THESIS TITLE: Feature Analysis of Diagrams with applications to retrieval and classification

AUTHOR: Mingyan Shao

Ph.D. Thesis Approved to complete all degree requirements for the Ph.D. Degree in Computer Science.

Thesis Advisor: Date

Thesis Reader: Date

Thesis Reader: Date

GrADUATE SCHOOL APPROVAL:

Director, Graduate School: Date

ORIGINAL COPY DEPOSITED IN LIBRARY:

Reference Librarian: Date

COPY RECEIVED IN GRADUATE SCHOOL OFFICE:

Recipient's Signature: Date

Distribution: Original to Reference Library (Boxed- with signature approval sheet)
One Copy to Thesis Advisor
One Copy to Each Member of Thesis Committee
One Copy to Director of Graduate School
One Copy to Graduate School Office (with signature approval sheet, including all signatures)

Copies submitted to those other than Reference Library should be bound in a black hardcover binder with a squeeze back. (Available in N.U. Bookstore).
FEATURE ANALYSIS OF DIAGRAMS WITH APPLICATIONS TO RETRIEVAL AND CLASSIFICATION

Submitted to

COLLEGE OF COMPUTER AND INFORMATION SCIENCE
NORTHEASTERN UNIVERSITY

in Partial Fulfillment of the Requirements for the Degree
of Doctor of Philosophy in Computer Science

March 19, 2010

by

Mingyan SHAO

Thesis Advisor: Robert P. Futrelle

Committee Members :  Javed A. Aslam  -  Northeastern University
                    Margrit Betke  -  Boston University
                    Harriet Fell  -  Northeastern University
                    Ronald J. Williams  -  Northeastern University
This dissertation is dedicated to

*my husband, Lei Chen,*

*my son, Bowang Chen,*

*and the memory of my mother, Xu Hui.*

for their love and support.
Acknowledgments

I owe my gratitude to all those people who have made this dissertation possible and because of whom my graduate experience has been one that I will cherish forever.

I would like to express the deepest appreciation to my advisor, Prof. Robert Futrelle, for his excellent guidance on my research. His persistence and optimism conveyed the spirit of research and scholarship. His encouragements and patience helped me overcome many tough situations and finish this dissertation. I would never have been able to finish my dissertation without him.

I would like to thank Prof. Jay Aslam, who not only guided my research on retrieval and machine learning but helped me to develop the solid theoretical statement for the thesis.

I would like to thank Prof. Harriet Fell, who patiently corrected my writing and gave me great support and warm encouragement.

I am specially grateful to Prof. Margrit Betke who willing to serve on my committee despite our having met only briefly, giving me advice on image processing, and supporting me during my thesis defense.

I would like to thank Prof. Ronald Williams for his suggestions on machine learning.

My thanks also go to Dr. Mel Simms and Prof. Viera Proulx for their financial support from 2003 to 2004.

So many friends and CCIS colleagues have supported and encouraged me on my long journey on this thesis. My sincere appreciation go to them: Prof. Agnes Chan, Chris Cieslik, Yang Du, Jun Gong, Andrea Elaina Grimes, Yongjian Huang, Yan Li, Yijun Luo, Xiaqin Ma, Dan Margo, Jeff Satterley, Jing Shan, Wenxu Tong, Virgil Pavlu, Yanning Wang, Tian Xia, Emine Yilmaz, Donghui Zhang, etc.

During the last few years of my thesis, I owe my deep thanks to Lei Xu, Ning Jia, and Christine Kong for their understanding and encouragement when I continue my long distance research on my thesis in New Jersey during 2007 and 2008, to Yue Zhao, Qiurong Xiao, Grandma Xiao, and Min Qin for their accommodation when I visited Boston. I am grateful to Mrs. Carolyn Futrelle for the warm conversations we have, the delicious meals she provided, her suggestions on my writing, and her support during my thesis defense. I also thank my lifelong friend, Danfeng Wang, for her understanding, blessing, and great support, especially during the last semester of my study.

Most importantly, none of this would have been possible without the love and patience of my family. My mother, Xu Hui, was the first person who inspired, trusted and encouraged me on the Ph.D. education. This thesis is for her. My husband, Lei Chen, and my son, Bowang Chen, have been proud, supportive, and sacrificing for my thesis during the past years. This thesis is for them. My father, Changtai Shao, my brother, Lixin Shao, and my sister, Wenwen Shao have been a constant source of love, support and strength all these years. I would like to express my deep gratitude to my aunt, Xiuping Xu, my uncle-in-law, Wenxiang Huang, my parents-in-law, Zhongfeng Chen and Feng’e Zhang, for their continuous support and caring my son when I am away from him. I thank my nieces: Caiwei Shao, Cailuo Fan, Caifei Fan, and Yujie Chi, and my nephew, Hengchen Zhao, for all the happy memories they have shared with me.
ABSTRACT

Millions of diagrams are available online in scientific and technical documents. The knowledge contained in diagrams is a rich resource, but one that has been little exploited, in comparison to text. In this thesis we demonstrate that it is possible to efficiently build compact representations of diagram content. The representations can support a wide variety of diagram-based systems such as retrieval, classification, and the building of knowledge bases that integrate text and diagrams. We demonstrate the strengths of our approach through studies of diagram retrieval as well as supervised and unsupervised machine learning for classification. The techniques are applied to a 700 diagram subset of more than 10,000 diagrams harvested from articles from the Open Access publisher, BioMed Central. A substantial set of Java-based tools was developed exclusively for this research. This will allow others to build on and extend what we have done.

The major accomplishment of this thesis has been in devising a representation of diagram content that can support high-quality retrieval and classification. Past work in content-based image retrieval (CBIR) has focused on approaches that include region analysis and texture in raster images. At the other extreme, detailed constraint-based grammars have been developed that can discover the full set of components in a diagram - the entities and their relations. CBIR has proven to be too imprecise, plagued by the "semantic gap". Full parsing does not scale well, because it would demand that thousands of complex grammar components would have to be developed.

Our results show that there is an intermediate approach, the grapheme, that is relatively simple to work with, scales well, and yet has the power to successfully retrieve and classify diagrams. Graphemes are simple sub-elements of diagrams such as data plots, bar charts, or tree structures. Graphemes include primitives such as horizontal and vertical lines, as well as composite structures such as tick marks, groups of bars (rectangles) and data point markers such as circles, triangles, diamonds, and rectangles. In the studies reported here, only twenty-four different types of graphemes were required to produce excellent results for retrieval and classification for a diagram collection made up of five diagram types. The description of a each diagram is a feature set, consisting of twenty-four attribute-value pairs: < grapheme, count >. The feature set can be described as a "bag of graphemes", similar to the "bag of words" description of documents that has been so successful in document retrieval. A feature vector forms a set of points in an n-dimensional feature space
which is used as input to unsupervised and supervised machine learning analysis.

Unsupervised learning was explored in a diagram information retrieval paradigm. A diagram is chosen as the query. The nearest diagrams, using a feature-based metric, are retrieved as a ranked set. As an example, when each diagram in a collection of bar charts is applied as a query on our full diagram collection, the mean precision of the results is 98.3% for the top ten diagrams returned. The accuracy of the results from supervised learning varies from 81% for point-based data plots to 93% for trees using the AdaBoostM1 algorithm.

We have focused on articles in PDF format that contain vector drawing instructions in a discrete and precise form. The technical challenges in extracting vector elements from PDF files are substantial. A PDF file is a sequence of commands for drawing primitives and for altering the graphics state containing line widths, colors, fonts, and other parameters. The sequence of commands does not map in any simple way to the two-dimensional layout of the document it represents. A Java-based emulator was designed to track the command sequences and graphic state changes, with the final result being a collection of static, non-procedural java object instances. Thus, the pair of commands to position at a line start and then draw to the line end, become a single line object defined by its start and end points. The resulting objects are installed in a spatial index that leads to a substantial speed-up in identifying graphemes.

It should be obvious that our approach can be extended to hundreds or even thousands of grapheme types. The results of this thesis demonstrate, beyond a doubt, that the grapheme-based approach is a powerful and potentially useful one for future information systems. Combined with text-based search, our techniques should lead to a unique new class of systems with greater versatility and breadth than any that exist today.
5 Graphemes

5.1 Definition of grapheme .................................................. 47
5.2 Extraction strategies ....................................................... 50
5.3 Representation of graphemes in feature sets ......................... 52
5.4 Pre-processing of grapheme values ..................................... 52
5.5 An example diagram with its grapheme values ....................... 52

6 Experiments and Analysis .................................................. 55

6.1 Weka 3: Data Mining software in Java ............................... 55
6.2 Retrieval experiments ...................................................... 56
   6.2.1 Mean precision at cut-off of 10 ................................. 56
   6.2.2 Mean average precision ........................................... 57
6.3 Classification ............................................................... 58
   6.3.1 The most powerful graphemes in binary classifications ....... 60
   6.3.2 Bar chart binary classification ................................... 64
   6.3.3 Data curve diagram binary classification ....................... 69
   6.3.4 Data line diagram binary classification ......................... 71
   6.3.5 Data point diagram binary classification ....................... 72
   6.3.6 Tree diagram binary classification ............................... 74
   6.3.7 Comparison of graphemes and histograms in binary classification 75
6.4 Multi-class classification ................................................ 77
   6.4.1 Comparison of multi-class and binary classification .......... 79
6.5 Clustering ................................................................. 80
   6.5.1 The five diagram classes used for clustering ................. 80
   6.5.2 Comparison of SimpleKMeans and EM .......................... 83

7 Summary and Future Work ................................................. 87

8 Appendix:

How to reproduce and extend the results of this thesis ............... 89
8.1 Diagram corpus .......................................................... 89
8.1.1 Collecting and filtering the PDF papers ......................................... 89
8.1.2 The interpreter and the creation of self-contained graphics objects .... 90
8.1.3 Building a spatial index for the objects ................................ ........... 90
8.2 Detecting graphemes and creating feature sets ................................. 91
8.3 The machine learning experiments .................................................... 91
  8.3.1 Supervised learning experiments ............................................... 92
  8.3.2 Unsupervised learning experiments ........................................... 92
  8.3.3 Diagram retrieval experiments ............................................... 92
References ......................................................................................... 92
List of Tables

5.1 Means and standard deviations of the grapheme values in our data set ............... 53
5.2 A diagram and its grapheme values and pre-processed values .......................... 54
6.1 Mean precisions at a cut off of 10 diagrams ................................................... 57
6.2 Mean average precisions of retrieval experiments ............................................. 57
6.3 Evaluate the binary classifications (one against others) on the five major diagram classes with TPR and FPR ................................................................. 60
6.4 Example diagrams in binary classifications of each of the 5 major diagram classes .. 61
6.5 Graphemes and the weights of decision stumps on these graphemes in binary classification of the five diagram classes ......................................................... 63
6.6 First 10 decision stumps in bar chart binary boosting classification .................... 66
6.7 First 10 decision stumps in data curve binary boosting classification .................. 70
6.8 First 10 decision stumps in data line binary boosting classification ..................... 72
6.9 First 10 decision stumps for data point diagram binary boosting classification ....... 73
6.10 First 10 decision stumps in tree binary boosting classification .......................... 74
6.11 Confusion matrix of multi-class classification of AdaBoost.M1 (C4.5 as base learner) on 5 major diagram classes ......................................................... 77
6.12 Metrics of AdaBoost.M1(C4.5 as base learner) on 5 major diagram classes .......... 78
6.13 Comparison of true positive rates for each individual diagram class in multi-class (boosted C4.5) and binary classification .................................................... 79
6.14 Effect of randomized starting points to clustering ............................................. 81
6.15 Comparison of the confusion matrices of SimpleKMeans and EM clustering with cluster number 4 ................................................................. 84
6.16 Comparison of the confusion matrices of SimpleKMeans and EM clustering with cluster number 5 ................................................................. 85
6.17 Comparison of the confusion matrices of SimpleKMeans and EM clustering with cluster number 8 ................................................................. 86
List of Figures

1.1 The five diagram classes studied in this thesis . . . . . . . . . . . . . . . . . . . . . . . 16
1.2 A bar chart with its magnified right corner in raster and vector formats. . . . . . . . 18

4.1 Three stages in preparing the corpus of vector diagrams . . . . . . . . . . . . . . . . 38
4.2 A simplified PDF structure example . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 40
4.3 A content stream example of a PDF page . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 41
4.4 A section of PDF source code of a triangle and rectangle. The first five commands
draw the triangle. The last three draw and fill the rectangle. . . . . . . . . . . . . . . . . . . . . . 44
4.5 A section of spatial index where a rectangle, a tick mark, and a horizontal line can
be found. The cell (3,3) contains a tick-mark consisting of two graphics primitives. . . . . . 45

5.1 Some grapheme examples: vertical tick, horizontal tick, line, curve, adjacent rectan-
gles, bars, arrow, and data point . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 48

6.1 Recalls for multi-class and binary classification experiments . . . . . . . . . . . . . . . . 58
6.2 Comparison of classification results on grapheme set and histogram . . . . . . . . . . . 76
6.3 Clustering experiment results of KMeans and EM with cluster number 5 . . . . . . . . 82
1.1 Motivation

Our research is motivated by the importance of diagrams, the availability of a large collection of diagrams, and a variety of excellent potential applications, e.g., Content-Based Diagram Retrieval (CBDR).

Diagrams are crucial parts of documents, particularly for scientific research papers, reports, and manuals. Diagrams convey information that is difficult or impossible to describe by text. Often, a reader can quickly gain an overview of a document by a glance at the diagrams included in a document without reading any text.

A large number of diagrams in electronic documents have become available in this digital era. One of the pioneer electronic publishers, BioMed Central, has published 64,999 articles in their 205 journals as of February 2010 [6]. Despite the fact that large numbers of diagrams have become available in electronic documents, no satisfactory research exists to analyze them from the perspective of diagram understanding and information retrieval [3, 18, 11].

The time has come to analyze diagrams in terms of their representation features, to understand diagrams, and to further categorize, cluster, and retrieve diagrams.

1.2 Objectives

The objective of this thesis is to gain insight of the diagrams through analyzing their features, the graphemes, and to prove that these features can be applied to further understanding and analysis of diagrams.

The five major diagram classes of our target are: Bar charts, Data curves, Data lines, Data
Figure 1.1: The five diagram classes studied in this thesis: Bar chart, Data curve diagram, Data line diagram, Data point diagram, and Tree diagram.
1.3 CHALLENGES

points, and Trees:

- **Bar Chart**: a $x, y$ data plot represented with a number of adjacent bars of the same width (e.g., Fig. 1.1(a)).

- **Data Curve diagram**: a $x, y$ data plot represented with data curves (e.g., Fig. 1.1(b)).

- **Data Line diagram**: a $x, y$ data plot represented with data lines (e.g., Fig. 1.1(c))

- **Data Point diagram**: a $x, y$ data plot represented with data points (e.g., Fig. 1.1(d))

- **Tree**: an arc-node structure consisting of nodes such as rectangles connected by directed or undirected lines (e.g., Fig. 1.1(e))

It is important to describe the difference between vector diagrams and raster images. Vector diagrams consist of geometrical primitives such as points, lines, curves, and shapes, in contrast to raster images that are represented with a rectangular array of gray level or color values. The difference in visual appearances of vector diagrams and raster images is illustrated in Fig. 1.2, where Fig. 1.2(a) is the original diagram, Fig. 1.2(b) is the magnified section of the raster format, and Fig. 1.2(c) is the magnified section of the vector format. It shows the advantage of vector format which can be scaled up to an arbitrary resolution without loss of visual quality. With the tendency for the vector format becoming the standard for diagrams in electronic articles, a large collection of vector diagrams is becoming available. Despite the fact that diagrams in raster format are the majority of the existing electronic documents, vector diagrams can be created from vectorization. Thus, the analysis and methods discussed in this thesis can be applied to raster diagrams once they are vectorized.

1.3 Challenges

Successful research on diagrams must address two major problems: Feature extraction regarding diagram representation and machine learning regarding diagram understanding.

Given the geometrical primitives in a vector diagram, how should we represent the content of the diagrams? Here are some of the options.
(a) The original diagram. The middle data point on the top data line is the section to be magnified.

(b) The magnified data point on the diagram of raster format.

(c) The magnified data point on the diagram in vector format.

Figure 1.2: A diagram and its magnified section of a data point in both vector and raster formats. (a) is the original diagram. (b) is the magnified data point of this diagram in raster format. (c) is the magnified data point of this diagram in vector format.
1.3. CHALLENGES

- We could attempt to do a full parse on the diagrams by analyzing the spatial relationships among the geometrical primitives, which is an advantage provided by the embedded precise geometrical information of the primitives.

- We can apply a “Bag of Features” approach to the diagrams by regarding each geometrical primitive analogue to a word in the text documents.

- We combine geometrical primitives and partial parses of diagrams to represent each diagram as a “Bag of Features”. A partial parse of a diagram is a combination of primitives that obey a certain geometrical constraints. This is our approach.

The second challenge lies in diagram understanding by applying machine learning techniques. We pursue machine learning attempting to answer the following questions.

- What classification algorithms are suitable for the diagrams in our research domain? Can the diagram classes be distinguished equally well? Are there any differences among the classifications of each individual diagram class?

- What clustering algorithms can help us determine the set of features to cluster the diagrams?

- What sets of features are capable of classifying individual diagram classes?

- What sets of features are capable of clustering diagram classes?

- What would the feature set achieve in diagram retrieval?

Current information retrieval systems can not be applied to vector diagram retrieval for they are either text based or raster image content based. Text based systems such as Google Image search diagrams by using the diagram file names or the HTML context text rather than the actual diagram content. Content based image retrieval (CBIR) systems aim at raster images such as photos of scenes and faces, instead of vector diagrams. Thus, a thorough and deep study of vector diagrams is indispensable for a reliable and successful content based diagram retrieval (CBDR) system.
1.4 Contributions of this thesis

We have extracted and analyzed the features that can characterize the content of diagrams. We demonstrate the power of the feature set in diagram understanding by retrieving, classifying, and clustering the diagrams.

- Content features for vector diagrams: Graphemes

We propose novel structural features, graphemes, that efficiently characterize the information embedded in diagrams. A grapheme is an elementary meaningful structure in a diagram. It may be a primitive such as a line segment or a simple n-gram such as a tick-mark together with its label in a data graph \( n = 2 \). A grapheme is both grammatical and semantic since it not only describes how a diagram element is structured but also gives evidence as to what this element is about, and what sense it might denote. In other words, they also contain more important information: geometric information that can be applied to analyze the spatial relations between different graphemes. For instance, a long horizontal line and a short vertical line construct a tick mark if they touch each other at right angles. This information content can be found in the graphemes.

The number of distinct structures that can be distinguished in published diagrams can be huge, depending on the complexity. Full structural analysis of diagrams is possible, as in [23], but getting reasonable coverage would demand writing thousands of constraint grammar components. At the other extreme, lie the primitives, with no structure taken into account. There are very few of these. In PDF there are only three primitives: rectangles, paths, and text. They alone are not enough to support classification of diagrams. The strategy of this research has been to split each collection of primitives into further subclasses, e.g., horizontal lines, curves, etc. But we have taken one further step, a step in the direction of constraint-based descriptions of structures. Our manual study of thousands of diagrams led us to choose a few simple combinations, primarily tick marks and small objects that typically serve as data points, such as disks and diamonds. To step beyond the grapheme classes we chose would open us to a flood of more complex structures. The results achieved in this thesis show that this was a wise decision, since our choice of graphemes led to excellent quantitative results in
the eighty to high ninety percent range, for the various measures we used.

The question one might ask is, ”Why these graphemes?” The answer lies in the data, the diagrams as produced by authors of scientific papers. Diagrams are based on conventions just as natural language is. Diagrams are cognitively generated and cognitively perceived. So our choice of graphemes is very much an empirical one. For this type of research, the proof is in the pudding. A large and exhaustive study might lead to more insight, but the results would still be of an empirical nature - they could hardly get to the cognitive roots of why humans produce and perceive diagrams in the ways that they do.

We have shown the graphemes defined in this thesis can successfully distinguish five rather distinct classes of diagrams. It is certainly possible to extend the techniques to subclasses of the five and to diagrams that are distinct from the five. The extension could be done systematically by defining additional graphemes using increasingly complex geometrical constraint specifications, guided by some manual, empirical research on the structure of additional diagram classes. The steady addition of more complex graphemes would lead to a higher level of ”diagram understanding”, beyond simple classification. For example, well-crafted specifications of error bars would add an to the analysis. The information content of diagrams in scientific papers, especially in the biomedical sciences, and the amount of space and discussion that revolve around diagrams make it clear that they carry substantial and critical types of information. A plot of temperature versus time could describe an exercise physiology experiment or global warming. But the form of the two diagrams is essentially the same. The interpretation of both of these examples requires, at a minimum, the types of analysis tools and techniques developed in this thesis.

- **Retrieval and Machine learning application**

We demonstrate the quality of the graphemes in applications including diagram retrieval, classification, and clustering. The most convincing results of our work come in our retrieval experiments in which the mean precisions at 10 reach as high as 98.8% in the five diagram classes. Binary classifications of each individual diagram class also prove that our grapheme sets successfully represent the diagrams. For instance, the recall value of the tree diagram
binary classification through the 10-fold cross validation is 93.6%. For each of the five diagram classes, we summarize a subset of the most powerful graphemes, in terms of binary classification. Our experiments show that boosting algorithms are the best match to vector diagrams when compared to decision tree and naive Bayes approaches.

1.5 Thesis outline

The structure of the thesis is as follows: Chapter 2 reviews the research fields of diagram and image analysis, aiming to clarify the differences and similarities between them, Chapter 3 explains important aspects of machine learning techniques applied in our research. Chapter 4 explores the data domain of our research, covering topics from characteristics of PDF files and data acquisition strategies. Chapter 5 introduces the novel content feature, the grapheme, that can represents the elementary components of vector diagrams. Chapter 6 reports our experiments on our dataset, and explains our analysis and discoveries. Chapter 7 summarizes the dissertation and describes future works that can be built on what we have done in this thesis. Chapter 8 explains the procedures and results of this thesis in a way that will allow other researchers to replicate our work.
In this chapter, we review three topics that are related to our thesis. Section 2.1 reviews the features that are extracted for image processing. Section 2.2 explains the general methods for the high level processing on diagrams. Section 2.3 describes one of the applications of diagram and image processing: information retrieval.

2.1 Feature extraction

Essential visual features need to be extracted to reduce the computation complexity of the approach. In raster image processing, visual features including color, shape, texture are extracted directly from the visual content of the image. These features are often called content features, though the term "content" does not indicate truly semantic content in those studies. As we show, features of high quality can improve the performance of the processing that follows, including object recognition and image retrieval.

Grapheme features, as discussed in detail in Chapter. 5, identify elementary meaningful structures of a vector diagram. For instance, a triangle grapheme consisting of three connected short line vector elements carries information of not only the three lines but also their spatial relationships. Thus, a grapheme, either basic(containing one element) or complex(containing at least 2 primitives), represents high level content as compares to low level features such as texture. A feature would be a $<attribute, value>$ set where value is the count of the occurrence of the attribute in a instance.

The most commonly extracted features are:

• Shape features characterize raster images using geometric properties, the most straightforward visual characteristic of an image. Two kinds of shape features are usually extracted:
boundary-based and region-based features. Classic boundary-based shape features, including critical points and contrast enhancement, extract only the boundaries of image regions and use the results in an attempt to represent the whole image. On the other hand, region-based features, such as moments, take all the pixels in a region into account and compute features to represent the entire regions [44, 27]. Shape features continue to attract researchers’ attention, so many new shape features have been devised. For instance, a novel shape feature, called shape context, has been proposed that records not only the group of critical points, e.g., the corners of the objects in the image, but also the context of each point. The context of a critical point is defined as the relations between this point and all other points. It is shown that so that shape context will improve the performance of similarity measure in the object recognition processing [4, 36].

- Texture features reflect the structural pattern arrangement in images, and they are often used in document image processing. Texture properties have been usually found through wavelet transform since it was first proposed in 1990s [53], and further progress in this area has been made such as the combination of texture feature extraction and distance metrics to improve the image retrieval rates [12]. A more extensive survey on texture features is discussed in [46].

- Color features represent images by extracting the (r,g,b) or more complicated vector values in the color space for each pixel in the images. One of the advantages of color features is that they are independent of orientation and transformations of the image that requires more computation. The basic representation of color features applied in image processing are color histograms and color moments [55]. Color features are widely used in the context of photography and painting retrieval and trademark retrieval.

In our thesis, we define graphemes to combine both low-level and high-level characteristics in a manner that each vector element in the diagram contain all the information of its location and graphics states but the spatial relationships of the vector elements can be studied to discover their semantic meaning.

There is no single feature that can best abstract information of an image, instead, most of the research on image processing include many features. For example, an image search engine called
Virage combines color, texture, and object boundary features together and supports arbitrary weight assignments to each of these three features [2]. Our approach also defines and extracts many features, but of a quite different nature.

2.2 Object recognition and categorization

One of the high level processing of the features is to recognize and categorize objects by analyzing the features. For instance, cars can be recognized by studying the features extracted from the photos of cars. The goal of object recognition and categorization, from the view of pattern recognition, can be achieved in the following approaches: template matching, syntactic approaches and statistical approaches.

In our research, we apply machine learning techniques to graphemes, to classify diagrams using the “Bag of features” approach. Similar to “bag of words” in document classification and text retrieval, “bag of features” in computer vision has been proven that a collection of independent features helps computers to find patterns hidden in the content of text and images. “Geometry free” bag-of-features models, which represents an image as a collection of local features, have received a lot of attention due to their simplicity by disregarding all information about the spatial layout of the features [49, 38].

- Template Matching is one of the simplest matching approaches that usually recognize an image by matching it with the pattern images in terms of similarity measure.

The similarity, i.e., distance, between the query and the pattern images are computed based on some feature set. The formulation of similarity measurement depends on the type of the features. If the features are about intensity such as color and texture, the distance metric such as the Euclidean distance is often applied. Intersection distance proposed by [56] is also used for color histograms. If the features are geometry based such as shape, shape matching algorithms such as geometric hashing [61] are then required. Other techniques are also integrated to improve the matching performance, for instance, in the well-known QBIC [17] system that defines similarity as a distance metrics in high dimensional feature space, a filter is applied to selecting only a subset of images for the similarity metric, and
R-tree is used to reduce dimension for the index of the images.

- The syntactic approach represents images in a hierarchical syntactic structure due to the fact that the images (patterns) can be viewed as a composition of sub-patterns. The elementary sub-pattern is called primitive, and the hierarchy structure of the pattern and the relations between sub-patterns (and primitives) can be described using logical description such as grammar, or structural descriptor such as spatial constraints. Thus the whole pattern can be illustrated by a group of primitives and the descriptions. The matching proceeds by comparing the primitives and the descriptions of the pattern and the query images.

The reasoning mechanisms on the structural relations between objects in images is applied to accomplish object recognition in [25]. A syntax describing “segmented regions as basic objects and complex objects as composition of basic ones”, and the similarity measurement based on the syntax are reported in [50].

- The statistical approach is emerging as an active trend in image categorization where various classification and machine learning algorithms are applied to achieve promising performance. Researchers attempted to group images into semantical categories by building hierarchical index on visual features such color and texture. A promising supervised classification method applies a binary Bayesian classifier to categorize vacation photographs into a hierarchy of high level concept classes based on low level visual features of 1st and 2nd color moments in LUV color space [58]. Eleven semantic concepts such as indoor and outdoor images are categorized among 171 vacation images. An unsupervised clustering approach is reported in [9], which group all the images similar with the query image together based on a fuzzy visual feature set consisting of color, texture, and shape properties. Besides, relevance feedback can also be applied in the query modification mechanism to improve the semantic categorization. [48] makes use of relevance feedback to dynamically modify the image database. However, one drawback of the relevance feedback method is that it usually requires the feedback from a human which makes it almost impossible for very large image collection.

Some publications report the combination of template matching and statistical approach. For instance, similarity measurement and classification are combined in [36, 4]. The similarity mea-
measurement is preceded by applying the correspondence between edge points on the query and pattern images to estimate an aligning transform. The Nearest Neighbor classifier looks for the stored pattern of the maximum similarity with the query image. A region based similarity measurement improves the robustness of the recognition, categorization, and retrieval, and the cluster technique make the retrieval more efficient by group the similar images together [8].

2.3 Content Based Image Retrieval (CBIR)

Pursuing similar goals as ours, e.g., image retrieval, Content-Based Image Retrieval (CBIR) aims to manage, organize, navigate, and retrieve images and videos, using machine learning technology as its support. Its research covers topics such as feature extraction, indexing, machine learning, query processing, and feedback [34, 41, 42, 43]. The features include both low-level features such as color and higher-level ones such as texture. Indexing is studied to reduce the number of dimensions of the feature space. For query processing, efficiency and optimization are crucial, especially if the queries are processed online. We review several well-known CBIR systems in this section.

- QBIC

As the first commercial CBIR system, the classic QBIC (Query Based on Image Content) [17] developed a prototype system that has influenced the many CBIR systems that followed. The content features extracted are color (color percentage and color layout) and texture. The image database is indexed with R*Tree to reduce dimensions of the feature space. Besides color and texture, text-based queries can also be processed to promote its query processing ability. Since proposed in 1995, QBIC has been applied to many real systems, including the website for The State Hermitage Museum of Russia that is voted the best in Russia [39].

- Google Image

“Google Image Search is a search service created by Google which allows users to search the Web for image content. The feature was originally announced in December 2001. The keywords for the image search are based on the file name of the image, the link text pointing to the image, and text adjacent to the image.”
• ImageRover

ImageRover builds index on both text features and image content features to search on a WWW image database. Textual features are extracted from URLs of the images. Color and texture are extracted as image content features. The initial search is based on a keyword, then users can refine the query through relevance feedback [30].

• Figurome

Figurome shares the similar goal with our research: retrieval diagrams in the articles. Like us, Figurome chooses BioMed Central as data repository owing to its rich collection of articles and diagrams. It labels the figures to Gel, Pathway, Structure, and Time, based on the figure’s caption, text embedded within the figures, and other image properties. The query consisting of text keywords such as “cancer” retrieves a set of figures from BioMed Central [47].

• ImgSeek

ImgSeek [5] focused on the problem of finding the top-K similar images for a query. The core image similarity algorithm they use is based on a multi-resolution wavelet decomposition approach.

• Viper

In Viper [37], the features extracted are color and texture. Their image database contains as many as 80,000 features. The inverted files (a data structure mapping from content features to its location in the image database) are created on these features so that once the system gets a query, they will read through the inverted file to find the matching images. Viper also supports query by example.

CBIR systems tend to concentrate more on indexing, organizing and browsing, rather than on the content features themselves. For instance, the CBIR researchers do not pay much attention to the problem of which features are better in representing the images and why they are better, instead they attempt to organize the index of the features so that a query can be processed efficiently. Although at the current stage of our research, we do not explore the issues related to CBIR, we keep them in mind so that the features we define and analyze can be applied to the image retrieval systems in the future work.
Machine learning technology plays a critical role in our research, since it dominates the performance of the diagram analysis and categorization [35, 28]. Machine learning allows computers to learn from data and then to produce good classifications of new data. What machine learning systems do is to generate a hypothesis as to what the structure of a collection of objects is. New, previously unseen objects are then tested against the hypothesis. An example is a decision tree. Once constructed, a decision tree embodies a hypothesis. An object is tested against the hypothesis by following the tree to the class value of a leaf node.

- Supervised learning

We have used supervised learning in most of our work reported here. Supervised learning requires two components. One is the descriptive data itself in the form of feature set values. The other is labels furnished by an evaluator, typically a knowledgeable human - a time-consuming process for large data sets. When applied to a new, previously unseen object, the output may be either continuous value for a regression problem, or one or more labels for a classification problem. If more than one label can be assigned to a single instance, it is called a multi-class classification problem.

For machine learning to succeed, a sufficient number of training instances and their feature sets are required. The choice of and the quality of the features are the major determinants of the accuracy of the induced function, the hypothesis. Each machine learning algorithm generates hypotheses of a specific nature tied to that hypothesis. Numerous supervised learning algorithms [10, 31, 19] have been proposed such as Neural networks, Bayesian classifiers, Decision trees, Support vector machines, and Boosting. Since every algorithm has strengths and weaknesses, there does not exist an algorithm that matches all kinds of data and domains.
CHAPTER 3. MACHINE LEARNING

The suitable match and optimal performance must be decided by experiments that include studying the domain, choosing the algorithms, adjusting parameters, and most importantly, studying the results [35].

To evaluate the hypothesis, testing data and testing methods are applied. The most common method is Cross Validation (CV) that splits the data into subsets so that some subsets are used in training and others are used in testing respectively in a set of iterations. K-fold cross-validation and random sub-sampling validation are two types of cross-validation. For instance, a 10-fold cross-validation splits the data set into 10 subsets and iterates 10 times. In each iteration, 9 of the 10 subsets are used as training data and the remaining subset is testing data, so that each subset is tested once [32].

• Unsupervised learning

We have also been using unsupervised learning in our studies. Unsupervised learning works with unlabeled data, grouping instances into classes, each member of which is similar to the others, but dissimilar to instances in other clusters. Unsupervised learning is free from human labeling biases, so it can be useful in discovering the "natural" structure of the collection. The similarity between instances is quantified by a Distance Metric (DM) that are computed in the n-dimensional space of feature set vectors. There are many choices for the DM such as the Euclidean and Manhattan distances. Depending on the property of the data, hierarchical or partitional clustering algorithms can be chosen. Hierarchical clustering will produce clusters in a multi-level fashion, bottom-up; partitional clustering separates data into several independent clusters, top-down [33, 13].

• Semi-supervised learning

Semi-supervised learning attempts to combine the advantages of both supervised and unsupervised learning, i.e., uses both labeled and unlabeled data in training. Although we have not yet used them in our research, semi-supervised learning technologies may be helpful when our work is scaled up at some future time. The two largest categories of machine learning algorithms, supervised and unsupervised algorithms have both advantages and disadvantages. On one hand, supervised learning requires labeling a large amount of data, on the other hand,
unsupervised learning simply splits data into clusters based on the nature of the data and the feature sets. With a small portion of training data labeled and the rest unlabeled, research in semi-supervised learning shows that this combination can sometimes promote the efficiency of training [7, 62].

3.1 Supervised learning

3.1.1 Decision trees

The decision tree is one of the most popular supervised classification algorithms and has been applied to a large variety of domains and tasks. Decision trees are designed to generate a hypothesis by induction from training data, and representing the hypothesis in the form of either a tree or a set of if – then rules. The attributes and class label are typically discrete-valued, but can extend to continuous-values [45].

The decision tree algorithm is explained below. To simplify the description, we use discrete values here.

**Algorithm 1** Decision Tree

**Input** Each training instance is composed of a set of discrete attribute values and discrete-valued label. Taking an example of the well-known contact lens data set, a training instance can be described as

\{< age, 30 >, < nearsight, 0.5 >, ..., < contactlens, YES >\}

**Output** The hypothesis is a tree structure. Each node of the tree except the leaf nodes represents an attribute, and each branch out of this node represents a value of this attribute. The leaf node indicates one of the class label values.

To find the class for a query instance, the tree is traversed starting at the root node, along the path from the root to the leaves, following the branches whose values match the corresponding attribute values of the query instance, until a leaf node is reached. The value of the leaf node is the class label of the query instance predicted by the decision tree.

The training stage of decision tree algorithm is the procedure of building the decision tree. The classical decision tree algorithm, C4.5, builds the decision tree by using the concept of information gain that evaluates the information provided by each attribute in classifying the training instances.
The information gain of each attribute is defined as a function of entropy:

\[
Gain(S, A) = Entropy(S) - \sum_{a \in A} \frac{|S_a|}{|S|} Entropy(S_a),
\]

where \( S \) is the training set, \( A \) is an attribute, \( S_a \) is the subset of training data whose value of attribute \( A \) is \( a \). Entropy is defined as

\[
Entropy(S) = -\sum_{l \in L} P_l \log_2 P_l,
\]

where \( P_l \) is the partition of the instances in \( S \) with the value \( l \) of the class label set \( L \), indicating the information needed to classify the training set \( S \). \( Entropy(S_a) \) represents the information needed to classify the subset \( S_a \) whose value of attribute \( A \) is \( a \). \( \sum_{a \in A} \frac{|S_a|}{|S|} Entropy(S_a) \) shows the sum of the normalized entropy needed to classify the training set given the attribute \( A \), thus \( Gain(S, A) \) represents the reduction of entropy for the training procedure provided attribute \( A \).

The more distinguishable the attribute \( A \) in classifying the training instances, the smaller the \( \sum_{a \in A} \frac{|S_a|}{|S|} Entropy(S_a) \), thus the larger the information gain \( Gain(S, A) \). Information gain can therefore be used as the criterion to find the most distinguishing attribute.

At the beginning of building the tree, the attribute of the largest information gain is selected as the root node of the resulting decision tree, i.e., decision tree is a greedy algorithm. For instance, the 3rd attribute in attribute space \( V \) is selected, we call it \( A_{13} \) in which \( 1 \) means the attribute is on the root node on the decision tree. Branches are created on node \( A_{13} \), each representing a discrete value of attribute \( A_{13} \). The whole training set is then split into these branches to be consistent with the branches, i.e., instances of value \( v1 \) of attribute \( A_{13} \) are assigned into branch \( v1 \). On each branch, another attribute will be selected so that it classifies the training instances in this branch by consuming the least information, i.e., gain the information to the largest scale. Say, the fifth attribute is selected as it contributes the largest information gain on this branch, a node called \( A_{25} \) is thus created for this attribute, the training subset on this branch is next split into further branches based on their values of attribute \( A_{25} \). The tree continues to grow until the instances in each branch are of the same class label.

On the other side, since the tree is built to be consistent with all and each of the training instances, overfitting is unavoidable, i.e., the tree fits the training instances so perfectly that it loses its generality to explain the rest of the domain. The more precise the tree explains the training data, the less accurate the tree predicts the query data, i.e., a greedy algorithm, with no
backtracking. This is overcome by pruning the result decision tree to generalize the tree without sacrificing the training accuracy.

The decision tree algorithm is robust to noisy training data since at each step of selecting the best attribute to split the training set, the whole set of training data in the current sub-tree is considered in a statistical fashion, so that the errors in some specific instances can hardly affect the tree accuracy given that the training set is sufficiently large.

Another property of the decision tree algorithm is that the tree is built to achieve the local optimal classification because first, it selects a single best attribute at one moment without looking back at the sub-tree that has been constructed, i.e., no backtracking. Second, only a single tree is built despite of the fact that there may exist multiple trees that fit the training set.

### 3.1.2 Boosting

We have studied and applied Boosting algorithms for machine learning. Boosting is a meta-algorithm of supervised learning. The basic idea of Boosting is that a simple learning algorithm, called base learner or weak learner, whose correct learning rate is slightly higher than random guessing, can be boosted to be a powerful learner by an iteration-updating mechanism. The base learner can be as simple as a decision stump (a one level decision tree), or as advanced as any supervised learner. Initially, all the instances are assigned weights that are the same for each instance. After each learning iteration, a hypothesis function is learned. According to the hypothesis, all the training instances are evaluated so that the weights of the mis-labeled instances are increased and the weights of those correctly-labeled are decreased, so that the instances with high weights will paid more attention in the next iteration. At the end of the training, usually after 10 iterations, the final hypothesis will be built to be the maximal weighted sum of the hypotheses created in all the iterations (Algorithm. 2.).

With the ultimate goal of categorizing not only simple diagrams such as bar diagram but also multi-labeled diagrams that draw one diagram on the top of another, such as a line graph on the top of a bar chart, we need to utilize multi-label boosting algorithms. AdaBoost.M1 is believed historically to be the first boosting algorithm for multi-label classification, i.e., a single instance may be assigned more than one label but with different weights.
Algorithm 2 AdaBoost.M1

<table>
<thead>
<tr>
<th>Input</th>
<th>1. Sequence of $m$ examples $(x_1, y_1), ..., (x_m, y_m)$ with labels $y_i \in Y = {-1, +1}$</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>2. distribution $D$ over the examples</td>
</tr>
<tr>
<td></td>
<td>3. base learning algorithm <strong>WeakLearn</strong></td>
</tr>
<tr>
<td></td>
<td>4. integer $T$ specifying number of iterations</td>
</tr>
<tr>
<td></td>
<td><strong>Initialize</strong> the weight vector: $D_1(i) = 1/m$ for $i = 1, ..., m$</td>
</tr>
<tr>
<td>for all $t = 1, 2, ..., T$ do</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1. Train <strong>WeakLearn</strong>, providing it with the distribution $D_t$; get back a hypothesis</td>
</tr>
<tr>
<td></td>
<td>$h_t : X \times Y \rightarrow [-1, +1]$.</td>
</tr>
<tr>
<td></td>
<td>2. Calculate the error rate of $h_t$: $\epsilon_t = Pr_{i \sim D_t}[h_t(x_i) \neq y_i]$.</td>
</tr>
<tr>
<td></td>
<td>3. Set $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$.</td>
</tr>
<tr>
<td></td>
<td>4. Update: $D_{t+1}(i) = \frac{D_t(i)}{Z_t} \times \left{ \begin{array}{ll} e^{-\alpha_t} &amp; \text{if } h_t(x_i) = y_i \ e^{\alpha_t} &amp; \text{if } h_t(x_i) \neq y_i \end{array} \right.$</td>
</tr>
<tr>
<td></td>
<td>$Z_t$ is a normalization factor</td>
</tr>
<tr>
<td>end for</td>
<td></td>
</tr>
<tr>
<td></td>
<td><strong>Ensure</strong>: the final hypothesis is $H(x) = \text{sign}(\sum_{t=1}^{T} \alpha_t h_t(x))$.</td>
</tr>
</tbody>
</table>

### 3.2 Unsupervised learning

Unsupervised learning focuses on learning from data without class labels. Concepts can be summarized from the unsupervised learning experiments. Groups can also be constructed to split the data set. The former is conceptual learning that aims to build a hierarchical descriptions of concepts explaining the data, while the latter is clustering. We’ll focuses on two clustering algorithms that we apply to our data.

#### 3.2.1 K Means

1. Initialize $K$ clusters with $K$ instances, compute the distance from each instance to these $K$ centers, assign each instance to the cluster with the least distance.

2. Compute the center of each instance based on the current assignment.

3. Repeat the procedure of computing distances from instances to the $K$ centers, re-assign the instances to the clusters, until the cluster centers remain stable in consecutive rounds.
3.2.2 EM

Assuming each cluster obeys a probabilistic distribution with latent variables, Expectation Maximization (EM) aims to find the maximum likelihood estimates of the parameters for the latent variables. EM iterates between an expectation (E) step and a maximization (M) step, until the expected total log likelihood of the data set fitting the distribution stops increasing. The final parameters are then used to determine the distribution of the latent parameters.

After initializing parameters for these distribution, E and M steps are performed in iterations as explained below, until the total log likelihood of data set matching the distribution reaches maximum.

1. Expectation (E) step calculates the probabilities of each instance belonging to the distribution of each clusters, assuming the instances obey the current parameters of the distribution. The expected value is calculated based on these probabilities.

2. Maximization (M) step calculates a new maximum likelihood probability distribution with the assumption that the expected value calculated in E step fits that of the hidden variables.
This chapter first introduces the source of our corpus of diagrams in Section 4.1. Then the characteristics of PDF files are introduced in Section 4.2. The preparation of the corpus is discussed in three stages Section 4.3 to Section 4.5.

The first stage, diagram acquisition extracts the graphics and text primitives corresponding to vector diagrams from the PDF source code. A PDF document is a data structure composed of objects of the the documents, the structure to access these objects, the structure of each PDF page, and the content streams that render the text and graphics on the documents. The commands in the content streams for graphics are vector commands. The fact that makes the extraction complicated is that the order of the commands for both diagram and text on a PDF page has little to do with their physical locations on the page. To escape the intricacies of the original PDF file, heuristics are used to locate the content on each page. An interpreter is then constructed to translate the PDF command sequence into a set of self-contained graphics and text objects in Java. A spatial index is next constructed to store these self-contained objects so that their geometrical relations can be analyzed. The three stages are listed in Fig. 4.1.
4.1 Data source – BioMed Central articles in PDF format

There is no dataset for diagrams available for machine learning research, like the ones in standard collections such as the popular UCI machine learning repository\[57\], thus, we need to build our dataset from the ground up. We downloaded a large number of vector diagrams from BioMed Central (BMC). BMC is an electronic publisher of scientific journals whose articles in PDF format contain vector diagrams. We have chosen life science articles since first, life science articles rely heavily on diagrams in presenting the points of their articles. Second, some top electronic publishers support the Open Access policy that enables researchers to access articles from these publisher’s websites without subscription. They are mostly science publishers such as. Among the primary publishers such as Public Library of Science \[40\] and BMC \[6\], we chose BMC that publishes 205 open access journals as of February 2010. The most distinguished characteristic of BMC is that the articles in their 71 BMC-series journals tend to support vector diagrams, which provides us an excellent data collection for our research.

We downloaded about 20,000 articles from BMC website that were published in the past 10 years. We found that about one third of them contain vector diagrams. The rest of the collection are reviews and supplements that seldom contain diagrams, or articles with only tables and images to represent their data. For the articles that contain diagrams, we discover that each article has 3
diagrams on average, which produced 18,423 diagrams in total.

4.2 Characteristics of PDF files and their vector diagrams

A PDF document is composed of a number of pages and their supporting resources (Fig. 4.2). Both pages and resources are numbered objects. A PDF page contains a resource dictionary and at least one content stream. The resource dictionary keeps a list of pairs of a resource object number and reference name. A resource object may be a font, graphics state, color space, etc. Once the resource objects are defined, they can be referenced within any page in one PDF file using a reference name.

The sequential content streams define the appearance of PDF documents. They are the most essential parts of PDF since they are the parts that utilize resources to render text and graphics, etc. A content stream consists of a sequence of instructions for text and graphics. Text instructions include text rendering instructions and text state instructions. Text rendering instructions write text on a page. Text state instructions specify how and where text will be rendered to a page, such as location, transform matrix, character space, text rise, size, color, etc.

Graphics instructions include graphics rendering instructions and graphics state instructions. Graphics rendering instructions draw graphics primitives such as lines, rectangles, and curves. Graphics states instructions specify the width, color, join style, painting pattern, clipping, transforms, etc. Graphics states can be specified either in internal graphics state instructions or in referenced external graphics state objects. PDF also provides a graphics state stack so that local graphics states can be pushed or popped to change the graphics state temporarily and then return to a previous state.
Figure 4.2: A simplified PDF structure example. A PDF file is composed of pages and resources such as font, graphics state and color space. Both page and resources are defined as objects with sequence number. In this example, page 1 is object #80, and font 1 is object #10. These sequence numbers are used as reference numbers when the object is referenced in another object. In this example, object #10 that defines a font is referenced in a page (object #80) object’s resource dictionary as “Ft1 10” in which 10 is the font object’s sequence number. Once the resource objects are defined, they are globally available, i.e., they can be referenced by any pages in the same PDF file. For instance, object #20 is referenced by two page objects: object #80 and #90. For a general and brief description of PDF structure, see [1, 26].
4.3 Acquisition of vector Diagrams from PDF articles

The difficulties of extracting the vector diagrams are threefold. First, the PDF content stream is a sequential list of instructions. The sequence is important because the sequence of resources (graphics states and text states) defines the local environment in which the graphics and text are rendered. The values of resources can be changed in the sequence, and the change affects only the instructions that follow the change and before the next change. This property makes extraction complicated because if we want to extract either graphics primitives or text inside graphics with all of their related state parameters, we need to look back through the instruction sequence to find the last values of all the parameters needed.

Second, despite the fact that the content stream is sequential, the instruction sequence in the content stream is not necessarily in accord with their positions on the page. In fact, the content stream instruction sequence and positioning on a page are totally different issues in PDF. Moreover, a PDF document may apply different strategies to write content streams to produce the same appearance, though their instructions may be arranged in different orders. This property also makes extraction difficult. We can’t directly apply content stream position information to help extraction. An example of content stream of a PDF page is illustrated in Fig. 4.3.

Figure 4.3: An illustration of the content stream sequence of a PDF page. The headnote and footnote of the page is first rendered. The body text is followed by the graphics rendering. Different sequences may be produced by different PDF creation templates. This example shows a BMC template.
The third complication comes from the fact that different authors draw diagrams in different ways. For instance, author A may like to draw an arrow as a small triangle at the end of a thin rectangle, while author B may prefer the arrow head composed of two short lines with a small angle between them, and the arrow tail being simply a thin long line. All these practical factors affect the vector commands of the diagrams in the articles, though the diagrams may look essentially the same.

We translate PDF documents into corresponding Java objects by applying the open source package, Etymon PJX [14]. We then translate the Etymon Java objects into independent Java objects that free us from the Etymon system. For a PDF document, we manipulate the Java objects representing pages, resources, fonts, graphics states, content streams, etc, to locate and extract those that draw diagrams, along with their supporting resources. Details of the extraction procedures are explained below.

4.3.1 Extraction of diagrams

To extract figures, we devised a selection strategy to decide whether a page has figures or not. The selection strategy is based on graphics primitive statistics for each page. Since some PDF pages only contain pure text or a few simple figures such as tables, which are not of interest, we can apply the statistics of line primitives to eliminate such a page — if a page has only a few line primitives, then this page does not contain any figure we need to extract. If there are more than a certain number non-line primitives such as curves or rectangles, we can conclude that this page must contain one or more figures. If there are neither curves nor rectangles in a page, we can still conclude that a PDF page has figures if the count of line primitives is large enough. Experiments led us to choose a threshold of 60 determine if the PDF page has vector diagrams.

Once we conclude that the graphics in a PDF page contains figure material, we extract both graphics rendering instructions and their supporting graphics states. Graphics states can be specified in either the content stream or in separate objects. Graphics state instructions in content streams can be easily extracted as normal instructions, while graphics states in separate objects are extracted with the help of reference and resource dictionary. We first read the reference instructions of these graphics state objects and get their reference names, go to resource dictionary to
find the object sequence numbers of these reference names, and then access and extract the actual graphics state objects using the sequence numbers.

4.3.2 Extraction of text embedded within diagrams

Though text in diagrams has yet been used in the thesis, the resulting data will be useful in future studies. For example, the numerical values attached to tick marks, and the label strings for the x and y axes of a data graph can both be used in diagram retrieval systems.

The sequence of text and graphics instructions is not necessarily in accord with the sequence in the rendered page, thus it is difficult to decide which part of text instructions in content stream renders the text inside of graphics. Our study shows that the majority of the BMC articles in PDF format occupies a standard Adobe FrameMaker template to write content stream.

Usually the text rendering instructions start with a font reference instruction so that the text follows will utilize until another font reference is specified. If the text following the graphics instructions does not start with a font reference, its font must have been declared somewhere before the graphics. This fact requires us to keep the last font reference when we go through the content stream so that once we get the text we need, we can immediately use the last font reference to get the correct font. Given the font reference, we look it up in the resource dictionary to get the object sequence number of this font, then access and extract the font definition object.

We also extract URLs of diagrams from the text of PDF files since they provide us a convenient access to not only the diagrams in JPEG format but the original PDF files. The diagrams in JPEG format will be used as a comparison and prototype of the vector diagrams we extracted.

4.4 Transformation to self-contained objects

We implement an interpreter to transform the Java objects created for PDF commands to enable Java objects to contain two important kinds of information: the sequence of PDF commands and the dependency among commands.

First, the sequence of graphics primitives and text of PDF is crucial since it determines which command is rendered first, which might occlude other, which graphics state applied to a graphics primitive, etc. However, the results of the extraction step, i.e., Java objects of graphics/text drawing
instructions and graphics/text, can not preserve the sequence of instructions. Thus, we store the extracted Java objects as a sequence, mirroring the PDF content stream.

A portion of PDF code of a triangle and a rectangle is shown in Fig. 4.4. The first five commands draw the triangle\((gs\) defines the graphics states, \(m\) and \(l\) draw the triangle, and \(s\) strokes the triangle\), and the last three draw the rectangle\((rg\) define the color, \(re\) draw the rectangle, and \(f\) fills the area\). Eight Java objects created for these commands are independent of each other, such that the object for drawing command \(m\) does not know which graphics states will be applied to the graphics to be drawn. The Java object for stroke command \(s\) does not know which graphics to stroke.

![Figure 4.4: A section of PDF source code of a triangle and rectangle. The first five commands draw the triangle. The last three draw and fill the rectangle.](image)

Second, PDF rendering instructions usually depend on the local environment defined by state instructions. For instance, the triangle in the above example depends on its graphics state object defined by command \(gs\). Similarly, a text rendering object depends on its font definition object. In principle, the entire preceding content stream must be read to get the state parameters needed for a graphics primitive.

In PDF, the graphic state stack is used to temporarily save the local graphics state so that it will not affect the environment that follows it. We deal with this problem by implementing a stack in our interpreter to simulate the PDF state stack so that the local graphics state and the pushed prior state(s) are preserved. Then every self-contained object, no matter how its graphics state is defined: internal graphics state instructions, external graphics state object, or graphics state stack, references the correct state.
4.5 Building the spatial index

An important technique to deal with the large collection of the self-contained Java objects in a
 diagram is the spatial index that is a coarse 2D-array of cells isomorphic to the 2D metric space
of a diagram. Each cell contains references to all graphics primitives that occupy or pass through
the cell. Each cell also records the position of each primitive in the drawing sequence in order to
faithfully represent occlusions that can occur accidentally or by design [54]..

The spatial index provides an efficient way to deal with spatial relations among graphics primi-
tives, and enables us to deal with various graphics objects such as lines, curves, and text in a single
uniform representation. For example, the touch() predicate for two primitives simply checks to see
if the intersection of the two sets of cells occupied by the primitives is non-empty.

Fig. 4.5 shows a section of a spatial index of a previous diagram example in Fig. 1.2(c). The
cell (3,3) of this spatial index contains two lines which draw the last tick mark on X-axis. The
cells (1,1), (1,2), (2,1), (2,2),(3,1), and (3,2) contains the rectangle which is the last bar in the bar
chart.

![Figure 4.5: A section of spatial index where a rectangle, a tick mark, and a horizon-
tal line can be found. The cell (3,3) contains a tick-mark consisting of two graphics
primitives.](image)

Besides accommodating the graphics primitives and text, the spatial index also provides an
efficient way to deal with spatial relations among graphics primitives, which inspires our definition
of the novel content feature, grapheme. We will introduce grapheme in Chapter 5.
This chapter introduces the novel content feature we have developed and exploited in this thesis: the grapheme. We first introduce the definition of grapheme, grapheme extraction strategies, then discuss the representation and pre-processing of graphemes in a feature set. An example of a diagram and its grapheme representations are illustrated at the end of this chapter in Table 5.2.

5.1 Definition of grapheme

A grapheme is an elementary structure in a diagram made up of vector graphics objects that satisfy certain unary or \( n \)-ary spatial constraints (e.g., a horizontal line) or the three boundary lines of a triangle.

A grapheme is an elementary meaningful structure in a diagram.

One of the advantages of graphemes is that they enable us to extract the useful element of a diagram without attempting a full parse of the diagrams. In this sense, we can think of grapheme as a partial parse of a diagram. Moreover, using a collection of graphemes to characterize an entire diagram is analogous to the "bag of words" approach which has been applied successfully in text document categorization and retrieval. Similarly, the "bag of words" approach has been recently applied to raster image object recognition where images are studied as a large collection of small regions, without considering their spatial relations [15, 16].

Some examples of graphemes are shown in Fig. 5.1. We have extracted 24 grapheme to represent the content elements of diagrams. Both the 12 primitive and 12 complex graphemes
Figure 5.1: Some grapheme examples: vertical tick, horizontal tick, line, curve, adjacent rectangles, bars, arrow, and data point are listed below. Each grapheme is represented as a

\(<\text{attribute, value}>\)

pair where value is the occurrence count of the grapheme in a diagram.

- \(<\text{Rectangle}, \text{number of rectangles drawn by command } re>\)
- \(<\text{Path}, \text{number of commands in a complete sequence for a path}>,\)
  A command may draw a line or a curve. A path ends with either stroke or fill operation. For example, the triangle drawn by a path in Fig. 4.4 consists of three drawing commands \((m, l, \text{and } l)\), and a stroke command \((s)\). The value of Path grapheme for this triangle is three.
- \(<\text{Horizontal line element}, \text{number of horizontal lines in a path}>,\)
  In Fig. 4.4, the value of horizontal line element grapheme is one.
- \(<\text{Vertical line element}, \text{number of vertical lines in a path}>,\)
  In Fig. 4.4, the value of vertical line element grapheme is one.
- \(<\text{Diagonal line element}, \text{number of diagonal line elements in a path}>,\)
  In Fig. 4.4, the value of diagonal line element grapheme is one.
5.1. DEFINITION OF GRAPHEME

- **<Curve element>, number of curves in a path>**

- **<Diagonal data line of the same length on X axis>,**
  number of line elements with same projection on X axis
  
  This grapheme is extracted due to the fact that the data lines in a data plot usually
  have the same length on their projections on X axis. The data lines are not necessarily
  in the same Path.

- **<Diagonal data line of the same length on Y axis>,**
  number of line elements with same projection on Y axis

- **<Data line on both X and Y axes>, sum of the above two graphemes>**

- **<Text>, number of text objects in a diagram>**
  
  A text object is specified with PDF commands $Tj$ or $TJ$.

- **<Horizontal text>, total length of the text in horizontal orientation>**
  
  The length is calculated in the number of characters.

- **<Vertical text>, total length of the text in vertical orientation>**

The complex graphemes are:

- **<Horizontal tick mark>, number of horizontal tick marks>**
  
  A horizontal tick mark consists of a short horizontal line and a long vertical line (usually
  Y axis).

- **<Vertical tick mark>, number of vertical tick marks>**
  
  A vertical tick mark consists of a short vertical line and a long horizontal line (usually
  X axis).

- **<Total tick mark>, sum value of the above two graphemes>**

- **<Rectangles of the same area>, number of rectangles in a diagram that are of the same area>**
• <Bars with the same width>,
  number of rectangles that are of the same width >
  We assume these rectangles are the bars in bar charts.

• <Bars with the same height>,
  number of rectangles that are of the same height >
  We assume these rectangles are the bars in a bar chart after a 90 degree rotation.

• <Arrow, number of arrow graphemes >
  An arrow is composed of either one long line touches two short lines in a certain angle,
  or a line touching a triangle.

• <Circle data point, number of circle data points >
  A circle data point is a circle whose area is smaller than a pre-determined parameter.

• <Triangle data point, number of triangle data points >
  A triangle data point is a triangle whose area is smaller than a pre-determined parameter.

• <Diamond data point, number of diamond data points >
  A diamond data point is a closed area composed of four short lines.

• <Rectangle data point, number of rectangle data points >
  A Rectangle data point is a rectangle whose area is smaller than a pre-determined parameter.

• <Total graphics and text objects, number of graphics objects and text objects >
  A graphics object is a closed path or a rectangle drawn by \texttt{re}. A text object is a text
  rendering command \texttt{Tj} or \texttt{TJ}.

5.2 Extraction strategies

The essence of grapheme extraction is the spatial constraints among the graphics objects.
Spatial constraints define the relationship of the elements in a grapheme \cite{20, 21, 22, 23}. 

The identification of graphemes is made possible by the data structure of spatial index. A spatial index provides two views of the diagram: a global view that considers the complete diagram in the space and a local view that focuses on the individual cells and the elements of the diagram that are rendered in these individual cells.

The global view helps us identify the basic graphemes such as the number of line elements since the basic graphemes are independent from each other and identifying them has nothing to do with the individual cells that the elements are rendered in. Moreover, the identification of the basic graphemes in the global view is not affected by the resolution of the spatial index, which is not the case for local view.

The local view, i.e., a subset of connected cells, enables us to examine the spatial relations among the elements that are near, touch or cross each other, since each cell records all the elements that are rendered in this cell. The local view of spatial index is the platform where the complex graphemes are identified since only in the local set of individual connected cells can the spatial relations among elements of the diagram be analyzed efficiently. To look for a tick mark, for instance, we only need to search through the limited number of cells that include long lines and short lines that are perpendicular with each other, without the necessity of considering all the other cells. Without a spatial index, we have to look through the complete list of elements in the diagram, and study the geometrical relation between each pair of them to find a tick mark.

The extraction of graphemes is made feasible and efficient by the spatial index that simulates a PDF page so that the spatial relations among the graphics objects can be studied. We define a set of geometrical constraints, either unitary or arbitrary to represent the spatial relations. A unitary constraint such as horizontal($l_1$) determines a horizontal primitive grapheme, and an arbitrary constraint such as connect($l_1, l_2$) identifies a complex grapheme consisting of two connected line elements. Thus a grapheme can be described as a tuple of primitives, often just a pair, that obey geometrical constraints. For example, the Vertical Tick grapheme, i.e., the one on the upper-right corner in Fig. 5.1, can be described as a pair of lines, $L_1$ and $L_2$, that obey the constraints described below.
CHAPTER 5. GRAPHEMES

Vertical Tick_Mark Geometric Constraints.
Assume a short vertical line \( L_1 \) and a long horizontal line \( L_2 \) are near to each other, they are a vertical tick_mark grapheme if the following constraints are satisfied:

1. short\((L_1)\);
2. vertical\((L_1)\);
3. long\((L_2)\);
4. horizontal\((L_2)\);
5. below\((L_1, L_2)\);
6. near\((L_1, L_2)\);

5.3 Representation of graphemes in feature sets

In a diagram, a grapheme is represented by a pair of \(<\) attribute, value \(>\), where the value is the occurrence number of the grapheme in the diagram. Thus, a diagram in our data set can be described as a vector of \(<\) grapheme, value \(>\) entries, i.e.,

\[<\text{grapheme}_1, \text{value}_1>, <\text{grapheme}_2, \text{count}_2>, ..., <\text{grapheme}_{24}, \text{count}_{24}>\]

5.4 Pre-processing of grapheme values

As we can see in Table. 5.1, the values of the graphemes vary in a large range such that the largest standard deviation of the grapheme values in our data set reach almost 2,000. We preprocess the grapheme feature values by replacing them with their base-10 logarithm values. The means and standard deviations of the processed grapheme values are listed in the fourth and fifth columns of Table. 5.1.

5.5 An example diagram with its grapheme values

An example bar chart diagram with its grapheme values are shown in Table. 5.2. The original values and the pre-processed values of the graphemes are both listed.
### Table 5.1: The means and standard deviations of grapheme values are calculated for our 700 instance data set. We then pre-process them with the base-10 logarithm. A grapheme value is defined as the occurrence number of the grapheme in a diagram.

<table>
<thead>
<tr>
<th>Grapheme</th>
<th>Original values</th>
<th>Log&lt;sub&gt;10&lt;/sub&gt; pre-processing</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Mean</td>
<td>STDEV.</td>
</tr>
<tr>
<td>Rectangle</td>
<td>96.35</td>
<td>508.61</td>
</tr>
<tr>
<td>Path</td>
<td>238.48</td>
<td>475.55</td>
</tr>
<tr>
<td>CurveElements</td>
<td>177.48</td>
<td>666.72</td>
</tr>
<tr>
<td>HorizontalLineElements</td>
<td>480.19</td>
<td>1399.21</td>
</tr>
<tr>
<td>VerticalLineElements</td>
<td>122.97</td>
<td>428.23</td>
</tr>
<tr>
<td>DiagonalLineElements</td>
<td>59.16</td>
<td>122.52</td>
</tr>
<tr>
<td>DataLinesX</td>
<td>16.81</td>
<td>49.9</td>
</tr>
<tr>
<td>DataLinesY</td>
<td>12.4</td>
<td>47.57</td>
</tr>
<tr>
<td>DataLinesSUM</td>
<td>29.21</td>
<td>96.27</td>
</tr>
<tr>
<td>Text</td>
<td>41.8</td>
<td>61.4</td>
</tr>
<tr>
<td>HorizontalText</td>
<td>1900.05</td>
<td>1939.9</td>
</tr>
<tr>
<td>VerticalText</td>
<td>121.41</td>
<td>196.45</td>
</tr>
<tr>
<td>horizontalTicks</td>
<td>34.9</td>
<td>148.76</td>
</tr>
<tr>
<td>verticalTicks</td>
<td>20.85</td>
<td>120.28</td>
</tr>
<tr>
<td>totalTicks</td>
<td>55.76</td>
<td>206.44</td>
</tr>
<tr>
<td>SameAreaRectangles</td>
<td>87.62</td>
<td>507.73</td>
</tr>
<tr>
<td>sameWidthBar</td>
<td>93.16</td>
<td>508.5</td>
</tr>
<tr>
<td>sameHeightBar</td>
<td>90.97</td>
<td>508.51</td>
</tr>
<tr>
<td>Arrow</td>
<td>10.87</td>
<td>41.74</td>
</tr>
<tr>
<td>CircleDataPoints</td>
<td>30.94</td>
<td>111.03</td>
</tr>
<tr>
<td>TriangleDataPoints</td>
<td>16.97</td>
<td>59.85</td>
</tr>
<tr>
<td>DiamondDataPoints</td>
<td>30.07</td>
<td>136.7</td>
</tr>
<tr>
<td>RectanglePoints</td>
<td>70.62</td>
<td>507.54</td>
</tr>
<tr>
<td>totalnumberofobjects</td>
<td>376.63</td>
<td>734.76</td>
</tr>
</tbody>
</table>
Table 5.2: A diagram and its grapheme values and pre-processed values. The value of Text grapheme is 57 since there are 57 text rendering commands in this diagram. The value of HorizontalText grapheme is 4560 since text length of the caption is also included, though the caption is not included in this diagram.
Chapter 6

Experiments and Analysis

6.1 Weka 3: Data Mining software in Java

Weka is an open source workbench for data mining that provides a rich set of machine learning algorithms from feature selection, to clustering, to classification. Here we mainly list the ones that we use in our research [60].

Weka implements a variety of supervised learning algorithms. C4.5 classifies the data set by applying an extended version of classical decision tree algorithm. NaiveBayes applies Bayes’ rule to instances that are assumed to be independent of one another. AdaBoost.M1 boosts a base learner in iterations on a weighted data set, focusing on learning from the “difficult” instances. We used two base learners: decision stump (a single level binary decision tree) and C4.5.

Unsupervised learning algorithms implemented in Weka are mostly partitional clustering algorithms such as KMeans and Expectation Maximization(EM). Hierarchical clustering algorithms including COBWEB are also available, though we do not explain them further in this section. Distance Metrics include Manhattan, Euclidean and Chebyshev distances.

SimpleKMeans repeatedly assigns each instance to the nearest clustering of the $K$ clusters and calculates the $K$ centers when all instances are assigned, until the $K$ centers stay unchanged in consecutive loops. The initial K centroids are determined by random seeds.

Expectation Maximization(EM) repeatedly estimates the cluster distribution based on the probabilities of each instance belonging to all the clusters, until the overall likelihood that the data set fits the cluster distribution stops increasing.

A distance metric calculates the distance between two instances in the feature space.
Assume the feature space is $n$-dimensional, the distance metric is defined as the distance between two points, $(x_1, \ldots, x_n)$ and $(y_1, \ldots, y_n)$ in $n$-dimensions.

- Manhattan Distance between two points is defined as the sum of the distances that one point needs to walk through toward the other point parallel to the axes, i.e., $|x_1 - y_1| + \ldots + |x_n - y_n|$.

- Euclidean Distance between two points is the length of the straight line connecting these two points, i.e., $\sqrt{(x_1 - y_1)^2 + \ldots + (x_n - y_n)^2}$.

- Chebyshev Distance between two points is the largest difference of the two points among all the differences along the $n$ dimensions, i.e., $\max\{|x_1 - y_1|, \ldots |x_n - y_n|\}$.

We analyze the experimental results in this chapter, focusing on not only discussing the algorithms, but also evaluating the capabilities of graphemes for representing diagrams.

### 6.2 Retrieval experiments

We take retrieval experiments by ranking the diagrams according to their distances to the query diagram and select the top ones to be the retrieved results. The Euclidean distance metric is applied as the distance metric to the grapheme set. Mean precision at a cut off of 10 and mean average precision are applied to evaluate the retrieval results.

#### 6.2.1 Mean precision at cut-off of 10

The precision at a cut off of 10 measures the portion of the relevant diagrams, i.e., the diagrams of the same label with the query diagram, among the top 10 retrieved diagrams. For example, if there are 8 bar charts among the top 10 retrieved diagrams for a bar chart query, the precision at 10 is 80%. The mean precision at 10 is the mean value of the precisions at 10 for all the possible query diagrams in a class. The higher the precision at 10, the better the grapheme represent the diagrams.
6.2. RETRIEVAL EXPERIMENTS

<table>
<thead>
<tr>
<th></th>
<th>Bar chart</th>
<th>Data curve</th>
<th>Data line</th>
<th>Data point</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>98.3%</td>
<td>98.0%</td>
<td>96.0%</td>
<td>98.2%</td>
<td>98.8%</td>
</tr>
</tbody>
</table>

Table 6.1: The mean precisions at a cut off of 10 diagrams for the five classes. The high mean precision at 10 of the five diagram classes show that for each of the classes, the dominating majority of the top 10 retrieved diagram are relevant.

6.2.2 Mean average precision

Another evaluation we apply is mean average precision that evaluates the retrieval by combining precision, relevance ranking, and overall recall. Unlike the precision at a cut off of 10 that counts the total number of relevant diagrams in the retrieval results, mean average precision also considers the other in which the retrieved diagrams are presented. It is the average of precision computed at the point of each of the relevant diagrams in the ranked retrieved list:

\[
\frac{\sum_{j=1}^{N} (\text{Precision}(j) \times \text{Relevance}(j))}{\text{number of relevant diagrams}}
\]

(6.1)

where \( j \) is the rank of a relevant diagram in the retrieval results, and \( N \) is the total number of retrieved diagrams. Function relevance\((j)\) is a binary function on the relevance: 1 when the \( j \)th result is relevant to the query, 0 otherwise.

The mean average precisions for the five classes are listed in Table. 6.2. The mean average precisions for the bar chart and tree diagrams are higher than .80. Those for the data curve, data line, and data point diagrams are between .64 and .78. Comparing to CBIR and text retrieval where average precision are usually smaller than .50, our retrieval results strongly prove that the graphemes can successfully represent the five diagram classes [59, 29].

<table>
<thead>
<tr>
<th></th>
<th>Bar chart</th>
<th>Data curve</th>
<th>Data line</th>
<th>Data point</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Precision</td>
<td>.856</td>
<td>.644</td>
<td>.772</td>
<td>.674</td>
<td>.805</td>
</tr>
</tbody>
</table>

Table 6.2: The high mean average precisions for the retrieval experiments prove that the bag of graphemes can successfully represent and retrieve the diagram classes.
6.3 Classification

We have taken multi-class (all-at-once) and binary classification (one-against-others) experiments on the five major diagram classes represented by twenty four graphemes in the collection of more than 700 diagrams. Four supervised learning algorithms are applied: decision tree algorithm C4.5, naive Bayes network, adaptive boosting algorithms that boosted decision stump and C4.5, respectively. The classifiers are tested in 10-fold cross-validations.

The performances of these algorithms are first illustrated in Fig. 6.1 proving that they perform well with our domain of vector diagrams. In binary classification, the algorithms produce recalls even as high as 95%.

![Figure 6.1: Recall values for multi-class (all-at-once) and binary classification (one-against-others) experiments of C4.5, NaiveBayes, AdaBoost.M1(DecisionStump), and AdaBoost.M1(C4.5). The lowest blue line is the recalls for the five classes all-at-once classifications. The top five lines show those of the binary classifications on five major diagram classes respectively.](image)

From the perspective of graphemes, we argue that our grapheme set is capable of representing the content of diagrams, in a “bag-of-features” manner. Particularly, the high true positive rates on the binary classification imply that these graphemes can successfully represent each individual diagram class.

For each of the five diagram classes, we have done four sets of binary classification experiments. As shown in Fig. 6.1, the four supervised learning algorithms can successfully
classify the diagrams. Particularly, all of the five diagram classes find boosting algorithms more efficient in binary classification, since the recalls of boosted C4.5 and boosted decision stump are higher than C4.5 and NaiveBayes. True positive, false positive and false negative example diagrams in the binary classifications for each diagram class can be found in Table 6.4. On the other hand, the extraordinary performance of binary classification implies that the grapheme set can sufficiently represent the diagrams individually. More details of the binary classifications will be discussed in later sections.

Furthermore, we analyze the experiments by discussing the true positive rates (TPR) and false positive rates (FPR). The TPR is defined as the portion of the instances in a certain diagram class that are correctly classified. The FPR is defined as the portion of the instances classified as a certain diagram class that are not truly belong to this diagram class. The TPR, FPR, and weighted error rates of the five binary classifications with AdaBoost.M1 (C4.5 as base learner) as shown in Table 6.3. The pair of TPR and FPR values indicate that the five diagram classes can be represented by a reasonably good operation point on an ROC curve, especially the pair of bar chart diagram, (.79, .05), and tree diagram, (.82, .03). An ROC is a graphical plot of TPR and FPR, providing an evaluation metric of a classification through the relationship between TPR and FPR. The error rate is weighted error rate on both positive and negative diagrams. The low error rates of less than .10 of bar chart and tree diagrams, less than .20 of data curve, data line, and data point diagrams prove that the binary classifications are successful for these five diagram classes.
Table 6.3: Evaluate the binary classifications (one against others) on the five major diagram classes with the true positive rate (TPR) and false positive rate (FPR) of the testing results for binary classifications (one against others). The TPR is the ratio of correctly classified diagrams to the total number of diagrams in this class. The FPR is the ratio of the diagrams that are incorrectly classified as a certain diagram class to the total number of diagrams that actually belong to the other class. Error rate is the weighted error rate on both positive and negative diagrams. The values are results from testing experiments of 10-fold cross validations.

6.3.1 The most powerful graphemes in binary classifications

To have a comprehensive view of the entire grapheme set, we overview the most powerful graphemes in Table 6.5. Each cell is a grapheme and its weights in boosted decision stumps in binary classification for each of the five diagram classes. The weight of a grapheme is defined to be the sum of the weights of the decision stumps built on the grapheme. For instance, the first row Rectangle, 1.79, 0.9, , , can be interpreted as: Rectangle is a grapheme selected to classify bar chart and data curve diagrams in their binary classifications respectively. Among the first ten decision stumps built for bar chart classification, the sum of weight of the decision stumps built on Rectangle is 1.79. That of data curve is 0.9. Rectangle is not selected to build a decision stump for data line, data point, or tree diagrams. This interpretation shows that Rectangle plays a more important role in bar chart than in data curve classification. Moreover, Rectangle does not help representing data line, data point or tree diagrams.

There are seven empty rows in the table: DiagonalLineY, VerticalTickMark, Rectangle-withSameArea, BarSameWidth, BarSameHeight, RectangleDataPoint, and TotalGraphics-Text. These graphemes are found by boosted decision stumps to not be capable of representing the diagrams. We will investigate these graphemes again with clustering and feature
<table>
<thead>
<tr>
<th>Class</th>
<th>True Positive</th>
<th>False Positive</th>
<th>False Negative</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar Chart</td>
<td><img src="image" alt="Bar Chart" /></td>
<td><img src="image" alt="False Positive" /></td>
<td><img src="image" alt="False Negative" /></td>
</tr>
<tr>
<td>Data Curve</td>
<td><img src="image" alt="Data Curve" /></td>
<td><img src="image" alt="False Positive" /></td>
<td><img src="image" alt="False Negative" /></td>
</tr>
<tr>
<td>Data Line</td>
<td><img src="image" alt="Data Line" /></td>
<td><img src="image" alt="False Positive" /></td>
<td><img src="image" alt="False Negative" /></td>
</tr>
<tr>
<td>Data Point</td>
<td><img src="image" alt="Data Point" /></td>
<td><img src="image" alt="False Positive" /></td>
<td><img src="image" alt="False Negative" /></td>
</tr>
<tr>
<td>Tree</td>
<td><img src="image" alt="Tree" /></td>
<td><img src="image" alt="False Positive" /></td>
<td><img src="image" alt="False Negative" /></td>
</tr>
</tbody>
</table>

**Table 6.4:** The True Positive, False Positive and False Negative example diagrams of the five major classes of diagrams in their binary classification experiments.
selection experiments. It appears that among these seven “useless” graphemes, six of them are complex graphemes that are composed of more than one basic graphemes. We have defined totally twelve complex graphemes and six of them are found not as capable as basic graphemes in classification. This discovery will provide us some guideline in our future work on defining graphemes.

Each column of the table lists the graphemes that have been selected to build decision stump in the first ten iterations of boosting. It is clear that not all of the 24 graphemes are selected to build the decision stumps. For example, four graphemes are found capable of representing and classifying bar chart, noticing the weight of grapheme HorizontalLine reaches as high as 2.36. Moreover, three out of four graphemes have weights larger than one. These high-weight graphemes proves that they can successfully dominate the classification of bar chart diagram. Take data point diagrams for another example, eight graphemes are selected to classify data point diagrams. The weight of Path is the highest 1.86, and six of the other graphemes have weights less than 0.5, which Path is extraordinary important to data point diagram, and it’s difficult to find other graphemes that are as efficient as Path in this binary classification. In summary, these different weights of graphemes explain the importance of these graphemes to the diagram classes, from the perspective of binary classification.
### Table 6.5: Graphemes and the weights of decision stumps on these graphemes in binary classification of the five diagram classes.

Each row illustrates a grapheme and its weight in five binary classifications respectively. Each column explains the graphemes that are selected to build the decision stump for a diagram class. The weight of a grapheme is defined as the sum of the weights of the decision stumps built on a grapheme, which shows the importance of the grapheme to the diagram in binary classifications. The graphemes are the most powerful features in classifying the diagram classes, according to our binary classifications.

<table>
<thead>
<tr>
<th>Grapheme</th>
<th>Bar chart</th>
<th>Data curve</th>
<th>Data dine</th>
<th>Data point</th>
<th>Tree</th>
</tr>
</thead>
<tbody>
<tr>
<td>Horizontal line</td>
<td>.36</td>
<td>.3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Diagonal line</td>
<td>1.16</td>
<td></td>
<td>2.99</td>
<td>0.38</td>
<td></td>
</tr>
<tr>
<td>Rectangle</td>
<td>1.79</td>
<td>0.9</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Path</td>
<td></td>
<td>1.04</td>
<td>1.86</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Vertical text</td>
<td></td>
<td>0.42</td>
<td></td>
<td>0.4</td>
<td>2.8</td>
</tr>
<tr>
<td>Data line (X)</td>
<td></td>
<td></td>
<td></td>
<td>1.33</td>
<td>0.3</td>
</tr>
<tr>
<td>Total Tick Mark</td>
<td></td>
<td>0.53</td>
<td>0.31</td>
<td>1.69</td>
<td></td>
</tr>
<tr>
<td>Horizontal text</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>1.39</td>
</tr>
<tr>
<td>Arrow</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.9</td>
</tr>
<tr>
<td>Curve</td>
<td></td>
<td>0.47</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total data line</td>
<td></td>
<td></td>
<td></td>
<td>0.81</td>
<td></td>
</tr>
<tr>
<td>Text</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.53</td>
</tr>
<tr>
<td>Horizontal Tick Mark</td>
<td></td>
<td>0.4</td>
<td>0.15</td>
<td>0.43</td>
<td></td>
</tr>
<tr>
<td>Rectangle of the same area</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Circle Data Point</td>
<td></td>
<td>0.68</td>
<td>0.5</td>
<td>0.53</td>
<td></td>
</tr>
<tr>
<td>Diamond Data Point</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.87</td>
</tr>
<tr>
<td>Vertical line</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>0.39</td>
</tr>
</tbody>
</table>
6.3.2 Bar chart binary classification

We have proved that the set of 24 graphemes successfully represents the content of vector diagrams. Starting this section, we study the graphemes regarding their individual capability of representing the diagrams. We proceed this study by analyzing the binary classifications, in particular, boosted decision stumps in AdaBoost.M1 for two reasons. First, the simple nature of decision stump enables us to focus on a single grapheme in one hypothesis. Other base learners require more effort to acquire the similar amount of information of graphemes. For example, the complete decision tree created by C4.5 makes it difficult to study each single grapheme in the tree. Second, since the four algorithms perform similarly on the binary classification experiments, we assume that our analysis on graphemes is independent from the choice of classification algorithm, which means that we may gain similar insights of the graphemes even if we study a different algorithm instead of boosted decision stump.

For each diagram class, we utilize a table, a decision stump table (e.g., Table 6.6), to summarize the first 10 decision stumps in the AdaBoost.M1 classification, starting with the first decision stump. Each decision stump, listed as a row of the table, includes the following information: the grapheme selected to build this decision stump, the split value on the grapheme, the value range of the grapheme, prediction probabilities on each of the two branches of the decision stump, and the weight of the decision stump. An additional column of the table is final true positive rates that is defined as the final hypothesis’ true positive rate if the decision stump on the current row is the last decision stump built for the boosting classification. The final true positive rate on the \( j \)th row is defined to be the true positive rate of the final hypothesis if AdaBoost.M1 is determined to iterate \( j \) times, i.e., create \( j \) decision stump hypotheses. For instance, the fourth final true positive rate 83.82% indicates that if AdaBoost.M1 iterates four times before terminate, the four decision stump hypotheses can the represented by the first four rows of the table, and the final hypothesis’s true positive rate is 83.82%. The weights of the four hypotheses are 1.15, 1.05, 0.74, and 0.44 sequentially.

For instance, the first row of Table 6.6 is
can be explained that the first hypothesis in the bar chart binary classification on AdaBoost.M1 (DecisionStump) is created on grapheme DiagonalLine such that if the value of DiagonalLine of an instance \(i\) is smaller than 0.15, then

\[ P(i \in \text{Bar-chart}) = 0.52, \ P(i \notin \text{Bar-chart}) = 0.47; \]

otherwise, i.e., instance \(i\)’s DiagonalLine is less than 0.15, then

\[ P(i \in \text{Bar-chart}) = 0.09, \ \text{and} \ P(i \notin \text{Bar-chart}) = 0.90. \]

The largest probability is bold, 0.90 in this example. The weight of this decision is 1.1.5. The true positive rate of the final hypothesis, in this example, the first decision stump itself, is 76.17%.

We gain insights of graphemes by discussing the decision stump built on the graphemes. The prediction probability can tell if a decision stump is able to predict positive instance or negative instance, how confident this decision stump is when make the prediction, and how important the decision stump to the final hypothesis, i.e., the weight of the decision stump. The more important the decision stump is, the more important the grapheme is to the diagram class. The higher the final hypothesis’ true positive rate, the more informative the grapheme.

It is shown in Table. 6.6 that DiagonalLine, Rectangle, HorizontalLine, and Curve are the most distinguishing graphemes for bar chart. Among them, DiagonalLine and Curve are more confident in prediction since they achieve distinctive probabilities for positive and negative prediction respectively on two branches. In detail, if DiagonalLine of a diagram is smaller than 0.15, the decision stump tends to predicts it as a bar chart; if larger than 0.15, it strongly predicts to be not a bar chart. This single grapheme produces true positive rate 76.17% for bar chart diagrams by identifying those diagrams that are not bar charts. The other two graphemes in this table, Rectangle and HorizontalLine, are less confident. In particular, Rectangle in the second row predicts negative on both branches with probabilities of 0.92 and 0.60 respectively, which means it always predict a diagram to be not a bar chart
<table>
<thead>
<tr>
<th>Grapheme</th>
<th>Split value(Range)</th>
<th>Pred.(&lt;=split)</th>
<th>Pred.(&gt;split)</th>
<th>Hypothesis Weight</th>
<th>Final TP rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>DiagonalLine</td>
<td>&gt;0.15 (0 .. 3.09)</td>
<td>0.52</td>
<td>0.48</td>
<td>0.09</td>
<td>0.91</td>
</tr>
<tr>
<td>Rectangle</td>
<td>&lt;=1.21 (0 .. 3.95)</td>
<td>0.07</td>
<td><strong>0.93</strong></td>
<td>0.39</td>
<td><strong>0.61</strong></td>
</tr>
<tr>
<td>Rectangle</td>
<td>&lt;=0.92 (0 .. 3.95)</td>
<td>0.12</td>
<td><strong>0.88</strong></td>
<td>0.62</td>
<td>0.38</td>
</tr>
<tr>
<td>HorizontalLine</td>
<td>&gt;2.78 (0 .. 4.18)</td>
<td>0.45</td>
<td><strong>0.55</strong></td>
<td>0.02</td>
<td><strong>0.98</strong></td>
</tr>
<tr>
<td>HorizontalLine</td>
<td>&gt;2.78 (0 .. 4.18)</td>
<td><strong>0.56</strong></td>
<td>0.44</td>
<td>0.03</td>
<td><strong>0.97</strong></td>
</tr>
<tr>
<td>Curve</td>
<td>&gt;0.15 (0 .. 3.84)</td>
<td>0.51</td>
<td>0.49</td>
<td>0.20</td>
<td><strong>0.80</strong></td>
</tr>
<tr>
<td>HorizontalLine</td>
<td>&gt;2.93 (0 .. 4.18)</td>
<td>0.41</td>
<td><strong>0.59</strong></td>
<td>0.01</td>
<td><strong>0.99</strong></td>
</tr>
<tr>
<td>HorizontalLine</td>
<td>&gt;2.93 (0 .. 4.18)</td>
<td><strong>0.53</strong></td>
<td>0.47</td>
<td>0.0</td>
<td><strong>1.0</strong></td>
</tr>
<tr>
<td>HorizontalLine</td>
<td>&gt;2.93 (0 .. 4.18)</td>
<td>0.46</td>
<td><strong>0.54</strong></td>
<td>0.0</td>
<td><strong>1.0</strong></td>
</tr>
<tr>
<td>HorizontalLine</td>
<td>&gt;2.93 (0 .. 4.18)</td>
<td><strong>0.53</strong></td>
<td>0.47</td>
<td>0.0</td>
<td><strong>1.0</strong></td>
</tr>
</tbody>
</table>

**Table 6.6**: First 10 decision stumps in bar chart binary boosting classification. The first four graphemes selected in the boosting iterations are listed. Each row consists of the grapheme, the split value of this grapheme, value range, prediction probabilities on each branch of the decision stump, weight of the decision stump, and the final true positive rate. The final true positive rate is the true positive rate of the final hypothesis assuming that the current decision stump is the last hypothesis in boosting. For instance, the first row can be interpreted as: the first grapheme selected in boosting is DiagonalLine with the split value 0.15 so that if the value of DiagonalLine of a diagram is smaller than 0.15, the diagram is a bar chart with probability of 0.52; if its value is larger than 0.15, the diagram is not a bar chart with probability of 0.90. Table. 6.7 to Table. 6.10 are structured the same as this table.
no matter what its Rectangle value is. This unreasonable prediction is adjusted by the next
decision stump that is also built on Rectangle: the split value on Rectangle drops from 1.21
to 0.92 such that if the value is larger than 0.92, the diagram is more believed to be a bar
chart.

HorizontalLine is an extraordinary grapheme in the boosting classification on bar chart.
Its has not only been selected in six iterations out of ten, but also unwilling to adjust its split
value like Rectangle does. In the 4th iteration, HorizontalLine predicts negative diagrams
on both branches with probabilities 0.54 and 0.98, which means this decision stump needs
adjustment to predict differently on two branches. However, its negative prediction is so
confident that HorizontalLine does not agree to shift its split value. Instead, it adjusts one
of its predictions from negative (0.54) that is almost a random guess to positive (0.56) that is
still an almost random guess, which means that the adjustment does not affect the prediction
on this branch. Meanwhile, the strong negative prediction on the other branch keeps high
probability of 0.97. Thus, HorizontalLine preserve its confidence prediction on a branch by
switching from a random guess negative prediction to a random guess positive prediction on
the other branch.

From the column of final true positive rate, we notice that the true positive rate of the
final hypothesis boosts from 76.17% to 83.82% on the third decision stump. The boosting
is caused by grapheme Rectangle when its decision stump adjusts its split value from 1.21
to 0.92, such that the decision stump can predict a bar chart with probability 0.61 when its
value of Rectangle is larger than 0.92.

The extreme grapheme, HorizontalLine, is interesting from the point of view of decision
stump. Between the fourth and fifth iterations, HorizontalLine boosts the final true positive
rate from 83.82% to 84.68%, however, the hypothesis weight only reaches 0.44. Considering
the last hypothesis weight is 0.74 on Rectangle, we regard HorizontalLine informative than
Rectangle to bar chart binary classification. HorizontalLine is then selected four times in a
row on the seventh to tenth iterations. We notice that the hypotheses weights drop from
0.56 on the seventh iteration to 0.23 on the tenth iteration. The decreasing weight explains
the fact that the final true positive rate tends to drop in the last four iterations.
6.3.3 Data curve diagram binary classification

Table 6.7 shows that the most representative graphemes for data curve diagrams are HorizontalLine, Rectangle, CircleDataPoint, VerticalTickMark, Text, and Vertical Text. Among them, HorizontalLine is the most powerful since the true positive rate of its own decision stump reaches 85.24%, same with the final true positive rate after ten iterations. Given the fact that the final true positive rate after one thousand iterations gets to 88.08%, HorizontalLine achieves the highest weight 1.75 among the 24 graphemes in all the binary classifications of the five diagram classes.

However, it is not safe to jump to a conclusion that HorizontalLine can classify data curve diagrams all by itself since HorizontalLine is a negative prediction grapheme for data curve, noticing it predicts a negative instance with high probability 0.91. In other words, it can predict with high confidence a diagram that is not a data curve, but is not able to correctly predict a data curve diagram. Even if it chooses to make positive prediction, its prediction may be wrong. In the third iteration where HorizontalLine attempts to make a positive prediction with high probability at 0.79, the final true positive rate actually drops 0.1%. This decision stump is later corrected in the fourth iteration when HorizontalLine is selected again to align with the original negative prediction.

Besides HorizontalLine, there is another grapheme that makes positive prediction in the classification: Rectangle. It can correctly predict a Data curve diagram with probability 0.74 if the Rectangle value is smaller than 0.92. Considering the value range of Rectangle is [0..3.95], we can interpret the prediction such that if a diagram contains a moderately small number of rectangles, it is a data curve diagram with probability 0.74. If the diagram has more rectangles, it is not a data curve diagram with probability 0.69, as shown on the second row on the table.

Another look at the first three iterations of Table 6.7 motivates us to argue that it is the instance weight adjustment after the second iteration that misleads HorizontalLine on the third iteration to predict contradictorily from the first iteration. The prediction probability on negative diagrams drops from 0.91 in the first iteration to 0.67 in the third iteration, given
the value ranges of the grapheme overlaps with each other (<=2.84 in the first iteration and <=2.54 in the third iteration). Moreover, the weak branch of the decision stump in the first iteration even comes to dominate the decision stump in the third iteration, noticing that the prediction probability 0.44 on the weak branch in the first iteration climbs up to 0.79 in the third iteration.

<table>
<thead>
<tr>
<th>Grapheme</th>
<th>Split value (Range)</th>
<th>Pred.(&lt;=split)</th>
<th>Pred.(&gt;split)</th>
<th>Hypothesis Weight</th>
<th>Final TP rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>HorizontalLine</td>
<td>&lt;=2.80</td>
<td></td>
<td></td>
<td>0.91</td>
<td>0.44</td>
</tr>
<tr>
<td>Rectangle</td>
<td>&lt;=0.92</td>
<td>0.74</td>
<td>0.26</td>
<td>0.31</td>
<td>0.69</td>
</tr>
<tr>
<td>HorizontalLine</td>
<td>&lt;=2.54</td>
<td>0.33</td>
<td>0.67</td>
<td>0.79</td>
<td>0.21</td>
</tr>
<tr>
<td>HorizontalLine</td>
<td>&lt;=1.65</td>
<td>0.32</td>
<td>0.68</td>
<td>0.63</td>
<td>0.37</td>
</tr>
<tr>
<td>CircleDataPoint</td>
<td>&gt;1.56</td>
<td>0.53</td>
<td>0.47</td>
<td>0.20</td>
<td>0.80</td>
</tr>
<tr>
<td>VerticalTickMark</td>
<td>&lt;=0.39</td>
<td>0.30</td>
<td>0.70</td>
<td>0.52</td>
<td>0.48</td>
</tr>
<tr>
<td>Text</td>
<td>&gt;1.31</td>
<td>0.56</td>
<td>0.44</td>
<td>0.33</td>
<td>0.67</td>
</tr>
<tr>
<td>CircleDataPoint</td>
<td>&gt;2.43</td>
<td>0.45</td>
<td>0.55</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>CircleDataPoint</td>
<td>&gt;2.43</td>
<td>0.51</td>
<td>0.49</td>
<td>0.0</td>
<td>1.0</td>
</tr>
<tr>
<td>VerticalText</td>
<td>&lt;=1.96</td>
<td>0.36</td>
<td>0.64</td>
<td>0.57</td>
<td>0.42</td>
</tr>
</tbody>
</table>

**Table 6.7:** First 10 decision stumps in data curve binary boosting classification. The structure of this table is the same as Table. 6.6. The first decision stump that is built on HorizontalLine can predict with probability 0.91 the diagrams that are not data curve. Its true positive rate reaches as high as 85.24%, only 1% less than the final recall after one hundred iterations. However, this grapheme builds an opposite decision stump in the third iteration, perhaps due to the instance weight adjustment in the second iteration. Also, it is shown that decision stump on Rectangle is able to positively prediction a data curve diagram with probability 0.74, such that if a diagram has a small number of rectangles, it can be predicted to be a data curve diagram with probability 0.74.
6.3.4 Data line diagram binary classification

For data line diagram binary classification, the four most distinguishing graphemes selected in the ten iterations are DiagonalLine, TickMark, DataLineX, and Path. Table 6.8 shows that these graphemes are able to predict negative diagrams with average probability 0.90, and the final recall reaches 87.23%.

In the first iteration, the decision stump predicts negative diagrams on both branches with probabilities 0.97 and 0.75 respectively. The split value of DiagonalLine is then adjusted from \( \leq 0.39 \) to \( \leq 0.15 \) in the second iteration such that weak branches of the second decision stump predict positive diagrams.

We notice that the fifth decision stump on Path predicts not very well and it is assigned a low weight, thus the final recall drops about 0.71%. The reason for the poor prediction of Path decision stump is probably that data line diagrams have Path values larger than the split value 1.57 given the value range \([0..3.96]\), however Path decision stump can only predict with high probability 1.0 when the value is smaller than 1.57. When larger than 1.57, the decision stump performs like a random guess with positive prediction probability 0.54 and negative prediction probability 0.46.
### Decision Stump Hypothesis

<table>
<thead>
<tr>
<th>Grapheme</th>
<th>Split value (Range)</th>
<th>Pred. (&lt;=split)</th>
<th>Pred. (&gt;=split)</th>
<th>Hypothesis Weight</th>
<th>Final TP rate</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>DiagonalLine</td>
<td>&lt;=0.39</td>
<td>0.03</td>
<td>0.97</td>
<td>0.25</td>
<td>0.75</td>
</tr>
<tr>
<td>DiagonalLine</td>
<td>&lt;=0.15</td>
<td>0.11</td>
<td>0.89</td>
<td>0.63</td>
<td>0.37</td>
</tr>
<tr>
<td>TickMark</td>
<td>&lt;=0.875</td>
<td>0.10</td>
<td>0.90</td>
<td>0.51</td>
<td>0.49</td>
</tr>
<tr>
<td>DataLineX</td>
<td>&lt;=0.39</td>
<td>0.21</td>
<td>0.79</td>
<td>0.47</td>
<td>0.53</td>
</tr>
<tr>
<td>Path</td>
<td>&lt;=1.57</td>
<td>0.0</td>
<td>1.0</td>
<td>0.54</td>
<td>0.46</td>
</tr>
<tr>
<td>DataLinexX</td>
<td>&lt;=0.39</td>
<td>0.30</td>
<td>0.70</td>
<td>0.59</td>
<td>0.42</td>
</tr>
<tr>
<td>DiagonalLine</td>
<td>&gt;1.51</td>
<td>0.57</td>
<td>0.43</td>
<td>0.27</td>
<td>0.73</td>
</tr>
<tr>
<td>Path</td>
<td>&lt;=1.57</td>
<td>0.0</td>
<td>1.0</td>
<td>0.45</td>
<td>0.55</td>
</tr>
<tr>
<td>Path</td>
<td>&lt;=1.57</td>
<td>0.0</td>
<td>1.0</td>
<td>0.53</td>
<td>0.47</td>
</tr>
<tr>
<td>Path</td>
<td>&lt;=1.57</td>
<td>0.0</td>
<td>1.0</td>
<td>0.47</td>
<td>0.53</td>
</tr>
</tbody>
</table>

**Table 6.8**: First 10 decision stumps in data line binary boosting classification. The structure of this table is the same as Table 6.6. In the first iteration, the decision stump predict negative diagrams on both branches with probabilities 0.97 and 0.75 respectively. The split value of DiagonalLine is then adjusted from <=0.39 to <=0.15 in the second iteration such that weak branches of the second decision stump predict positive diagrams. The true positive rate of the first decision stump on DiagonalLine reaches as high as 83.54%. The final true positive rate is boosted in the sixth iteration, also on DiagonalLine, up to 86.52%.

#### 6.3.5 Data point diagram binary classification

In data point binary classification, as many as eight graphemes: Path, TotalTickMark, DataLine, CircleDataPoint, VerticalText, DiagonalLine, DataLineX and HorizontalTickMark, are selected for the decision stumps in the first ten iterations. Grapheme Path, the low-weight grapheme in other binary classifications such as in data line classification, is first selected and achieves high weight 0.9 and 0.71. The data point diagrams where data points are composed of paths can be predicted correctly with probability 0.62 when the value of Path is larger than 1.9 in the range of [0..3.69], as shown in Table 6.9.

We notice the final true positive rate boosts from 77.73% to 80.14% on grapheme CircleDataPoint in the sixth iteration. Although CircleDataPoint is more confident (probability 0.63) in negative prediction, it can also predict the positive diagrams with probability 0.61 when the grapheme value is larger than 0.82 in the range [0..3.1]. Obviously CircleDataPoints can represent data point diagrams that contains a large number of circle data points.
Another grapheme, DiagonalLine, can positively predict data point diagrams such that if the DiagonalLine value of a diagram is larger than 2.20 in the range of $[0..3.09]$, the diagram is predicted to be a data point diagram with probability 0.68.

<table>
<thead>
<tr>
<th>Grapheme</th>
<th>Split value(Range)</th>
<th>Pred.(&lt;=split)</th>
<th>Pred.(&gt;split)</th>
<th>Hypothesis Weight</th>
<th>Final TP rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Path</td>
<td>&lt;=1.90</td>
<td>0.10 0.90</td>
<td>0.31 0.69</td>
<td>1.25</td>
<td>77.73%</td>
</tr>
<tr>
<td>Path</td>
<td>&lt;=1.90</td>
<td>0.29 0.71</td>
<td>0.62 0.38</td>
<td>0.61</td>
<td>77.73%</td>
</tr>
<tr>
<td>TickMark</td>
<td>&lt;=0.15</td>
<td>0.22 0.78</td>
<td>0.52 0.48</td>
<td>0.31</td>
<td>77.73%</td>
</tr>
<tr>
<td>DataLine</td>
<td>&gt;0.39</td>
<td>0.49 0.51</td>
<td>0.25 0.75</td>
<td>0.37</td>
<td>77.73%</td>
</tr>
<tr>
<td>DataLine</td>
<td>&gt;0.39</td>
<td>0.58</td>
<td>0.42 0.67</td>
<td>0.44</td>
<td>77.73%</td>
</tr>
<tr>
<td>CircleDataPoint</td>
<td>&lt;=0.82</td>
<td>0.37 0.63</td>
<td>0.61 0.39</td>
<td>0.5</td>
<td>80.14%</td>
</tr>
<tr>
<td>VerticalText</td>
<td>&lt;=0.35</td>
<td>0.32 0.68</td>
<td>0.57 0.43</td>
<td>0.4</td>
<td>79.85%</td>
</tr>
<tr>
<td>DiagonalLine</td>
<td>&gt;2.20</td>
<td>0.42 0.58</td>
<td>0.68 0.32</td>
<td>0.38</td>
<td>80.14%</td>
</tr>
<tr>
<td>DataLineX</td>
<td>&lt;=1.57</td>
<td>0.55 0.46</td>
<td>0.23 0.77</td>
<td>0.3</td>
<td>80.28%</td>
</tr>
<tr>
<td>Horiz.TickMark</td>
<td>&gt;1.7</td>
<td>0.50</td>
<td>0.50 0.75</td>
<td>0.15</td>
<td>80.70%</td>
</tr>
</tbody>
</table>

Table 6.9: First 10 decision stumps for data point diagram binary boosting classification. The structure of this table is the same as Table 6.6. Grapheme Path is the most capable of classifying data point diagrams, though is the least important one of the top four graphemes in Data line diagram binary classification. The final true positive rate boosts from 77.73% to 80.14% in the sixth iteration on grapheme CircleDataPoint, which indicates CircleDataPoint is able to represent data point diagram. Grapheme DiagonalLine can positively predict Data point diagram with probability 0.68.
6.3.6 Tree diagram binary classification

In tree diagram binary classification, grapheme TotalTickMark, VerticalText, Arrow, HorizontalText, DiamondDataPoint, VerticalLine, HorizontalTickMark, and CircleDataPoint are selected in the first 10 iterations. TotalTickMark first classifies those non-tree diagrams with probability 0.97 if the grapheme value is larger than 0.65 in range [0..3.47], considering a tree diagram usually does not contain tick marks at all. The classification is boosted in the third iteration from 84.11% to 90.35%, thanks to VerticalText’s high probability 0.82 to predict the diagrams that are not tree. Tree diagrams usually only contains horizontal text. If many instances of vertical text can be found in a diagram, grapheme VerticalText predicts it is not a tree with probability 0.82. Graphemes that help positively predict tree diagrams include HorizontalText, Arrow and CircleDataPoint since a tree diagram may contain these graphemes with pretty high values. Graphemes that help negatively predict tree diagrams include HorizontalTickMark and DiamondDataPoint, since a tree diagram usually does not include tick mark, neither small data point.

<table>
<thead>
<tr>
<th>Decision Stump Hypothesis</th>
<th>Split value(Range)</th>
<th>Pred.(&lt;=split)</th>
<th>Pred.(&gt;split)</th>
<th>Hypothesis Weight</th>
<th>Final TP rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>TickMark</td>
<td>&gt;0.65</td>
<td>0.58 0.42 0.03</td>
<td>0.97</td>
<td>1.69</td>
<td>84.11%</td>
</tr>
<tr>
<td>VerticalText</td>
<td>&gt;0.35</td>
<td>0.41 0.59 0.05</td>
<td>0.95</td>
<td>1.5</td>
<td>84.11%</td>
</tr>
<tr>
<td>VerticalText</td>
<td>&gt;0.35</td>
<td>0.76 0.24 0.18</td>
<td>0.82</td>
<td>1.3</td>
<td>90.35%</td>
</tr>
<tr>
<td>Arrow</td>
<td>&gt;0.54</td>
<td>0.31 0.69 0.76</td>
<td>0.24</td>
<td>0.91</td>
<td>90.21%</td>
</tr>
<tr>
<td>HorizontalText</td>
<td>&gt;3.50</td>
<td>0.42 0.58 0.85</td>
<td>0.15</td>
<td>0.66</td>
<td>88.65%</td>
</tr>
<tr>
<td>DiamondDataPoint</td>
<td>&gt;1.02</td>
<td>0.70 0.30 0.26</td>
<td>0.74</td>
<td>0.87</td>
<td>89.92%</td>
</tr>
<tr>
<td>VerticalLine</td>
<td>&lt;=0.54</td>
<td>0.15 0.85 0.55</td>
<td>0.45</td>
<td>0.39</td>
<td>90.49%</td>
</tr>
<tr>
<td>HorizontalTickMark</td>
<td>&gt;0.39</td>
<td>0.52 0.48 0.23</td>
<td>0.77</td>
<td>0.43</td>
<td>90.21%</td>
</tr>
<tr>
<td>HorizontalText</td>
<td>&gt;3.81</td>
<td>0.34 0.66 0.93</td>
<td>0.07</td>
<td>0.73</td>
<td>90.78%</td>
</tr>
<tr>
<td>CircleDataPoint</td>
<td>&gt;0.15</td>
<td>0.40 0.60 0.67</td>
<td>0.33</td>
<td>0.53</td>
<td>90.92%</td>
</tr>
</tbody>
</table>

Table 6.10: First 10 decision stumps in tree binary boosting classification. The structure of this table is the same as Table 6.6. Both TotalTickMark and HorizontalTickMark are selected as negative prediction graphemes since tree diagrams do not usually contain tick marks. VerticalText also helps predict those diagrams that are not trees since trees mainly contains only horizontal text.
6.3.7 Comparison of graphemes and histograms in binary classification

To compare the capability of representing diagrams by graphemes and histograms, we take path length histogram and rectangle area histogram respectively as feature set, run the bar chart binary classifications. We compare these two histograms with the 24 grapheme set on bar chart binary classification. It shows that our grapheme set is more capable of representing diagrams in terms of binary classification.

Path length histogram is an array of 10 integers, each representing the number of paths with a certain range of length. The histogram is created in the following way. First, the lengths of all the paths in a diagrams are measured, then the range of the length is splitted into 10 sub-ranges. For a sub-range, say, the $i$th sub-range, the paths of such lengths are counted, and the number of the paths is the $i$th element of the histogram. Rectangle area histogram is created in the similar way, except that the areas of all the rectangles in a diagram is measured and splitted into 10 sub-ranges.

As shown in Fig. 6.2, we compare grapheme set, path length histogram, and rectangle area histogram in bar chart binary classifications. The grapheme set (the upper fine dashed line) outperforms the path length histogram (the middle solid line) and the rectangle area histogram (the lower dashed line) in terms of true positive rate.
Figure 6.2: To compare the capability of representing diagrams of grapheme and histogram, we take path length histogram and rectangle area histogram respectively as feature set, run the bar chart binary classifications. We compare these two histograms with the 24 grapheme set on bar chart binary classification. It shows that our grapheme set is more capable of representing diagrams in terms of binary classification. The upper fine dashed line is for grapheme, the middle solid line is for path length histogram, and the bottom dashed line is for rectangle area histogram.
6.4 Multi-class classification

We examine the capability of the graphemes as a feature set to represent the entire five diagram classes. We study the evaluation metrics and confusion matrix of the best-performing algorithm, boosted C4.5, in Table 6.12 and Table 6.11. It appears that among the five diagram classes, bar chart and tree are easier to be classified than the data curve, data line and data point classes. The precision and true positive rates of bar chart and tree are at most 20% higher than those of the other three. This proves that the grapheme set represents more sufficiently bar chart and tree diagram classes than data curve, data line and data point classes.

<table>
<thead>
<tr>
<th>Classes</th>
<th>Number of instances</th>
<th>predicted as</th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Bar-chart</td>
<td>Data-curve</td>
<td>Data-line</td>
<td>Data-point</td>
<td>Tree</td>
</tr>
<tr>
<td>Bar-chart</td>
<td>181</td>
<td>140</td>
<td>5</td>
<td>7</td>
<td>16</td>
<td>13</td>
</tr>
<tr>
<td>Data-curve</td>
<td>104</td>
<td>3</td>
<td>62</td>
<td>11</td>
<td>15</td>
<td>13</td>
</tr>
<tr>
<td>Data-line</td>
<td>116</td>
<td>2</td>
<td>15</td>
<td>70</td>
<td>20</td>
<td>9</td>
</tr>
<tr>
<td>Data-point</td>
<td>157</td>
<td>16</td>
<td>21</td>
<td>20</td>
<td>88</td>
<td>12</td>
</tr>
<tr>
<td>Tree</td>
<td>147</td>
<td>15</td>
<td>11</td>
<td>7</td>
<td>12</td>
<td>102</td>
</tr>
</tbody>
</table>

Table 6.11: Confusion matrix of multi-class classification of AdaBoost.M1 (C4.5 as base learner). The fact that the diagonal numbers are much larger than the others on the same row proves the graphemes successfully represent the classes, especially for bar chart(140 out of 181) and Tree diagrams(102 out of 147). Another fact is noticed that data curve, data line, and Data point are more easily to be predicted as one of these three classes than the other two.

The boosting algorithm is shown in Fig. 6.1 to be a slightly better choice for our domain regarding multi-class classification, since boosted C4.5 outperforms NaiveBayes and C4.5 with the highest true positive rate 65.7%. Comparing boosting classifications with C4.5 and decision stump as base learner, it is clear that the performance of a boosting classification partially depends on the choice of base learner. When decision stump is chosen to be the base learner in AdaBoost.M1, the classification gets a low true positive rate value of 16.0%. Because of its nature of binary classification, decision stump fails to classify more than two classes. With decision stump’s error rate 0.64 in classifying the five classes all at once in an iteration, AdaBoost.M1 quits boosting the decision stump.
Table 6.12: Metrics of multi-class classification of AdaBoost.M1 (C4.5 as base learner): True Positive Rate (TP), False Positive Rate (FP), Precision, and true positive rate. Each diagram class and the weighted average are both calculated. We can notice that (1) bar chart and tree diagrams can be classified with pretty high precision rates of 0.795 and 0.685, respectively; (2) low false positive rates can be guaranteed among all the five diagram classes, which indicates that among the predicted diagram in a class, only a small proportion are from the other four diagram classes.

<table>
<thead>
<tr>
<th>Class</th>
<th>TP Rate</th>
<th>FP Rate</th>
<th>Precision</th>
<th>TP rate</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar-chart</td>
<td>77.3%</td>
<td>6.9%</td>
<td>79.5%</td>
<td>77.3%</td>
</tr>
<tr>
<td>Data-curve</td>
<td>59.6%</td>
<td>8.7%</td>
<td>54.4%</td>
<td>59.6%</td>
</tr>
<tr>
<td>Data-line</td>
<td>60.3%</td>
<td>7.6%</td>
<td>60.9%</td>
<td>60.3%</td>
</tr>
<tr>
<td>Data-point</td>
<td>56.1%</td>
<td>11.5%</td>
<td>58.3%</td>
<td>56.1%</td>
</tr>
<tr>
<td>Tree</td>
<td>69.4%</td>
<td>8.4%</td>
<td>68.5%</td>
<td>69.4%</td>
</tr>
<tr>
<td>Weighted Avg.</td>
<td>65.5%</td>
<td>8.6%</td>
<td>65.7%</td>
<td>65.5%</td>
</tr>
</tbody>
</table>

We also notice that the majority of the FP rates (false alarm), for all the five diagram classes are below 0.1. FP rate is the proportion of the negative instances that are incorrectly predicted as positive instances. This indicates that though C4.5 can only correctly classify around 65.7% of the positive instances of the diagrams, it can identify about 90% of the instances that are not in each of the classes. For example, even though data curve diagram has the lowest true positive rate 54.4%, i.e., only 62 out of 104 Data curve diagrams are correctly classified, there are only 8.7% of the other four diagram instances that are predicted incorrectly as data curve diagrams.

Furthermore, we argue that, for each of the three less represented diagram classes (data curve, data line and data point), our grapheme set can successfully predict these three classes as different from bar chart and tree, however fail to distinguish from each other. The italicized 3x3 sub-table with diagonal values 62, 70, and 88 in Table 6.11 explains the confusion brought by the graphemes. Take data line for example, 35 out of 116, i.e., 30.1%, data line diagrams are classified as data curve or data point, comparing to only 2 are classified as bar chart.
6.4.1 Comparison of multi-class and binary classification

Despite of the performance differences between multi-classification and binary classifications, we further find the similarity between them in Table 6.13. We list the true positive rates of each diagram class in the multi-class classification, in particular, AdaBoost.M1(C4.5 as base learner), in the second column. The third column lists the highest true positive rate for each diagram class in its four sets of binary classifications. It can be noticed that bar chart and tree diagrams achieve higher true positive rates, align with the their true positive rates in multi-class classification. Moreover, although the true positive rates of data curve, data line, and data point diagrams classification are lower, they have been improved dramatically in binary classification comparing to multi-class classification.

<table>
<thead>
<tr>
<th>Class</th>
<th>Multi-class classification</th>
<th>Binary classification</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar-chart</td>
<td>0.795</td>
<td>0.88</td>
</tr>
<tr>
<td>Data-curve</td>
<td>0.544</td>
<td>0.883</td>
</tr>
<tr>
<td>Data-line</td>
<td>0.609</td>
<td>0.882</td>
</tr>
<tr>
<td>Data-point</td>
<td>0.583</td>
<td>0.807</td>
</tr>
<tr>
<td>Tree</td>
<td>0.685</td>
<td>0.936</td>
</tr>
</tbody>
</table>

Table 6.13: Comparison of true positive rates for each individual diagram class in multi-class (boosted C4.5) and binary classification. true positive rate for binary classification is the highest one among the four binary classifications with C4.5, NaiveBayes, AdaBoost.M1(DecisionStump) and AdaBoost.M1(C4.5). It is clear that, first, binary classification outperforms multi-class classification. Second, bar chart and tree are easier to be classified. Third, significant improvement can be seen on the true positive rates of data curve, data line and data point diagrams in binary classification.
6.5 Clustering

We choose KMeans and EM to cluster the diagrams and analyze the experimental results in this section. We first discuss two factors of clustering: cluster number and starting points for KMeans. To find the optimal cluster number, we take a set of experiments with cluster numbers from 3 to 10. Among these experiments, we find cluster 4, 5, and 8 outperform other choices. In particular, cluster number 5 is the most suitable to our diagram domain, align with our original labeling. The optimal clustering results are discussed in Section 6.5.1. The extensive interpretation of experiments with different cluster numbers are in Section 6.5.2.

We examine the effect of different starting points of clustering by taking a set of KMeans (Euclidean and Manhattan distance metrics) clustering experiments with different randomized starting points. With eight different random seeds to select starting points, we cluster the diagrams into 4, 5 and 8 clusters respectively, and summarize their rates of of correctly clustered diagrams in Table 6.14. The rate of correctly clustered diagrams is calculated by choosing the dominant class label to be a cluster’s label, and calculating the proportion of the dominant diagrams among all the diagrams in this cluster.

We have totally more than 700 diagrams. We pick one of the eight experiments as a representative for this cluster number. For cluster number 4, we can pick the experiment with the second random seed (correctly clustered rate 41.2%) to represent all of the eight experiments. Without lose of generality, we pick the one with lowest incorrectly clustered rate.

6.5.1 The five diagram classes used for clustering

The clustering results of a set of KMeans (Manhattan distance metric) and EM experiments are illustrated in Fig. 6.3. We set both of them to cluster diagrams into 5 groups, aiming at comparing the natural groups sought by the computer and class labels assigned by human.

Among the five diagram classes, bar chart and tree are the most distinguishable diagrams noticing the fact that 69% of cluster 1 in EM experiment (Fig. 6.3(b)) are bar chart, and that 68% of cluster 5 in KMeans experiment (Fig. 6.3(a)) are tree. Moreover, in cluster 1 in
6.5. CLUSTERING

<table>
<thead>
<tr>
<th>Cluster No. in SimpleKMeans</th>
<th>Incorrectly clustering rate with different random seeds</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Seed1</td>
</tr>
<tr>
<td>SimpleKMeans with Euclidean Distance</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>41.9</td>
</tr>
<tr>
<td>5</td>
<td>44.9</td>
</tr>
<tr>
<td>8</td>
<td>40.6</td>
</tr>
<tr>
<td>SimpleKMeans with Manhattan Distance</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>47.3</td>
</tr>
<tr>
<td>5</td>
<td>48.7</td>
</tr>
<tr>
<td>8</td>
<td>43.9</td>
</tr>
</tbody>
</table>

Table 6.14: To examine the influence of starting instances to clustering performance, we run two sets of SimpleKMeans experiments, with cluster number 5 and 8 respectively. In each set of experiments, we start with 8 different sets of starting instances. The clustering results are evaluated with their labels, and the rates of incorrectly clustered instances are listed in the table. All the values in the tables are percentages.

KMeans experiment (Fig. 6.3(a)), although bar chart only occupy 52% of this cluster, it is clear that the majority of bar charts in the entire data set grouped into this single cluster. Actually this cluster contains 72% bar charts of the entire data set.

Tree diagrams are found even more distinctive by clustering algorithms. In cluster 5 of (Fig. 6.3(a)), the extremely dominating diagrams, i.e., 68% of the cluster, are trees. It shows that trees are found significantly better than other diagrams, especially data curve diagrams noticing only 3 data curve diagrams are clustered in this cluster.

Data curve, data line, and data point diagrams are found similar with each other by the clustering algorithms. Each of the five clusters contains similar numbers of these kinds of three diagrams. For instance, in cluster 2 of EM experiment (Fig. 6.3(b)), the numbers of these three diagram classes are 52, 36 and 38 among the totally 162 diagrams.
Figure 6.3: The Clustering results of KMeans (Manhattan distance metric) and EM algorithms. (a) is KMeans with cluster number 5, (b) is EM with cluster number 5. Bar chart and Tree diagrams are found to be the easiest ones to be clustered. The majority diagrams in the first clusters are bar charts. Those in the second clusters are data curve diagrams. Those in the third clusters are data line diagrams. Those in the fourth clusters are data point diagrams. Those in the fifth clusters are tree diagrams.
6.5.2 Comparison of SimpleKMeans and EM

To compare KMeans and EM, we assign the label of the dominate class in a cluster to the entire cluster as the cluster label. For example, in Table. 6.15.(a), bar chart is assigned to the cluster of the first column since there are 137 bar charts in this cluster. We notice that the label assignment may bring some confusion, e.g., in Table. 6.15.(b), data curve and data point diagrams similarly dominate the second cluster with 56 and 54 instances respectively.

In general, KMeans and EM perform comparably to each other, noticing their correctly clustered rates are similar in Table. 6.15 to Table. 6.17. Take bar chart diagram as an example, KMeans effectively clusters 137 out of 181 bar charts into one group with cluster number 4 (Table. 6.15.(a)), 134 with cluster number 5 (Table. 6.16.(a)), and 123 with cluster number 8 (Table. 6.17.(a)). EM successfully clusters 114, 104 and 103 bar chart into one group with cluster number 4, 5, and 8 respectively (Table. 6.15.(b), Table. 6.16.(b), and Table. 6.17.(b) respectively). KMeans and EM perform similar with each other in clustering data curve, data line and data point diagrams.

However, tree diagrams are clustered differently with different cluster numbers. Take KMeans as an example, small cluster numbers outperform large ones. With cluster number 4 (Table. 6.15), 92 out of 147 tree diagrams are clustered into one group. With cluster number 5 (Table. 6.16), 86 are clustered. With cluster number 8 (Table. 6.17), only 75 are clustered together.
(a) SimpleKMeans with cluster number 4 (correctly clustered rate 47.7%)

(b) EM with cluster number 4 (correctly clustered rate 38.8%)

Table 6.15: Comparison of the confusion matrices of SimpleKMeans and EM clustering with cluster number 4. The labels of the clusters are assigned with the labels with the maximal instances in the cluster.
### Table 6.16: Comparison of the confusion matrices of SimpleKMeans and EM clustering with cluster number 5.

(a) SimpleKMeans with cluster number 5 (correctly clustered rate 48.7%)

<table>
<thead>
<tr>
<th>Class</th>
<th>Clusters</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar chart</td>
<td></td>
<td>134</td>
<td>19</td>
<td>10</td>
<td>5</td>
<td>13</td>
</tr>
<tr>
<td>Data curve</td>
<td></td>
<td>15</td>
<td>62</td>
<td>16</td>
<td>8</td>
<td>3</td>
</tr>
<tr>
<td>Data line</td>
<td></td>
<td>20</td>
<td>32</td>
<td>34</td>
<td>14</td>
<td>16</td>
</tr>
<tr>
<td>Data point</td>
<td></td>
<td>56</td>
<td>44</td>
<td>21</td>
<td>27</td>
<td>9</td>
</tr>
<tr>
<td>Tree</td>
<td></td>
<td>33</td>
<td>12</td>
<td>9</td>
<td>7</td>
<td>86</td>
</tr>
<tr>
<td>clustered:</td>
<td></td>
<td>Bar chart</td>
<td>Data curve</td>
<td>Data line</td>
<td>Data point</td>
<td>Tree</td>
</tr>
</tbody>
</table>

(b) EM with cluster number 5 (correctly clustered rate 46.3%)

<table>
<thead>
<tr>
<th>Class</th>
<th>Clusters</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bar chart</td>
<td></td>
<td>104</td>
<td>27</td>
<td>20</td>
<td>18</td>
<td>12</td>
</tr>
<tr>
<td>Data curve</td>
<td></td>
<td>0</td>
<td>52</td>
<td>28</td>
<td>15</td>
<td>9</td>
</tr>
<tr>
<td>Data line</td>
<td></td>
<td>5</td>
<td>36</td>
<td>53</td>
<td>7</td>
<td>15</td>
</tr>
<tr>
<td>Data points</td>
<td></td>
<td>11</td>
<td>38</td>
<td>27</td>
<td>48</td>
<td>33</td>
</tr>
<tr>
<td>Tree</td>
<td></td>
<td>31</td>
<td>9</td>
<td>29</td>
<td>9</td>
<td>69</td>
</tr>
<tr>
<td>clustered:</td>
<td></td>
<td>Bar chart</td>
<td>Data curve</td>
<td>Data line</td>
<td>Data point</td>
<td>Tree</td>
</tr>
</tbody>
</table>
Table 6.17: Comparison of the confusion matrices of SimpleKMeans and EM clustering with cluster number 8.

(a) SimpleKMeans with cluster number 8 (correctly clustered rate 43.7%)

<table>
<thead>
<tr>
<th>Class</th>
<th>Clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Bar chart</td>
<td>123</td>
</tr>
<tr>
<td>Data curve</td>
<td>2</td>
</tr>
<tr>
<td>Data line</td>
<td>4</td>
</tr>
<tr>
<td>Data point</td>
<td>14</td>
</tr>
<tr>
<td>Tree</td>
<td>31</td>
</tr>
</tbody>
</table>

(b) EM with cluster number 8 (correctly clustered rate 42.9%)

<table>
<thead>
<tr>
<th>Class</th>
<th>Clusters</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1</td>
</tr>
<tr>
<td>Bar chart</td>
<td>103</td>
</tr>
<tr>
<td>Data curve</td>
<td>11</td>
</tr>
<tr>
<td>Data line</td>
<td>27</td>
</tr>
<tr>
<td>Data point</td>
<td>19</td>
</tr>
<tr>
<td>Tree</td>
<td>5</td>
</tr>
</tbody>
</table>

Table 6.17: Comparison of the confusion matrices of SimpleKMeans and EM clustering with cluster number 8.
Chapter 7

Summary and Future Work

This chapter summarizes our thesis, highlights our contributions, and explains the future directions of our thesis.

This thesis presents our research on the feature analysis and classification of vector diagrams in PDF documents. PDF is a major document format for electronic articles, and diagrams carry a huge amount of information. Our major contribution is discovering content features, graphemes, that can represent the diagram semantically. Another contribution is acquiring insight of the features through the machine learning experiments. The experiment results shows that our research has built a solid foundation for a diagram retrieval system, which will be our future work.

In our work, we have gained deep insights into the structure of vector diagrams through this fundamental research, a major step towards content-based diagram retrieval systems in the future [51, 24, 52]. This thesis demonstrates that we can successfully classify five diagram classes by extracting a set of content features, graphemes, of vector diagrams, and applying machine learning techniques to the features.

Our research has promising prospects of future research and application potential with the rapid progress in electronic publishing and the quickly growing number of vector diagrams online. Moreover, our research will not be restricted to vector diagrams alone, it can also be applied to the traditional raster diagrams after vectorizing them using high-quality vectorization algorithms.
Appendix:

How to reproduce and extend the results of this thesis

The many steps that would be required to reproduce and extend the results in this thesis are described in the various sections. This appendix gives a step-by-step outline of what those steps are, with references to the sections where the reader can find more details. A fair amount of the code developed for this work is rather complex, so we would suggest that people interested in pursuing this get the code from the author or Futrelle’s lab.

8.1 Diagram corpus

8.1.1 Collecting and filtering the PDF papers

The papers used in this work came from journals in the BMC Series. All of the papers in these journals have a standard layout for the PDF command structure. Papers from the BMC "independent journals" vary in their PDF format, so they were not used. A simple HTTP client package was used to download sets of journals. A typical PDF paper has a unique URL such as, http://www.biomedcentral.com/content/pdf/1471-2407-7-237.pdf. The URL of the JPEG files of the diagrams in the articles can be easily parsed from the article URL.

The PDFs are processed by the open-source Etymon PJ system that parses PDF files, turning each PDF document into a sequence of Java objects, one object per PDF command. At that point, the number of line and curve graphics primitives on each document page is
CHAPTER 8. APPENDIX: HOW TO REPRODUCE AND EXTEND THE RESULTS OF THIS THESIS

counted. A useful cut-off for the counts turned out to be 80 - any page with more than 80
lines + curves was assumed to be a diagram. (Lower values often corresponded to tables.)
To free our system from dependency on the Etymon API the Etymon classes and their
 corresponding were mapped to an API of our own design. The PDF command sequence
contains sub-sequences corresponding to paths, e.g., which denote the boundary of a region;
these were easily mapped.

More details of data source can be found in Section 4.1.

8.1.2 The interpreter and the creation of self-contained graphics objects

Another aspect of the PDF command structure is the use of a graphics state, and more
importantly, operations which alter, and push and pop the graphics state. It is critical
that the graphics state be tracked, most importantly because the coordinate system can be
altered. Therefore, an interpreter was written to do this. It is not suggested that a new user
try to write such an interpreter. The author will gladly share it and other code.

The PDF contains command sequences of the form move to x1, y1 followed by draw to x2,
y2 commands rather than commands which fully specify the line, e.g., drawline(x1,y1,x2, y2).
By converting command sequences to the equivalent of single fully specified commands can
create a line as a fully specified and self-contained object. This changes the PDF command
sequence from a procedural specification to an object-based specification. The result of the
interpreter process and the conversion to objects is a collection of static objects each with
the final coordinates and dimensions that appear in the diagram. The object collections are
persisted using Java serialization.

More details of PDF specification and graphics in PDF can be found in Section 4.2 and
4.4. Figure 4.2 illustrates the structure of a PDF page.

8.1.3 Building a spatial index for the objects

An object such as a line maps to locations on the plane. But it is important to invert
this map, so that given a region on the plane, the objects that it contains or that pass
8.2 DETECTING GRAPHEMES AND CREATING FEATURE SETS

through it can be found directly without search. The inverted map is called a spatial index. The spatial index is an inverted index, mapping from locations to objects. The objects are rendered into the spatial index much as objects are rendered to a pixel view. However, the spatial index is coarser than a full pixel-level structure, typically a $22 \times 28$ array of spatial cells. A typical use of the spatial index would be in finding tick marks - when a long line and a perpendicular short line are found in the same cell, then detailed computations are done to see if they actually touch one another. More details of spatial index is explained in Section 4.5. Figure 4.5 shows a section of the spatial index.

8.2 Detecting graphemes and creating feature sets

Primitive graphemes are found by examining the collection of objects in a diagram. Complex graphemes such as tick marks or triangular data points use the spatial index to zero in on the two or three lines that make comprise them. For example, triangular data points are discovered by finding a closed path made up of three short line segments of equal length. Triangles with edge lengths less than 10 points were considered to be data point markers.

All feature values are based on a counts of each grapheme type. Because of the wide variation in counts the logs of the counts were used as the feature values in all subsequent machine learning analyses. Graphemes are explained in detail in Chapter 5.1.

8.3 The machine learning experiments

Labels are manually assigned to diagrams into five classes: bar chart, data plot of curves, data plot of data points, data plot of data line, and tree diagrams. These five classes are the ones most distinguishable to human vision. Weka was chosen for the machine learning experiment platform. Weka is an open source data mining software system that implements a variety of machine learning algorithms. Details of machine learning experiments are in Chapter 6.
8.3.1 Supervised learning experiments

Supervised learning experiments are taken on the data set consisting of grapheme feature set and labels. The supervised learning algorithms include: decision tree C4.5, naive Bayes, AdaBoost.M1 (decision stump as base learner), and AdaBoost.M1 (C4.5 as base learner). For C4.5, the minimum number of objects on a leaf is 2. For AdaBoost.M1, the iteration of boosting is 10. The hypotheses are tested in 10-fold cross validation. The analysis of supervised learning experiments are explained in Chapter 6.3.

8.3.2 Unsupervised learning experiments

Two unsupervised learning algorithms are selected: KMeans and Expectation Maximization (EM). The number of clusters varies between 3 and 10. The starting points of each iteration are randomly chosen. For KMeans, distance metrics include Euclidean and Manhattan distances. The experiments apply 66% of the data set as training data and the rest as testing data. The Unsupervised learning experimental results are analyzed in Chapter 6.5.

8.3.3 Diagram retrieval experiments

Diagram retrieval can be explored by ranking the diagrams in a sequence that the top k are the nearest to the query diagram. The distance metric applied is Euclidean distance. The retrieval analysis is in Chapter 6.2.


[57] Uc irvine machine learning repository.


