 182-videos synthetic dataset created from cutting parts of 7 original videos (with one explicit tag per part) to ensure a strong correlation between the tags and the visual content of the videos. The experiments with this second dataset confirmed the validity of our framework but also showed that global SIFT-BoW features alone do not yield the best results as also confirmed in the previous chapter. The experiments on these first two datasets were carried out before trying the local SIFT-BoW features, therefore, all the results are reported using global features for these datasets. The third dataset is a real and significantly larger dataset with 668 videos and 150 tags. Many strategies with different settings were applied to evaluate our framework on this dataset.

3.1 A First Real Dataset (51 videos)

In the following experiments we use a 51-videos dataset taken from a benchmark dataset including 80,031 of the most viewed videos for every month from Dec. 2008 to Feb. 2009 on YouTube [12]. We kept this dataset reasonably small to manually assess the relevance of the original tags and of the corrected ones for each video. We are interested in evaluating the interest of the frequent pattern-based features compared to the BOW-based features.

3.1.1 Dataset Description

Each of the 51 videos in our dataset is decomposed into key-frames based on [115]. An example of the keyframes for some of the videos are given in Figure 3.1. There are about 18 shots per video and 1.5 key-frames per shot, i.e. about 27 key-frames for one video. A visual vocabulary of 1000 words built from SIFT features is used to describe the video. A video is thus represented by a matrix which contains for all key-frames of the video the visual word histogram which describe the frame. The dataset contains 1895 keyframes in total.

The 51 videos were chosen such that they belong to 4 topics to ensure that this dataset contains pairs of similar videos and pairs of dissimilar videos. The topics are *anime* (10 videos), *football* (17 videos), *news* (12 videos) and *racing* (12 videos). We chose the top-35 most frequent tags from all tags that were used for these videos in the original YouTube dataset. The 51 videos were then re-tagged using only these 35 tags to create a ground truth. The final tags associated to the videos are shown in Table 3.1.

3.1.2 Encoding Procedure

We first tried to find frequent *sequential* patterns in these videos. Note that in our cases, the original sequences are itemsets which limit the number of algorithms we can use to tackle the problem. We used the SPAM algorithm presented in the first chapter. Even if SPAM is able to produce frequent sequential patterns, it has several drawbacks: the software limits the number of keyframes in each video to 64 and, it cannot generate closed patterns (thus the number of sequence patterns created by SPAM is extremely large). When applying it to this small dataset with a support threshold of 30% (which means that the frequent sequences should appear in at least 15 videos which is roughly equal to the size of each class), SPAM generates 1 242 660 frequent sequence patterns.

Video IDs	Tags
3107012	anime one sword people lying down outside
3107013	anime sword people discussing fighting blue outside
3107085	anime people fighting discussing nature tree green blue outside inside
3188073	anime people discussing city outside inside
3188083	anime people dark discussing inside outside nature
3286509	anime one people fighting tree blue green outside nature sword discussing
3107846	football people playing running green red penalty area goal outside
3107915	football people playing running ball green black red penalty area outside
3139445	football people playing walking goal green white black outside
3149093	football people playing walking talking running green red blue outside lying down
3282258	football video game people playing walking red green outside
3283038	football people green outside
3283053	football people lying down penalty area outside
3283054	football people playing green red penalty area outside
3283068	football people running green blue penalty area outside
3249151	news people discussing talking city black inside outside
3249191	news red eye people discussing fox inside
3249309	news people talking discussing city red eye inside
3249636	news presenter inside outside
3281891	news people presenter talking blue inside
3282532	news fox presenter talking red inside outside
3282535	news fox city people presenter talking car inside outside
3287317	news people discussing inside
3147888	racing race car outside
3195813	racing formula one motor outside
3249014	racing formula one presenter talking outside inside
3280533	racing car race motor
3286194	racing race formula one

Table 3.1: List of tags corresponding to some of the 51 videos.



Some keyframes of an anime video.



Some keyframes of a football video.



Some keyframes of a racing video.

Figure 3.1: Several example keyframes of the 51-videos dataset.

We also tried the CloSpan algorithm which eliminates the two above drawbacks of SPAM. With the same threshold, CloSpan extracts 613 150 closed frequent sequences. This number of patterns is still too large to encode videos without any post-processing step we just decided to abandon this strategy.

We then computed frequent itemsets from the global histograms associated to the 1895 keyframes using the LCM and the KRIMP algorithms as explained in the previous chapter. In both cases, the support threshold is set to 80 frames (over 1895 frames, it means about 4%). The size and the number of returned patterns for both algorithms are shown in Figure 3.2.

First we focused on long patterns: LCM returns 4682 closed frequent patterns of a size larger or equal to 8 items. To estimate the relevance of these patterns, we compute the number of visual words in the visual vocabulary that is included in a set of patterns. Surprisingly, this substantial amount of patterns (equal to five times the number of original visual words) contained only 54 visual words (from the 1000 possible ones). This seems to indicate that closed frequent patterns are too specific and using these patterns to represent the data might not be more useful than using the original visual words. With KRIMP, with a size threshold of 4 items, we obtained only 82

Size of patterns (no of items)	Number of closed frequent patterns (from LCM)	Number of KRIMP patterns
1-item	...	1 000
2-item	...	4 266
3-item	...	7 257
4-item	1 431 976	71
5-item	146 649	6
6-item	59 313	2
7-item	20 139	2
8-item	4 114	0
9-item	457	0
10-item	17	1

Table 3.2: Number of frequent patterns from LCM and KRIMP

patterns which covered 110 visual words. This seemed much already better than in the previous case. To increase the coverage, we could have lowered the size threshold to, e.g, 3 ending up with a total number of 7339 patterns which would have covered 911 visual words.

3.1.3 Evaluation Procedure

To evaluate the relevance of our itemsets, we recoded the video using the 82 KRIMP patterns. A video is represented by a histogram of KRIMP patterns where each histogram bin gives the number of keyframes containing a given pattern (as explained in the previous chapter, each pattern is only counted once per frame). To assess if videos sharing the same topic would be visually close to each other, we first did not use the tag and measured the distances between all possible pairs of the 51 videos. We calculated the distance within a class and outside a class. Table 3.3 shows the final result. The values in the diagonal are the within-class distances and are expected to be the smallest value in the table (videos within the same class should be closer).

These results are good for the *anime* class and quite good for the *racing* class, but they are not for the *news* and for the *football* classes. This phenomenon can be explained by the big difference between the *anime* videos (a kind of cartoon) and the rest of the videos (which are real life videos). The *racing* score is good probably because this class contains some video games, which should be also be different from the real

Classes	Anime	Football	News	Racing
Anime	0.15	0.18	0.30	0.15
Football	0.18	0.21	0.32	0.18
News	0.30	0.32	0.40	0.30
Racing	0.15	0.18	0.30	0.15
Mean over all dataset	0.24			

Table 3.3: Mean of differences between videos of each class. In ideal case, **bold numbers** are the minimum values over column and row.

life scenes.

We then evaluate our tag propagation procedure by including the tags for each video as shown in Figure 3.1. To do so, we consider a video dataset as a triplet (V, T, tag) where V is the set of videos $V = \{v_1, \dots, v_n\}$, the set of possible tags is $T = \{t_1, \dots, t_m\}$ and tag is a relation on $V \times T$ such that $tag(v, t)$ is true if and only if the video v has the tag t .

Our evaluation procedure is then:

- add some noise on the tags, i.e, we choose a noise proportion $0 < p < 1$ and we compute a noisy tag function tag_{noisy} such that, for each $t \in T$ and $v \in V$, with probability p we have: $tag_{noisy}(v, t) = \neg tag(v, t)$ (it means that we flip the value of a given tag with probability p);
- apply our tag correction technique, the output of the tag correction step is tag_{corr} ;
- compute the proportion of the incorrect tags after the correction step as:

$$err(tag, tag_{corr}) = \frac{|\{(v, t) \in V \times T \mid tag(v, t) \neq tag_{corr}(v, t)\}|}{(|V| \cdot |T|)}$$

The ideal case is of course when $err(tag, tag_{corr}) = 0$. Notice that $err(tag, tag_{noisy}) \approx p$. This means that as soon as $err(tag, tag_{corr}) < p$, there is less incorrect tags after the tag propagation step than before on the noisy set. Therefore it means that our algorithm has decreased the number of incorrect tags. On the contrary, if $err(tag, tag_{corr}) > p$, it means that our algorithm has actually increased the number of incorrect tags. In the following figures, we plot the error $err(tag, tag_{corr})$ against the value of p . If this curve is below the diagonal line, it means that our algorithm has decreased the number of incorrect tags.

3.1.4 Results

The results on the first 51 videos dataset are presented in Figure 3.2. They are averaged over 100 runs for each noise level.

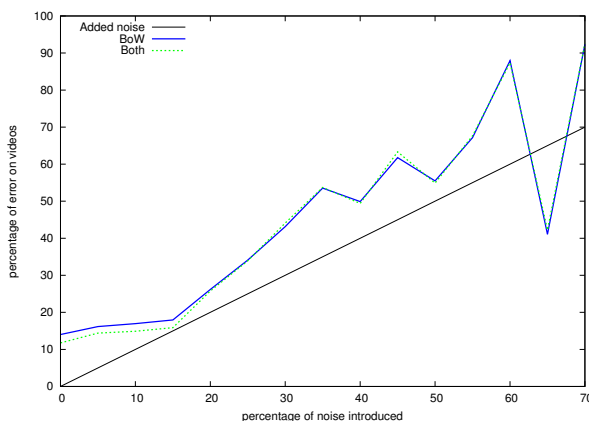


Figure 3.2: Result of our tag correction algorithm on the small real video dataset. Our algorithm uses only bag of word features (solid line) or histograms of both words and frequent patterns (dotted line).

Our algorithm does not improve the original tagging for almost all the noise levels: the number of incorrect tags is higher after our correction algorithm than before. These errors can be the result of the correction algorithm or the fact that the computed distance between videos does not reflect the real similarity of the videos. The number of videos that we use is quite small. In a dataset of millions of videos (as the complete YouTube repository), the k nearest neighbors of a given video should be much more similar than in our small dataset (and thus have very similar tags).

Another problem lies in the tags themselves: our algorithm uses the visual similarity between videos to correct the tags. Thus, it can be efficient only on tags that are correlated with the visual content. For instance, tags such as “forest” or “sea” can be corrected because they are correlated with the visual content (green). A tag which describes a higher level concept (such as the name of a person in a video or the date when the video was shot) is probably not correlated enough with the visual content.

3.2 Synthetic 182 Videos Dataset

Since the results on the above small dataset were not satisfactory because of its too limited size and because of the quality of the tags, we proposed another synthetic dataset of 182 videos build from 7 very different videos from the previous dataset.



Some keyframes of a synthetic video with 3 tags: *cat, football, news*



Some keyframes of a synthetic video with 2 tags: *news, crowd*



Some keyframes of a synthetic video with 3 tags: *moon, cat, anime*

Figure 3.3: Several examples of keyframes of the synthetic 182 video dataset along with the tags associated with the video the keyframe belongs to.

3.2.1 Dataset Design

To better control the similarity or dissimilarity of the videos and the tags, we build a synthetic dataset made of 182 videos. To construct this dataset, we first select 7 real videos A, B, \dots, G belonging respectively to the classes “moon”, “crowd”, “anime”, “racing”, “cat”, “news” and “football”. Note that if each synthetic video was given one of the tags, the total number of tags would be $182 * 7 = 1274$. This means that when adding 5% of noise in the dataset, $63 (\approx \frac{1274 * 5}{100})$ tag values would be flipped (added or removed) in the dataset.

Then, to build each synthetic video we:

- choose randomly between 2 and 4 of the real videos;
- choose randomly frames from each of the chosen real videos. The set of frames thus obtained is concatenated to become the new synthetic video;
- tag this synthetic video with A if it contains frames from video A, with B if

it contains frames from video B and so on. Each synthetic video has therefore between 2 and 4 tags out of the 7 possible tags.

Examples of some frames used for this dataset are shown in Figure 3.3. By construction, if two synthetic videos share for instance the tag A, it means that they both contain some frames (possibly the same ones) extracted from the real video A. Thus, we ensure that they contain similar frames. Moreover, by construction, each tag is correlated with the visual content of the video. We therefore avoid the last problem encountered with the previous small dataset.

3.2.2 Evaluation Procedure

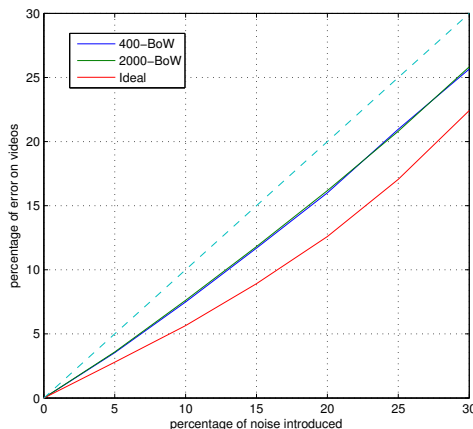


Figure 3.4: Percentage of wrong tags in the videos after one propagation step according to the percentage of noise introduced. The blue and green lines give the results of our method which compare histograms of SIFT-BOW using a L1 distance function. The red line gives the result of the propagation step starting from an ideal visual comparison.

The design of this dataset allows us to build a perfect measure to assess the visual similarities between the videos. Suppose that a first video contains 3 tags A,B and C and a second video contains 4 tags A,D,E and F. The similarity between these two videos is defined as the number of common tags divided by the largest number of tags in the two videos i.e. $similarity(ABC,ADEF) = \frac{|A|}{|ADEF|} = 1/4$. Intuitively, it means that ABC and ADEF have at least 1/4 of their keyframes that are similar (here with tag A). The distance between the two videos is the inverse of this similarity. This “ideal” distance function will be used as our baseline in the following experiments when computing the

K NN matrix. Apart from this baseline, we use the same evaluation procedure as in Section 3.1.3.

3.2.3 Results

Figure 3.4 shows the tag propagation performance using the SIFT-BoW features and the L1 distance function between the histograms of patterns. In Figure 3.4, the lower the lines are, the more labels are corrected during the tag propagation process. We use two vocabulary size for the BoW: 400 and 2000 but obtain almost the same results (blue and green lines). The ideal case line is lower than our L1 distance function using the SIFT-BOW features. This means that these features or this distance function can still be improved to obtain a better tag propagation system.

Figure 3.5 shows that the results for this dataset are however a lot better than with the simpler dataset presented in the previous section. For a noise level between 0 and 30%, we see on Figure 3.5 that the proportion of incorrect tags significantly decreases. For instance, at a noise level of 20%, the error proportion after the tag correction is around 16%. This means that the algorithm has removed about one quarter of the errors introduced by the noise. Note that for a higher noise level, the number of incorrect tags is too important to hope improving the results by tag propagation.

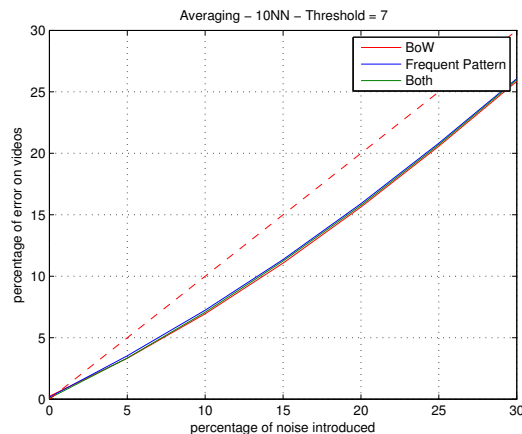


Figure 3.5: Percentage of wrong tags in the videos after one propagation step according to the percentage of noise introduced computed on a synthetic video dataset. Our algorithm uses only SIFT-BOW features, frequent patterns or both.

3.3 A second real dataset (668 videos)

3.3.1 Building Dataset

To illustrate our method, we pre-processed a Youtube dataset [12] with more than 10,000 videos already decomposed into shots and keyframes. There are about 18 shots per video and 1.5 keyframes per shot, i.e. about 27 keyframes for one video. We first decided to focus on videos with some common tags to obtain an interesting sample of the original dataset. For that, we focused on the 500 most frequent tags in the original set. Table 3.4 shows some tags from this list from various locations. We can notice the tags frequency highly varies at the beginning of the list and becomes more stable when reaching a frequency under 100. Besides, the most frequent tags are mainly meaningless (e.g. “the”, “of”, “full”, “lol” ...). We removed from this list of 500 frequent tags the articles, pronouns, prepositions, words with less than two letters and also the 100 most common tags in the remainder list (those, such as the word “video” who were considered too frequent to be informative). Finally, top-150 tags were selected. This 150-tags are meaningful and with a frequency still large enough to be able to select similar videos in the original corpus. We then kept the videos that contain at least 5 of those 150 tags and more than 1 keyframes. This process gave us a corpus of 668 videos with at least 5 meaningful tags per video. The global distribution of the 150 tags is given in Figure 3.6 (top).

3.3.2 Evaluation Procedure

To assess the accuracy of our system we need to manually check the 668 videos to remove all noisy (incorrect and incomplete) tags. Since this is not doable in practice, we randomly chose 50 videos from the original set, and tagged them by hand using the 150 pre-processed tags to obtain a reliable ground truth. We ran our tag propagation method on the 668 videos and reported the accuracy results for these particular 50 videos. This accuracy is measured in terms of “percentage of good corrections” (PGC). Let $T_{add,correct}$ be the correctly added tags out of $T_{add,total}$ added tags and $T_{remove,correct}$ be the correctly removed tags out of $T_{remove,total}$ removed tags.

$$PGC = \frac{T_{add,correct} + T_{remove,correct}}{T_{add,total} + T_{remove,total}}$$

If PGC is larger than 0.5, our system improves the tags in the videos. Note that since most existing tag correction systems use some supervised information, we do not

top-10		“top” 111 → 120		“top” 241 → 250	
frequency	tag	frequency	tag	frequency	tag
1288	the	140	pet	85	kristen
1072	video	139	james	84	nick
927	of	138	york	84	mobile
912	funny	138	taylor	84	fun
816	2009	138	part	84	family
730	new	137	white	84	cosmetics
508	to	137	full	84	channel
486	music	137	episode	83	young
464	world	136	your	83	ufo
418	in	135	lol	83	piers

Table 3.4: Examples of the top-500 most frequent tags present in a 10000 video-dataset extracted from the Youtube dataset.

compare our system to them. The following experiments stand for a proof of concept of our system.

The number of neighbors considered for the tag propagation and the statistical tests are directly responsible for the propagation (or the removal) of a tag. To evaluate our choices experimentally, we selected a frequent tag (“amanda”) in our video and 4 videos that should not be tagged with this particular tag¹. Fig. 3.8 shows the number of nearest neighbor videos that contain the tag “amanda” for the 4 different videos (plain lines). It also shows how many nearest neighbor videos should contain the tag “amanda” to trigger the propagation step (dashed line). Since the tag should not be propagated to these videos, the plain lines should stay below their corresponding dashed line. This is correct for all 4 out of 4 videos for 30 nearest neighbors. It is not correct anymore when increasing the number of neighbors. This means that for this particular tag, increasing the number of neighbors will actually degrade the propagating system. Other similar experiments tend to confirm the relevance of choosing 30 neighbors.

Figure 3.6 (bottom) shows the distribution of the tags around the neighbors of a each video in the dataset. From both histograms in Figure 3.6, we can conclude that there is no direct correlation between the global and the local distributions of the tags which justifies the use of the propagation criteria presented in Section 2.4 which takes

¹Amanda Holden is a TV show presenter that is often present in our video samples.

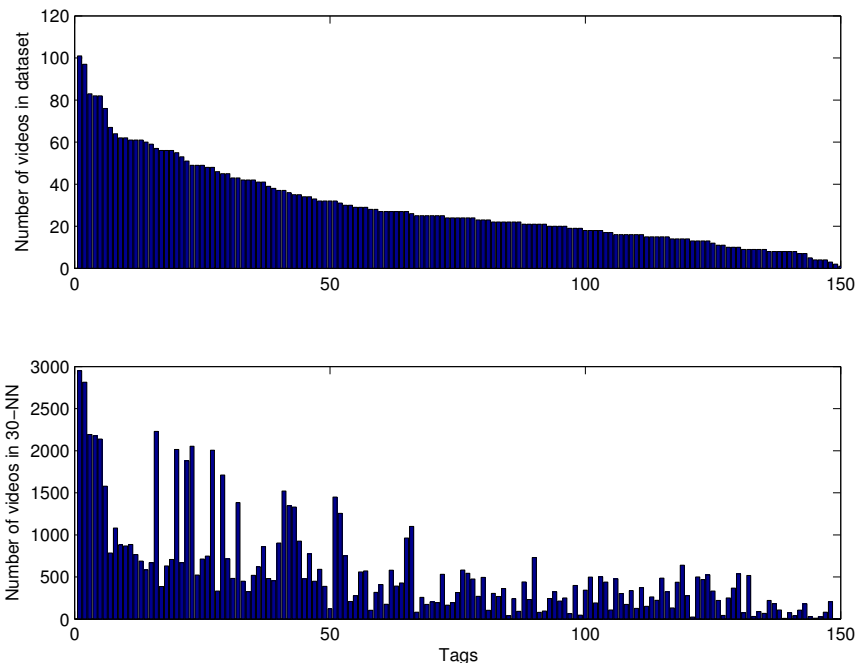


Figure 3.6: Distribution of the 150 kept tags over the 668 video dataset (top), and, for each video in the dataset, over all its 30 nearest neighbors videos (bottom). The nearest neighbors are computed using the SIFT-BOW baseline and the L1 distance function.

both the local and global information into account.

3.3.3 Experiment Settings

As shown in the experiments using the image datasets (see Section 2.2.4.2), the global visual features (one histogram per image) do not provide enough information to obtain discriminant patterns after the mining phase. Therefore, we only implemented the local visual features (as explained in Section 1.1.1.4) with this real video dataset. Each keyframe is represented as multiple local histograms, and we constructed a dictionary of 1000 visual words. We choose this value which is significantly larger than the 200 visual words vocabulary that was used for the image datasets because of the larger number of keyframes (around 14000).

From this 668 videos and from the local SIFT-BOW feature vectors computed from the keyframes we created a dataset of about 6,000,000 local histograms. To reduce the computational load, we use the Simple Binarization strategy introduced in Section 1.2.3.

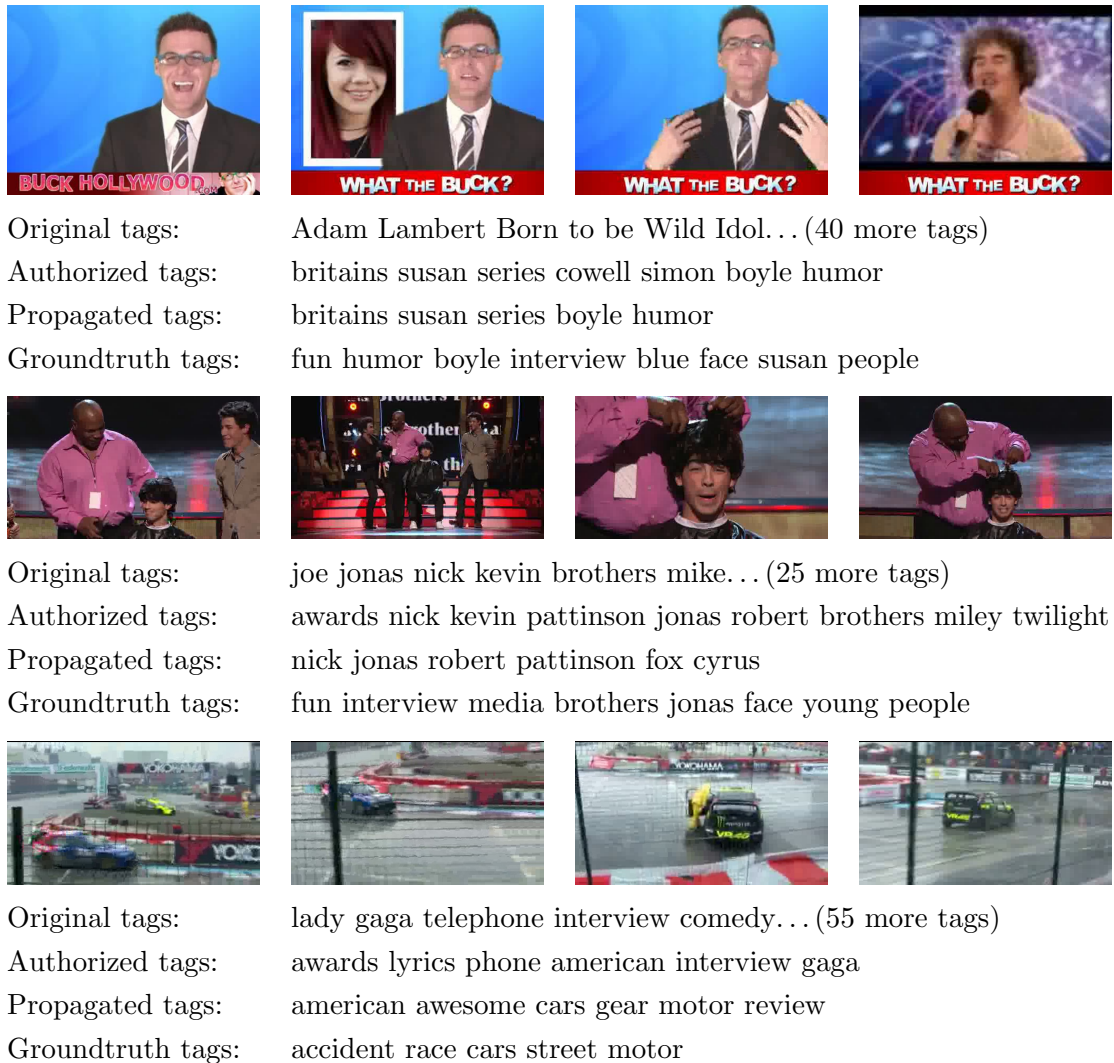


Figure 3.7: Several examples of keyframes of the real 668 videos dataset along with parts of the original tags and the complete pre-processed tags associated with the video the keyframe belongs to. The result of the tag propagation using our system and the tag ground truth is also given for these keyframes.

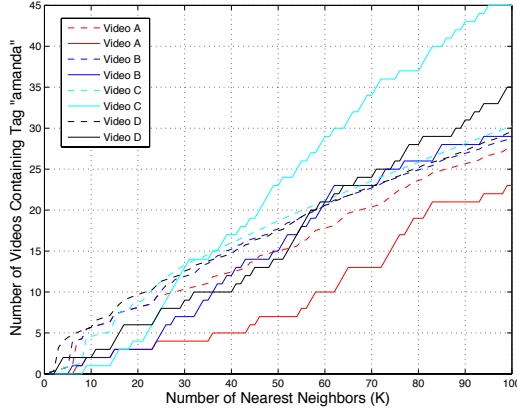


Figure 3.8: Number of nearest neighbor videos that contain (plain line) or should contain the tag “amanda” to trigger the propagation step (dashed line) according to the number of nearest neighbors for 4 different videos. A video is described using an average of the histograms representing all its keyframes. The keyframes are described using one global SIFT bag-of-words. The videos are compared using a L1 distance function.

Consequently, the input data used for the data mining algorithms consists of 6,000,000 transactions where each transaction is a binary 1000-D vector. Different strategies were implemented to try to extract the most valuable frequent patterns and shown in Figure 3.9. In fact, a fifth strategy which consists in applying the SLIM algorithm directly from the 6 millions transactions (running for two weeks) was tried but the final results presented in Figure 3.10 were not better than the baseline. This strategy was thus abandoned.

As in Figure 3.9, we separate all experiments into two parts: one where do not use the tag information and another where we do.

3.3.3.1 Finding Patterns Without Tag Information

We run the LCM algorithm using the protocol described in Chapter 2 without any supervised information. The minimum frequency threshold used is 500 which is the lowest value we can set to be able to post-process the results in a reasonable amount of time and within a reasonable amount of memory (lower than 20 GB).

We compute both closed and maximal frequent patterns. The LCM algorithm returns about 700,000 closed and 550,000 maximal frequent patterns (these numbers are rounded for the sake of readability). These values are, not surprisingly, too large

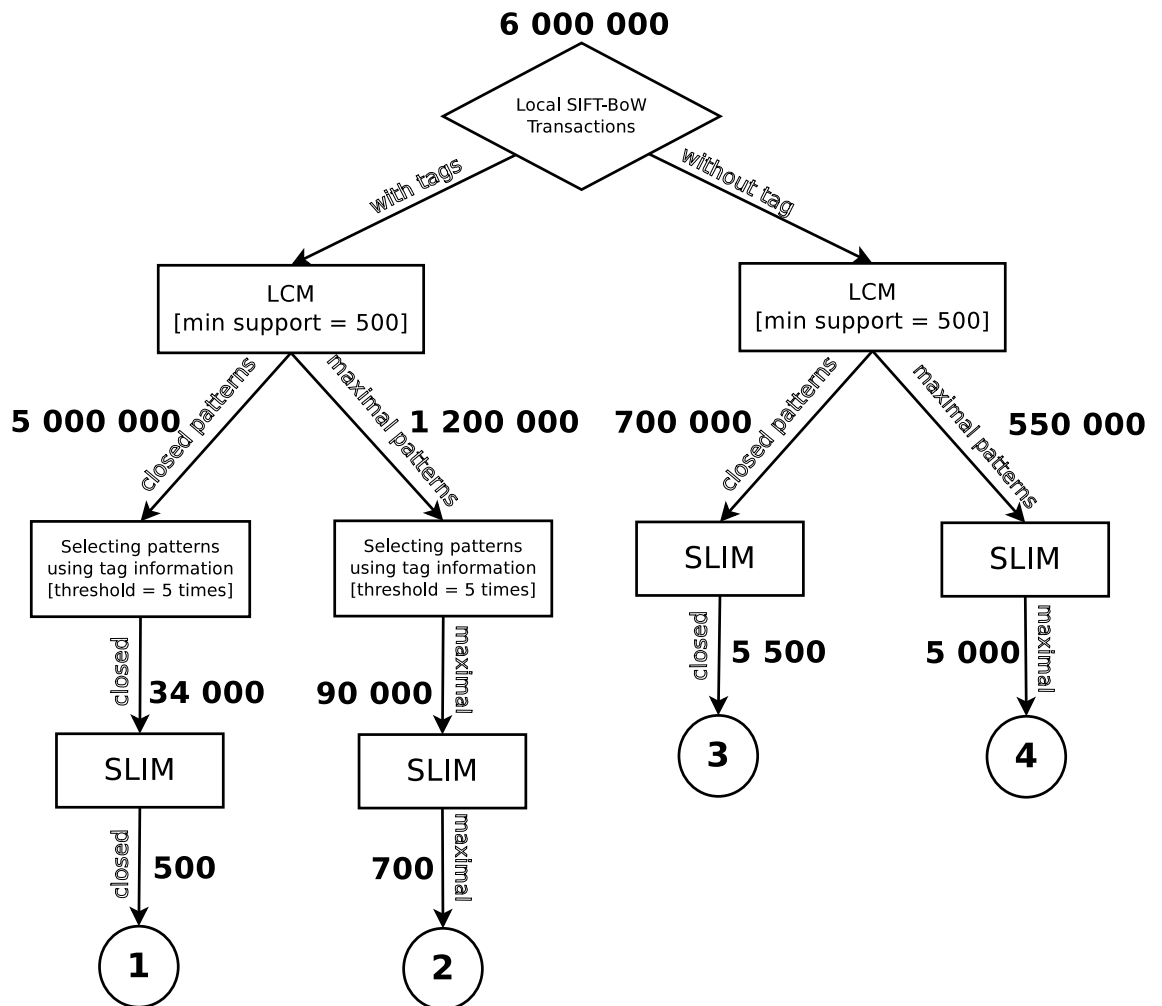


Figure 3.9: Four different strategies to obtain a discriminant set of frequent patterns. The two first ones use first the LCM algorithm on the dataset and compute the maximal or the closed patterns. Then the patterns are post-processed to keep the emerging ones (using the tag information) and further post-processed using the SLIM algorithm. In the third and fourth strategies, the tag information is not used.

to be used directly for image encoding and thus, a post-processing step is needed to significantly reduce them.

Without any supervised information, we rely on SLIM MDL-based optimization criterion to find a reduced interesting set of patterns from the set of frequent patterns output by LCM. This means that the output from SLIM is a set of higher order patterns not a set of patterns build directly from the visual words. There is no straightforward

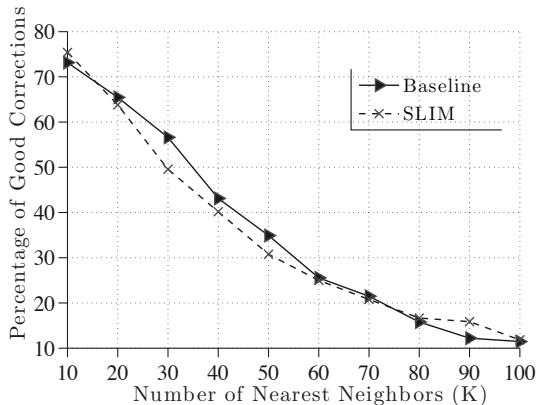


Figure 3.10: Percentage of good corrections according to the number of neighbors after one propagation step using a SIFT-BOW video encoding (baseline) and an encoding based on SLIM patterns derived directly from SIFT-BOW. The dataset is the 50 ground-truth sample taken from the 668 real video preprocessed dataset.

connections between these “new” patterns and the original transactions. However, since the “new” patterns are the set that compresses the classic frequent patterns well, and since the classic frequent patterns are all extracted from the original transactions, the “new” patterns might contain some useful information of the original transactions.

Applying SLIM greatly reduce the number of patterns of two order of magnitude: from 700,000 to 5,500 for the closed and from 550,000 to 5,000 for the maximal ones. These values can be used directly to re-encode our videos and then perform the tag propagation.

3.3.3.2 Finding Patterns With Tag Information

Even if the tags (that don’t belong to the 50 ground truth videos) cannot be entirely trusted, they are still one important source of information. In fact, they are used in the tag propagation step to transfer tags to videos. Thus they could also be used as a measurement to choose the good patterns.

In addition to the 1000-D binary input vector, we add another 150 dimensions for the tag information. Each position in the last 150 dimensions symbolize one tag, and its value is 1 if that video contains the tag and 0 if otherwise. LCM is used first to find all closed and maximal frequent patterns from the set of 6 millions larger transactions. It returns a very large number of patterns: 5 millions closed and 1.2 millions maximal frequent patterns. Note that this is not surprising since each video is described by

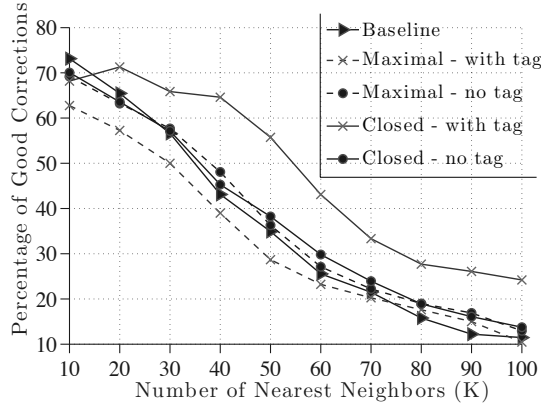


Figure 3.11: Percentage of good corrections according to the number of neighbors in 50 videos represented with SIFT-BOW (baseline) and frequent patterns obtained LCM(closed and max)+SLIM and LCM(closed and max)+ SLIM+ tag_information.

around 10,000 ($6M/668$) local histograms so each tag belonging to a video will appear in at least 10,000 transactions.

We then use the tag information integrated into these classic frequent patterns to keep only the discriminant patterns. We use a method very similar to the one used for emerging patterns in a classification problem [23]. We call $Freq_{in}$ the frequency of a pattern FP only considering the transactions containing the tag T , and $Freq_{out}$ the frequency of the pattern FP in all transactions that don't contain the tag T . We define a pattern FP as *relevant* for a certain tag T if $Freq_{in}$ is 5 times larger than $Freq_{out}$.

This simple strategy reduces the number of patterns significantly: there remains 34,000 closed and 90,000 maximal patterns. We then remove the tag information out of all patterns and apply the SLIM algorithm to post-process them further. SLIM returns 500 closed and 700 maximal patterns. Finally, we perform the tag propagation using these patterns as the main visual features for representing our videos.

3.3.4 Results

Figure 3.11 shows the results for the 5 experimental settings described above. Note that using less than 30 neighbors questions our statistical tests. On the contrary, using 100 videos out of 668 clearly introduces a lot of noise. The best results are obtained using the combination of LCM (closed), SLIM and the tag information (output 1 of Fig. 3.9). In this case for 30 neighbors, the system is able to produce around 65% of good corrections which means that 54 correct tags were added or deleted in the process

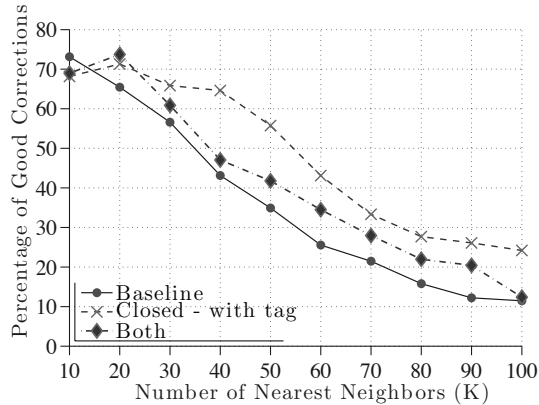


Figure 3.12: Percentage of good corrections according to the number of neighbors for a video dataset represented with 3 different features: SIFT-BOW (baseline) and frequent patterns obtained with LCM(closed) + SLIM + tag_info, and the concatenation of both feature vectors.

(28 were wrong propagations). For the same setting, the baseline only allows us to produce 57% good corrections. Figure 3.12 and 3.13 focus on the baseline and the best case. It also shows the results when both vectors are concatenated. The concatenation produces worse results than the best case which means that the information given by the baseline is not complementary to the information given by the frequent patterns.

Figure 3.14 shows a comparison between our proposed asymmetrical similarity measure and the basic method which consists in simply averaging for one video all the keyframe features and computing an histogram intersection between the feature vectors representing two videos. Our proposed method does give better results but the difference is not significant using 30 neighbors. The frame average method may thus be preferred for efficiency reasons.

3.4 Conclusion

We have presented the design of 3 datasets and the experiments conducted with the system described in the previous chapter to automatically correct and complete tags in a video dataset. The experiments show that the size of the dataset should be important enough for the system to be accurate. The experiments also show that the whole system greatly depends on the choice of the original features and on the encoding of the videos. However, the experiments on the last bigger dataset showed that relying on good visual

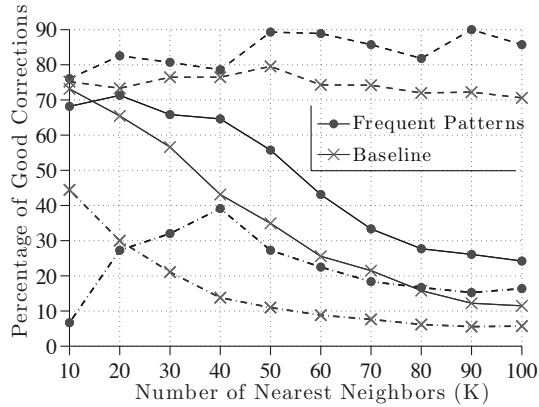


Figure 3.13: Percentage of good corrections when counting only the addition (dashed line with +), the deletion (dashed line with \circ) and both according to the number of neighbors for a SIFT-BOW (baseline), LCM(closed) + SLIM + tag_info and the concatenation of both feature vectors.

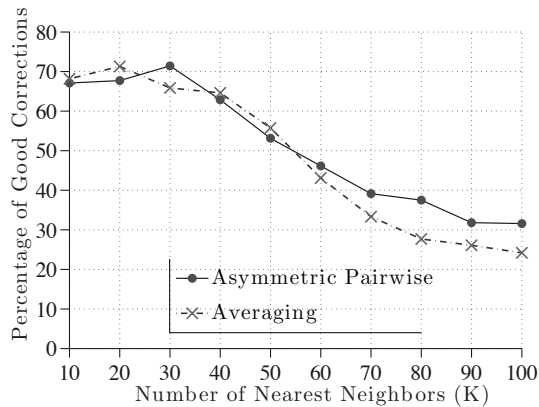


Figure 3.14: Percentage of good corrections to the number of neighbors for a video dataset represented by frequent patterns obtained with LCM(closed) + SLIM + tag_information using the asymmetrical similarity measure and simply averaging the keyframes features and computing an histogram intersection.

features and some partial information about the tags could lead to interesting results and an effective end-to-end system.

Chapter 4

Conclusions and Future Works

This dissertation has presented new visual mining-based features, and evaluated their effectiveness when integrated in an auto tagging framework for video datasets. We conclude our work by summarizing our main contributions and drawing conclusions in Section 4.1. We then suggest several interesting directions for future research in Section 4.2.

4.1 Contributions & Conclusions

A simple but effective framework. Our first contribution is a general framework for unsupervised tag propagation in video datasets. This framework is modular and contains four independent blocks. Different visual features, distance functions, video comparison methods and propagation techniques can be used.

Videos to transactions conversion. Video data are specific in the sense that their structure is complex and they can be described by a large number of different visual features. Representing images and videos into a format suitable for existing pattern mining tools is not a trivial task. We chose to use Bag-of-Visual-Word computed from 128-D SIFT descriptors to convert images or frames into histograms. These histograms are more similar to the traditional pattern mining transactional format. Specifically, each keyframe is densely sampled into numerous local patches which are then represented by a SIFT descriptor. Inside this 128-D space, all SIFT descriptors from all keyframes of all videos are clustered into multiple (for example, 1000) groups. The center of each group is called a visual word, and all SIFT descriptors belonging to the group are associated to a unique visual word. Keyframes then become a set (bag) of visual words which can appear multiple times. There are two popular ways to generate

histograms from this set: one histogram per keyframe (called global histogram), or one histogram for each local patch and thus multiple histograms for one keyframe (called local histograms). The histograms must then be binarized. The binarization step for the local histograms is straightforward since the variations of the bin values within the histograms are small (and depends on the size of local patch), i.e., in general, the bin values are already 0 or 1. For global histogram, the variability is much more important and some more complex methods based on discretization should be used.

Local frequent patterns generation. Then, we have extracted frequent patterns in the set of transactions representing the videos. We use the existing LCM implementation to extract classic closed and maximal frequent patterns. However, the number of patterns is often extremely large (millions of patterns), and they can not be directly used to encode images or videos. By applying the *minimum description length* principle and combine it with some heuristic approaches (as done in by the KRIMP and SLIM algorithms), we are able to significantly reduce the number of patterns. We also make use of tag information to keep emerging patterns (in our case, patterns that are 5 times more frequent in the set of video having a given tag than in the set of videos without the tag). With this process, we succeeded in finding better mid-level visual features for a better propagation process. However, we have demonstrated that finding good patterns without supervised information is still a challenging problem. We tried different approaches of selecting the patterns without using the initial tag information but all these results are only comparable with the results obtained with primitive visual features.

Local is better than global. The experiments with the image dataset *Oxford-Flower17* show that local features have more potential than global features when applying data mining tools: all global results are worse than our baseline, while all local results are better than the baseline (see e.g. Table 2.3 for global results, and Table 2.4 for corresponding local results). Moreover, with local histograms, the binarization is easier to implement without losing information.

Distance functions and video comparisons. Multiple distance functions, from simple the $L1$ distance to the more complex *intersection kernel*, are examined. With a pre-defined distance function, comparing two keyframes is effortless since a keyframe is often encoded as a vector. To compare two videos represented as a set of vectors we tried multiple comparison methods: from simply taking the average of all keyframes, to an asymmetric pairwise distance which compare all pairs of keyframes of two videos. The simplest method is often selected for its good tradeoff performance/computation time.

Tag propagation procedure. We have presented an intuitive scheme to propagate the tags mainly based on the neighborhood of the videos. Our method takes into account both the global impact (the tag’s distribution at the dataset level) and the local impact (the tag’s distribution in the video neighborhood) when making the propagation decision. A tag of a video is removed if it never appears in the neighborhood of that video.

Dataset and evaluation. Due to the lack of properly labeled datasets and evaluation procedure, we have created multiple datasets, each one for a different purpose. By using a synthetic dataset in Chapter 3, we undoubtedly prove that there is room for improvement towards primitive visual features like SIFT-BoW. With wisely chosen visual features, it is possible to construct a better neighborhood with really similar videos and thus obtain better propagated tags. The next video dataset of 668 videos with 150 tags is used to demonstrate our framework’s effectiveness: considering 40 neighbors, the baseline returns 44% of good corrections while our patterns returns 64% of good corrections (see Figure 3.12). The size of our video datasets is still not comparable to, e.g. the number of Youtube videos but they are large enough to illustrate the complexity and difficulty of our problems.

4.2 Future Works

Working with a bigger dataset. 668 videos are a large number of videos, but this obviously does not reach the scale of video-sharing websites. Thus, using our framework on a bigger dataset is a straightforward perspective. With a larger number of videos, we expect to be able to find more similar videos while keeping the same number of nearest neighbors, which in return might increase the performance of the propagation step. However, with a larger dataset, several problems arise: visual words generation uses a clustering algorithm that may become intractable when the number of SIFT descriptors is too large. One way to solve this problem is to use a subset of the SIFT descriptors in this phase. Pattern mining itself can also become problematic, even if current mining algorithms scales well with respect to the number of transactions (and the number of attributes does not increase). Finally, the distance function between videos is expensive and we must compute, for each video, its nearest neighbors. In our experiments, we used a naive quadratic procedure to compute this neighborhood and this would not be tractable with millions of videos.

Incremental video dataset. Large video datasets are often build in an incre-

mental fashion (In Youtube for instance, thousands of videos are added daily). Thus, being able to correct the tags of the videos as they are added would be a significant improvement on our framework. Given the visual features, it is easy to compute the description of a new video using them. Then, computing the nearest neighbors would still be a significant challenge. But how can we update the visual features that are used to describe our videos as new videos arrive ? If the new videos are very different than the videos already in the dataset, it would probably be interesting to update the visual features. However, doing so would imply to recompute the descriptors of all videos (including those already in the dataset), recompute all the neighborhoods and update the tags. The cost of such an operation would probably makes it impossible if done in a naive way.

Using different visual features. In this thesis, we tried with only one primitive visual feature: SIFT-BoW. Applying the same principle with different visual features might return better results. There is no limitation on choosing the primitive features to represent the videos, therefore high level features, such as human faces, car's wheel... can also be good candidates.

Applying sequence pattern mining. Even if we tried several sequence mining methods unsuccessfully (Section 3.1.3), applying sequence mining into videos feels so natural since a video is basically a time series data. With current sequence mining algorithm, the number of output patterns grows too fast, and they are often redundant. By developing a SLIM-like sequence mining method, it might become possible to mine a reasonable number of discriminative sequence patterns in videos. These patterns could then be used either alone, or combined with non sequential patterns in our framework.

References

- [1] HERVÉ ABDI AND LYNNE J WILLIAMS. Principal component analysis. *Wiley Interdisciplinary Reviews Computational Statistics*, **2**[713587337], 2010. [46](#)
- [2] ANKUR AGARWAL AND BILL TRIGGS. Hyperfeatures–multilevel local coding for visual recognition. In *Computer Vision–ECCV 2006*, pages 30–43. Springer, 2006. [16](#)
- [3] RAKESH AGRAWAL, TOMASZ IMIELIŃSKI, AND ARUN SWAMI. Mining association rules between sets of items in large databases. In *ACM SIGMOD Record*, **22**, pages 207–216. ACM, 1993. [18](#), [19](#), [20](#), [32](#)
- [4] RAKESH AGRAWAL AND RAMAKRISHNAN SRIKANT. Fast algorithms for mining association rules in large databases. In *VLDB*, pages 487–499, 1994. [20](#)
- [5] RAKESH AGRAWAL AND RAMAKRISHNAN SRIKANT. Mining sequential patterns. In *Data Engineering, 1995. Proceedings of the Eleventh International Conference on*, pages 3–14. IEEE, 1995. [21](#)
- [6] ARNON AMIR, MARCO BERG, SHIH-FU CHANG, WINSTON HSU, GIRIDHARAN IYENGAR, CHING-YUNG LIN, MILIND NAPHADE, APOSTOL NATSEV, CHALAPATHY NETI, HARRIET NOCK, ET AL. Ibm research trecvid-2003 video retrieval system. *NIST TRECVID-2003*, 2003. [7](#), [8](#), [36](#)
- [7] J. AYRES, J. FLANNICK, J. GEHRKE, AND T. YIU. Sequential pattern mining using a bitmap representation. In *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 429–435. ACM, 2002. [22](#)
- [8] LEI BAO, SHOOU-I YU, ZHEN-ZHONG LAN, ARNOLD OVERWIJK, QIN JIN, BRIAN LANGNER, MICHAEL GARBUS, SUSANNE BURGER, FLORIAN METZE,

-
- AND ALEXANDER HAUPTMANN. Informedia@ trecvid 2011. *TRECVID2011, NIST*, 2011. 36
- [9] FAISAL I BASHIR, ASHFAQ A KHOKHAR, AND DAN SCHONFELD. Real-time motion trajectory-based indexing and retrieval of video sequences. *Multimedia, IEEE Transactions on*, **9**[1]:58–65, 2007. 9
- [10] TINNE TUYTELAARS BASURA FERNANDO. Mining multiple queries for image retrieval: On-the-fly learning of an object-specific mid-level representation. In *ICCV*, 2013. 13
- [11] G CAMARA-CHAVEZ, F PRECIOSO, M CORD, S PHILLIP-FOLIGUET, AND A DE A ARAUJO. Shot boundary detection by a hierarchical supervised approach. In *Systems, Signals and Image Processing, 2007 and 6th EURASIP Conference focused on Speech and Image Processing, Multimedia Communications and Services. 14th International Workshop on*, pages 197–200. IEEE, 2007. 14
- [12] J. CAO, Y.D. ZHANG, Y.C. SONG, Z.N. CHEN, X. ZHANG, AND J.T. LI. MCG-WEBV: A benchmark dataset for web video analysis. Technical report, ICT-MCG-09-001, Institute of Computing Technology, 2009. 56, 65
- [13] ZUZANA CERNEKOVA, IOANNIS PITAS, AND CHRISTOPHOROS NIKOU. Information theory-based shot cut/fade detection and video summarization. *Circuits and Systems for Video Technology, IEEE Transactions on*, **16**[1]:82–91, 2006. 15
- [14] YUCHOU CHANG, DAH-JYE LEE, YI HONG, AND JAMES ARCHIBALD. Unsupervised video shot detection using clustering ensemble with a color global scale-invariant feature transform descriptor. *Journal on Image and Video Processing*, **2008**:9, 2008. 14
- [15] G CAMARA CHAVEZ, F PRECIOSO, M CORD, S PHILIPP-FOLIGUET, AND ARNALDO DE A ARAUJO. Shot boundary detection at trecvid 2006. *Proc. TREC Video Retrieval Eval*, 2006. 15
- [16] LIN CHEN, DONG XU, IVOR W TSANG, AND JIEBO LUO. Tag-based web photo retrieval improved by batch mode re-tagging. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3440–3446. IEEE, 2010. 38

-
- [17] HONG CHENG, XIFENG YAN, JIAWEI HAN, AND CHIH-WEI HSU. Discriminative frequent pattern analysis for effective classification. In *ICDE*, pages 716–725, april 2007. 32
- [18] MATTHEW COOPER, TING LIU, AND ELEANOR RIEFFEL. Video segmentation via temporal pattern classification. *Multimedia, IEEE Transactions on*, **9**[3]:610–618, 2007. 14
- [19] THOMAS M COVER AND JOY A THOMAS. *Elements of information theory*. John Wiley & Sons, 2012. 24
- [20] N. DALAL AND B. TRIGGS. Histograms of oriented gradients for human detection. In *Conf. on Computer Vision and Pattern Recognition*, **1**, pages 886–893, 2005. 8
- [21] MINH-SON DAO, FGB DENATALE, AND ANDREA MASSA. Video retrieval using video object-trajectory and edge potential function. In *Intelligent Multimedia, Video and Speech Processing, 2004. Proceedings of 2004 International Symposium on*, pages 454–457. IEEE, 2004. 9
- [22] F DIRFAUX. Key frame selection to represent a video. In *Image Processing, 2000. Proceedings. 2000 International Conference on*, **2**, pages 275–278. IEEE, 2000. 39
- [23] GUOZHU DONG AND JINYAN LI. Efficient mining of emerging patterns: discovering trends and differences. In *Int. Conf. on Knowledge Disc. and Data Mining*, pages 43–52, 1999. 72
- [24] DAVID L DONOHO ET AL. High-dimensional data analysis: The curses and blessings of dimensionality. *AMS Math Challenges Lecture*, pages 1–32, 2000. 42
- [25] LIXIN DUAN, DONG XU, IVOR WAI-HUNG TSANG, AND JIEBO LUO. Visual event recognition in videos by learning from web data. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, **34**[9]:1667–1680, 2012. 36
- [26] SUSAN DUMAIS, JOHN PLATT, DAVID HECKERMAN, AND MEHRAN SAHAMI. Inductive learning algorithms and representations for text categorization. In *Proceedings of the seventh international conference on Information and knowledge management*, pages 148–155. ACM, 1998. 36

-
- [27] M. EVERINGHAM, L. VAN GOOL, C. K. I. WILLIAMS, J. WINN, AND A. ZISSERMAN. The pascal visual object classes (voc) challenge. *International Journal of Computer Vision*, **88**[2]:303–338, June 2010. [16](#)
- [28] A. FARHADI, I. ENDRES, D. HOIEM, AND D. FORSYTH. Describing objects by their attributes. In *CVPR*, pages 1778–1785, june 2009. [32](#)
- [29] A MÜFIT FERMAN AND A MURAT TEKALP. Two-stage hierarchical video summary extraction to match low-level user browsing preferences. *Multimedia, IEEE Transactions on*, **5**[2]:244–256, 2003. [15](#)
- [30] BASURA FERNANDO, ELISA FROMONT, AND TINNE TUYTELAARS. Effective use of frequent itemset mining for image classification. In *Europ. Conf. on Computer Vision*, pages 214–227, 2012. [31](#), [43](#), [44](#)
- [31] COLUM FOLEY, CATHAL GURRIN, GARETH JF JONES, HYOWON LEE, SINÉAD MCGIVNEY, NOEL E O’CONNOR, SORIN SAV, ALAN F SMEATON, AND PETER WILKINS. Trecvid 2005 experiments at dublin city university. *Proc. TREC Video Retrieval Eval*, 2005. [8](#)
- [32] XINBO GAO, JIE LI, AND YANG SHI. A video shot boundary detection algorithm based on feature tracking. In *Rough Sets and Knowledge Technology*, pages 651–658. Springer, 2006. [14](#)
- [33] A. GILBERT, J. ILLINGWORTH, AND R. BOWDEN. Fast realistic multi-action recognition using mined dense spatio-temporal features. In *ICCV*, pages 925–931, 29 2009-oct. 2 2009. [32](#)
- [34] RAFAEL C GONZALEZ, RICHARD E WOODS, AND STEVEN L EDDINS. *Digital image processing using MATLAB*, **2**. Gatesmark Publishing Knoxville, 2009. [viii](#), [7](#)
- [35] PETER D GRÜNWARD. *The minimum description length principle*. The MIT Press, 2007. [22](#)
- [36] MATTHIEU GUILLAUMIN, THOMAS MENSINK, JAKOB VERBEEK, AND CORDELIA SCHMID. Tagprop: Discriminative metric learning in nearest neighbor models for image auto-annotation. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 309–316. IEEE, 2009. [37](#)

-
- [37] AMIRHOSSEIN HABIBIAN, KOEN EA VAN DE SANDE, AND CEES GM SNOEK. Recommendations for video event recognition using concept vocabularies. In *Proceedings of the 3rd ACM conference on International conference on multimedia retrieval*, pages 89–96. ACM, 2013. [37](#)
- [38] BING HAN, XINBO GAO, AND HONGBING JI. A shot boundary detection method for news video based on rough-fuzzy sets. *Int. J. Inform. Technol*, **11**:101–111, 2005. [14](#)
- [39] JIAWEI HAN, JIAN PEI, AND YIWEN YIN. Mining frequent patterns without candidate generation. In *Proceedings of the 2000 ACM SIGMOD international conference on Management of data*, SIGMOD '00, pages 1–12, New York, NY, USA, 2000. ACM. [19](#), [20](#)
- [40] ROBERT M HARALICK. Statistical and structural approaches to texture. *Proceedings of the IEEE*, **67**[5]:786–804, 1979. [8](#)
- [41] CHRIS HARRIS AND MIKE STEPHENS. A combined corner and edge detector. In *Alvey vision conference*, **15**, page 50. Manchester, UK, 1988. [9](#)
- [42] ZELIG SABBETAI HARRIS. Word. *Distributional Structure*, **10**[23]:146–162, 1954. [12](#)
- [43] TAL HASSNER, VIKI MAYZELS, AND LIHI ZELNIK-MANOR. On sifts and their scales. In *CVPR*, pages 1522–1528. IEEE, 2012. [10](#)
- [44] ALEX HAUPTMANN, ROBERT V BARON, MING-YU CHEN, M CHRISTEL, PINAR DUYGULU, C HUANG, R JIN, W-H LIN, T NG, AND N MORAVEJI. Informedia at trecvid 2003: Analyzing and searching broadcast news video. Technical report, DTIC Document, 2004. [8](#)
- [45] STEVEN CH HOI, LAWSON LS WONG, AND ALBERT LYU. Chinese university of hongkong at trecvid 2006: Shot boundary detection and video search. In *TRECVID 2006 Workshop*, pages 76–86, 2006. [14](#)
- [46] WEIMING HU, NIANHUA XIE, LI LI, XIANGLIN ZENG, AND STEPHEN J. MAYBANK. A survey on visual content-based video indexing and retrieval. *IEEE Transactions on Systems, Man, and Cybernetics*, **41**[6]:797–819, 2011. [1](#)
- [47] MIHIR JAIN AND C. V. JAWAHAR. Characteristic pattern discovery in videos. In *Proceedings of the Seventh Indian Conf. on Computer Vision, Graphics and*

-
- Image Processing*, ICVGIP '10, pages 306–313, New York, NY, USA, 2010. ACM. [32](#)
- [48] HONG-WEN KANG AND XIAN-SHENG HUA. To learn representativeness of video frames. In *Proceedings of the 13th annual ACM international conference on Multimedia*, pages 423–426. ACM, 2005. [16](#)
- [49] S. KIM, X. JIN, AND J. HAN. Disiclass: discriminative frequent pattern-based image classification. In *Proceedings of the Tenth Int. Workshop on Multimedia Data Mining*, pages 1–10. ACM, 2010. [31](#), [32](#)
- [50] SVETLANA LAZEBNIK, CORDELIA SCHMID, AND JEAN PONCE. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, **2**, pages 2169–2178. IEEE, 2006. [16](#)
- [51] MING LI AND PAUL VITÁNYI. An introduction to kolmogorov complexity and its applications. texts and monographs in computer science, 1993. [22](#)
- [52] XIRONG LI, CEES GM SNOEK, AND MARCEL WORRING. Learning social tag relevance by neighbor voting. *Multimedia, IEEE Transactions on*, **11**[7]:1310–1322, 2009. [38](#)
- [53] XIRONG LI, CEES GM SNOEK, AND MARCEL WORRING. Unsupervised multi-feature tag relevance learning for social image retrieval. In *Proceedings of the ACM International Conference on Image and Video Retrieval*, pages 10–17. ACM, 2010. [38](#)
- [54] DONG LIU, XIAN-SHENG HUA, MENG WANG, AND HONG-JIANG ZHANG. Image retagging. In *Proceedings of the International Conference on Multimedia, MM '10*, pages 491–500, New York, NY, USA, 2010. ACM. [37](#)
- [55] D.G. LOWE. Distinctive image features from scale-invariant keypoints. *Int. journal of computer vision*, **60**[2]:91–110, 2004. [6](#), [8](#), [9](#)
- [56] HONG LU, YAP-PENG TAN, XIANGYANG XUE, AND LIDE WU. Shot boundary detection using unsupervised clustering and hypothesis testing. In *Communications, Circuits and Systems, 2004. ICCAS 2004. 2004 International Conference on*, **2**, pages 932–936. IEEE, 2004. [14](#)

-
- [57] YU-FEI MA AND HONG-JIANG ZHANG. Motion texture: a new motion based video representation. In *Pattern Recognition, 2002. Proceedings. 16th International Conference on*, **2**, pages 548–551. IEEE, 2002. [8](#)
- [58] AMEESH MAKADIA, VLADIMIR PAVLOVIC, AND SANJIV KUMAR. A new baseline for image annotation. In *Computer Vision–ECCV 2008*, pages 316–329. Springer, 2008. [37](#)
- [59] MICHELE MERLER, BERT HUANG, LEXING XIE, GANG HUA, AND APOSTOL NATSEV. Semantic model vectors for complex video event recognition. *Multimedia, IEEE Transactions on*, **14**[1]:88–101, 2012. [36](#)
- [60] CARL H MOONEY AND JOHN F RODDICK. Sequential pattern mining—approaches and algorithms. *ACM Computing Surveys (CSUR)*, **45**[2]:19, 2013. [22](#)
- [61] HANS P MORAVEC. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, DTIC Document, 1980. [9](#)
- [62] N. MORSILLO, G. MANN, AND C. PAL. Youtube scale, large vocabulary video annotation. *Video Search and Mining*, pages 357–386, 2010. [3](#), [7](#), [8](#), [15](#)
- [63] E. MOXLEY, T. MEI, AND BS MANJUNATH. Video annotation through search and graph reinforcement mining. *IEEE Transactions on Multimedia*, **12**[3]:184–193, 2010. [7](#), [8](#), [34](#)
- [64] CHIKASHI YAJIMAT YOSHIHIRO NAKANISHI AND KATSUMI TANAKA. Querying video data by spatio-temporal relationships of moving object traces. In *Visual and Multimedia Information Management: IFIP TC 2/WG 2.6 Sixth Working Conference on Visual Database Systems, May 29-31, 2002, Brisbane, Australia*, page 357. Kluwer Academic Pub, 2002. [9](#)
- [65] M.E. NILSBACK AND A. ZISSERMAN. Automated flower classification over a large number of classes. In *Computer Vision, Graphics & Image Processing, 2008. ICVGIP'08. Sixth Indian Conf. on*, pages 722–729. IEEE, 2008. [32](#), [40](#)
- [66] ERIC NOWAK, FRÉDÉRIC JURIE, AND BILL TRIGGS. Sampling strategies for bag-of-features image classification. In *Computer Vision–ECCV 2006*, pages 490–503. Springer, 2006. [9](#)

-
- [67] A. OPELT, M. FUSSENEGGER, A. PINZ, AND P. AUER. Weak hypotheses and boosting for generic object detection and recognition. *Computer Vision-ECCV 2004*, pages 71–84, 2004. [40](#)
- [68] PAUL OVER, GEORGE AWAD, MARTIAL MICHEL, JONATHAN FISCUS, GREG SANDERS, WESSEL KRAAIJ, ALAN F. SMEATON, AND GEORGES QUÉENOT. Trecvid 2013 – an overview of the goals, tasks, data, evaluation mechanisms and metrics. In *Proceedings of TRECVID 2013*. NIST, USA, 2013. [36](#)
- [69] TAN PANG-NING, MICHAEL STEINBACH, VIPIN KUMAR, ET AL. Introduction to data mining. In *Library of Congress*, 2006. [viii](#), [19](#), [20](#)
- [70] JIAN PEI, JIAWEI HAN, BEHZAD MORTAZAVI-ASL, JIANYONG WANG, HELEN PINTO, QIMING CHEN, UMESHWAR DAYAL, AND MEI-CHUN HSU. Mining sequential patterns by pattern-growth: The prefixspan approach. *Knowledge and Data Engineering, IEEE Transactions on*, **16**[11]:1424–1440, 2004. [22](#)
- [71] JIAN PEI, HELEN PINTO, QIMING CHEN, JIAWEI HAN, BEHZAD MORTAZAVI-ASL, UMESHWAR DAYAL, AND MEI-CHUN HSU. Prefixspan: Mining sequential patterns efficiently by prefix-projected pattern growth. In *2013 IEEE 29th International Conference on Data Engineering (ICDE)*, pages 0215–0215. IEEE Computer Society, 2001. [22](#)
- [72] SARAH VICTORIA PORTER. *Video segmentation and indexing using motion estimation*. PhD thesis, University of Bristol, 2004. [14](#)
- [73] T. QUACK, V. FERRARI, B. LEIBE, AND L. VAN GOOL. Efficient mining of frequent and distinctive feature configurations. In *Proceedings 11th IEEE international conference on computer vision, ICCV 2007*, 2007. [32](#)
- [74] T. QUACK, V. FERRARI, AND L. VAN GOOL. Video mining with frequent itemset configurations. In *ACM Int. Conf. on Image and Video Retrieval, CIVR*, 2006. [32](#)
- [75] GEORGES M QUÉENOT, DANIEL MORARU, AND LAURENT BESACIER. Clips at trecvid: Shot boundary detection and feature detection. *TRECVID*, 2003. [15](#)
- [76] GERARD SALTON AND MICHAEL MCGILL. *Introduction to Modern Information Retrieval*. McGraw-Hill Book Company, 1984. [1](#)

-
- [77] JIALIE SHEN, MENG WANG, SHUICHENG YAN, AND XIAN-SHENG HUA. Multi-media tagging: past, present and future. In *19th ACM Int. Conf. on Multimedia*, pages 639–640, 2011. [2](#), [3](#)
- [78] J. SIVIC AND A. ZISSERMAN. Video data mining using configurations of view-point invariant regions. In *IEEE Conf. on Computer Vision and Pattern Recognition*, **1**, pages 1–488, 2004. [32](#)
- [79] A F SMEATON, P OVER, AND W KRAAIJ. Evaluation campaigns and TRECVID. In *Proceedings of the 8th ACM international workshop on Multimedia information retrieval*, pages 321–330. ACM, 2006. [16](#), [36](#)
- [80] K. SMETS AND J. VREEKEN. SLIM: Directly mining descriptive patterns. In *SIAM Int. Conf. on Data Mining*, pages 236–247, 2012. [28](#)
- [81] CHIH-WEN SU, H-YM LIAO, HSIAO-RONG TYAN, CHIA-WEN LIN, DUAN-YU CHEN, AND KUO-CHIN FAN. Motion flow-based video retrieval. *Multimedia, IEEE Transactions on*, **9**[6]:1193–1201, 2007. [9](#)
- [82] CHEN SUN AND RAM NEVATIA. Large-scale web video event classification by use of fisher vectors. In *Applications of Computer Vision (WACV), 2013 IEEE Workshop on*, pages 15–22. IEEE, 2013. [16](#), [36](#)
- [83] ZHONGHUA SUN, KEBIN JIA, AND HEXIN CHEN. Video key frame extraction based on spatial-temporal color distribution. In *Intelligent Information Hiding and Multimedia Signal Processing, 2008. IHHMSP'08 International Conference on*, pages 196–199. IEEE, 2008. [15](#)
- [84] HIDEYUKI TAMURA, SHUNJI MORI, AND TAKASHI YAMAWAKI. Textural features corresponding to visual perception. *Systems, Man and Cybernetics, IEEE Transactions on*, **8**[6]:460–473, 1978. [8](#)
- [85] NIKOLAJ TATTI AND JILLES VREEKEN. The long and the short of it: summarising event sequences with serial episodes. In *KDD*, pages 462–470. ACM, 2012. [22](#)
- [86] G. TODERICI, H. ARADHYE, M. PASCA, L. SBAIZ, AND J. YAGNIK. Finding meaning on youtube: Tag recommendation and category discovery. In *IEEE Conf. on Computer Vision and Pattern Recognition*, pages 3447–3454, 2010. [6](#), [7](#), [8](#), [34](#)

-
- [87] BA TU TRUONG AND SVETHA VENKATESH. Video abstraction: A systematic review and classification. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, **3**[1]:3, 2007. [15](#)
- [88] MARK R TURNER. Texture discrimination by gabor functions. *Biological Cybernetics*, **55**[2-3]:71–82, 1986. [8](#)
- [89] JASPER RR UIJLINGS, ARNOLD WM SMEULDERS, AND REMKO JH SCHA. Real-time visual concept classification. *Multimedia, IEEE Transactions on*, **12**[7]:665–681, 2010. [9](#)
- [90] A. ULGES, C. SCHULZE, M. KOCH, AND T.M. BREUEL. Learning automatic concept detectors from online video. *Computer vision and Image understanding*, **114**[4]:429–438, 2010. [36](#)
- [91] T. UNO, M. KIYOMI, AND H. ARIMURA. LCM ver. 3: collaboration of array, bitmap and prefix tree for frequent itemset mining. In *1st Int. Workshop on Open Source Data Mining: Frequent Pattern Mining Implementations*, pages 77–86. ACM, 2005. [21](#), [32](#), [43](#)
- [92] TIBERIO URICCHIO, LAMBERTO BALLAN, MARCO BERTINI, AND ALBERTO DEL BIMBO. An evaluation of nearest neighbor methods for tag refinement. *Proc. of IEEE ICME*, 2013. [38](#)
- [93] J. VAN DE WEIJER AND C. SCHMID. Applying color names to image description. In *Image Processing, 2007. ICIP 2007. IEEE Int. Conf. on*, **3**, pages III–493. IEEE, 2007. [41](#)
- [94] WINN VORAVUTHIKUNCHAI, BRUNO CRÉMILLEUX, FRÉDÉRIC JURIE, ET AL. Finding groups of duplicate images in very large dataset. In *Proceedings of the British Machine Vision Conference (BMVC 2012)*, 2012. [32](#)
- [95] WINN VORAVUTHIKUNCHAI, BRUNO CRÉMILLEUX, FRÉDÉRIC JURIE, ET AL. Histograms of pattern sets for image classification and object recognition. In *IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–8, 2014. [30](#), [32](#)
- [96] J. VREEKEN, M. VAN LEEUWEN, AND A. SIEBES. KRIMP: mining itemsets that compress. *Data Mining and Knowledge Discovery*, pages 1–46, 2011. [viii](#), [23](#), [24](#), [25](#), [26](#)

-
- [97] M. WANG AND X.S. HUA. Active learning in multimedia annotation and retrieval: A survey. *ACM Transactions on Intelligent Systems and Technology (TIST)*, **2**[2]:10, 2011. [37](#)
- [98] MENG WANG, BINGBING NI, XIAN-SHENG HUA, AND TAT-SENG CHUA. Assistive tagging: A survey of multimedia tagging with human-computer joint exploration. *ACM Computing Surveys (CSUR)*, **44**[4]:25, 2012. [33](#)
- [99] DINGYUAN XIA, XUEFEI DENG, AND QINGNING ZENG. Shot boundary detection based on difference sequences of mutual information. In *Image and Graphics, 2007. ICIG 2007. Fourth International Conference on*, pages 389–394. IEEE, 2007. [15](#)
- [100] RONG YAN AND ALEXANDER G HAUPTMANN. A review of text and image retrieval approaches for broadcast news video. *Information Retrieval*, **10**[4-5]:445–484, 2007. [7](#)
- [101] X. YAN, J. HAN, AND R. AFSHAR. Clospan: Mining closed sequential patterns in large datasets. In *Proceedings of SIAM Int. Conf. on Data Mining*, pages 166–177, 2003. [22](#)
- [102] J. YANG, Y.G. JIANG, A.G. HAUPTMANN, AND C.W. NGO. Evaluating bag-of-visual-words representations in scene classification. In *Int. Workshop on Multimedia Information Retrieval*, pages 197–206. ACM, 2007. [6](#)
- [103] MINGQIANG YANG, KIDIYO KPALMA, JOSEPH RONSIN, ET AL. A survey of shape feature extraction techniques. *Pattern recognition*, pages 43–90, 2008. [8](#)
- [104] W. YANG AND G. TODERICI. Discriminative tag learning on youtube videos with latent sub-tags. In *Computer Vision and Pattern Recognition*, pages 3217–3224, 2011. [8](#), [34](#)
- [105] YANG YANG, YI YANG, AND HENG TAO SHEN. Effective transfer tagging from image to video. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMCCAP)*, **9**[2]:14, 2013. [36](#)
- [106] YANG YANG, ZHENG-JUN ZHA, HENG TAO SHEN, AND TAT-SENG CHUA. Robust semantic video indexing by harvesting web images. In *Advances in Multimedia Modeling*, pages 70–80. Springer, 2013. [36](#)

-
- [107] BANGPENG YAO AND LI FEI-FEI. Grouplet: A structured image representation for recognizing human and object interactions. In *CVPR*, 2010. 32
- [108] JINHUI YUAN, HUIYI WANG, LAN XIAO, WUJIE ZHENG, JIANMIN LI, FUZONG LIN, AND BO ZHANG. A formal study of shot boundary detection. *Circuits and Systems for Video Technology, IEEE Transactions on*, 17[2]:168–186, 2007. 14
- [109] JUNSONG YUAN, JIEBO LUO, AND YING WU. Mining compositional features for boosting. In *CVPR*, 2008. 32
- [110] JUNSONG YUAN, YING WU, AND MING YANG. Discovery of collocation patterns: from visual words to visual phrases. In *Conf. on Computer Vision and Pattern Recognition, 18-23 June 2007, Minneapolis*, 2007. 32
- [111] JUNSONG YUAN, MING YANG, AND YING WU. Mining discriminative co-occurrence patterns for visual recognition. In *Conf. on Comp. Vision and Pat. Recognition*, pages 2777–2784, 2011. 32
- [112] MOHAMMED J. ZAKI. Scalable algorithms for association mining. *IEEE Trans. on Knowl. and Data Eng.*, 12[3]:372–390, May 2000. 22
- [113] MOHAMMED J ZAKI. Spade: An efficient algorithm for mining frequent sequences. *Machine learning*, 42[1-2]:31–60, 2001. 22
- [114] MOHAMMED JAVEED ZAKI. Scalable algorithms for association mining. *Knowledge and Data Engineering, IEEE Transactions on*, 12[3]:372–390, 2000. 19
- [115] HONGJIANG ZHANG, ATREYI KANKANHALLI, AND STEPHEN W SMOLIAR. Automatic partitioning of full-motion video. *Multimedia systems*, 1[1]:10–28, 1993. 56
- [116] XU-DONG ZHANG, TIE-YAN LIU, KWOK-TUNG LO, AND JIAN FENG. Dynamic selection and effective compression of key frames for video abstraction. *Pattern recognition letters*, 24[9]:1523–1532, 2003. 15
- [117] ZHI-CHENG ZHAO AND AN-NI CAI. Shot boundary detection algorithm in compressed domain based on adaboost and fuzzy theory. In *Advances in Natural Computation*, pages 617–626. Springer, 2006. 15
- [118] ZHI-CHENG ZHAO, XING ZENG, TAO LIU, AND AN-NI CAI. Bupt at trecvid 2007: Shot boundary detection. In *TRECVID*. Citeseer, 2007. 15

REFERENCES

- [119] Y. ZHUANG, Y. RUI, T.S. HUANG, AND S. MEHROTRA. Adaptive key frame extraction using unsupervised clustering. In *Int. Conf. on Image Processing*, pages 866–870, 1998. [16](#)