Undergraduate Research Opportunity Program

(UROP) Project Report

Non-photographic Image Categorization

By

Wang Fei

Department of Computer Science

School of Computing

National University of Singapore

2004

Undergraduate Research Opportunity Program

(UROP) Project Report

Non-photographic Image Categorization

By

Wang Fei

Department of Computer Science

School of Computing

National University of Singapore

2004

Project No.: U79090

Advisor: Dr. Min-Yen Kan

Deliverables:

Report: 1 volume

Program: 1 CD

Abstract

The rapid growth of IT industry today has undoubtedly boosted the widespread use of computer images in both web pages and modern computer programs. Applications like online image search, automatic webpage summarization and web mining, rely heavily on image categorization. This project presents a system that categorizes non-photographic images[1] based on their textual and image features. The correlation between image category and each of these features is also examined. Some example categories defined in this project include map, diagram, icon, artwork and cartoon. With the machine learning program "BoosTexter" and a large set of training data obtained from Google^{TM}, the system is able to correctly categorize images up to an accuracy of around 97.6%.

Subject descriptors:

I.2.6 Learning

I.4.10 Image Representation

I.5.2 Design Methodology

Keywords:

Image categorization, textual features, image features, feature extraction, machine learning.

Implementation software and hardware:

Perl, Java, MS-Windows, UNIX, BoosTexter

[1] Since non-photographic images are the main focus of the project, the term "images" will be used interchangeably with "non-photographic images" in the rest of the report.

Content page

1. Introduction

Images are an important means for recording and presenting visual information. A key differentiation between the World Wide Web and earlier precursors is the presence of images which makes the surfing experience more interesting and interactive.

While digital cameras have made photographic pictures an important category of images, images created entirely by digital means are growing in importance. Software developments have facilitated their creation, for purposes ranging from the visualization of raw data and artwork and artificial images used in games and entertainment. Thus the accurate classification of these non-photographic images – such as icons, maps, diagrams and charts – has become increasingly important. Such a system has many practical applications. As an example, online image search engines can be enhanced with the categorization feature to return more accurate results for queries.

The product of this thesis is a system that fulfills this need by performing non-photographic image classification (NPIC). The implemented system uses semi-supervised machine learning to create a time-efficient classifier. The system assigns an image to one of the five pre-defined categories, and achieves an overall accuracy of around 97.6% for any randomly given non-photographic image.

The thesis examines the implemented NPIC system in detail. Chapter 2 presents some background knowledge and related works. Chapter 3 discusses the various image categories and features that are used to categorize them. Chapter 4 presents the methodology for feature extraction and details the use of machine learning and cross validation. A thorough system evaluation and analysis is presented in Chapter 5. Chapter 6 concludes the thesis and suggests a future work plan.

2. Background

Previous attempts in building image categorization systems have been made; however, some of them have very general classification schemes. Lienhart and Hartmann (2002) implemented and evaluated a system that performs a two-stage classification of images: first, distinguishing photo-like images from non-photographic ones, followed by a second round in which actual photos are separated from artificial, photo-like images, and non-photographic images are differentiated into presentation slides, scientific posters and comics. The scheme is neither exclusive nor exhaustive, *i.e.*, there are many images that fall into none of the above-mentioned categories, and therefore, cannot be categorized.

Current online image search engines, like the one used by Google^{TM}, retrieve relevant images based on textual hints such as the image's filename, alternate (ALT) tag text, and the surrounding text on the webpage in which the image is found. Although this technique is quite effective, it is not robust in cases where the image is not identifiable by these features, yet is easily associated with a category by the image's visual features.

2.1 Difficulties involved in image categorization

Attempts have always been made in developing an automatic image categorization system though not many of them can do so with high accuracy. Categorizing images is not easy, especially when it involves both textual and image features.

As a matter of fact, even if this is done independently by two human annotators, they may put the same image into different categories, especially when they are using a complicated classification scheme. This is because it is not easy to define categories into which any given image fits nicely. In addition, sometimes, it may be the case that an image possesses features of two or more categories, in which case, a single category classification is not possible.

2.2 My approach

A non-photographic image categorization (NPIC) system is proposed in this project. It deals only with non-photographic images and makes use of 36 features, combining both textual and image ones.

The NPIC is a system that successfully achieves automatic general image categorization. The system uses machine learning trained on a large set of automatically acquired data and results in accurate and reliable categorization. An analysis of the results shows an expected increase in test accuracy as larger training datasets are used.

In addition, the NPIC system overcomes the restriction that an image must reside on the Internet and be given in the form of a URL for it to be categorized. In other words, both “local” and “web” images can be processed by the system.

3. System definition and design

3.1 Image categorization scheme

In previous work, various image categorization schemes have emerged. Gorkani and Picard (1994) proposed a method for distinguishing city/suburb from country/landscape scenes using the most dominant orientation in the image texture. Yiu (1996) classified images into indoor and outdoor scenes using color histograms and texture orientation.

In this project, a three-level categorization scheme is proposed (as shown in Figure 1).

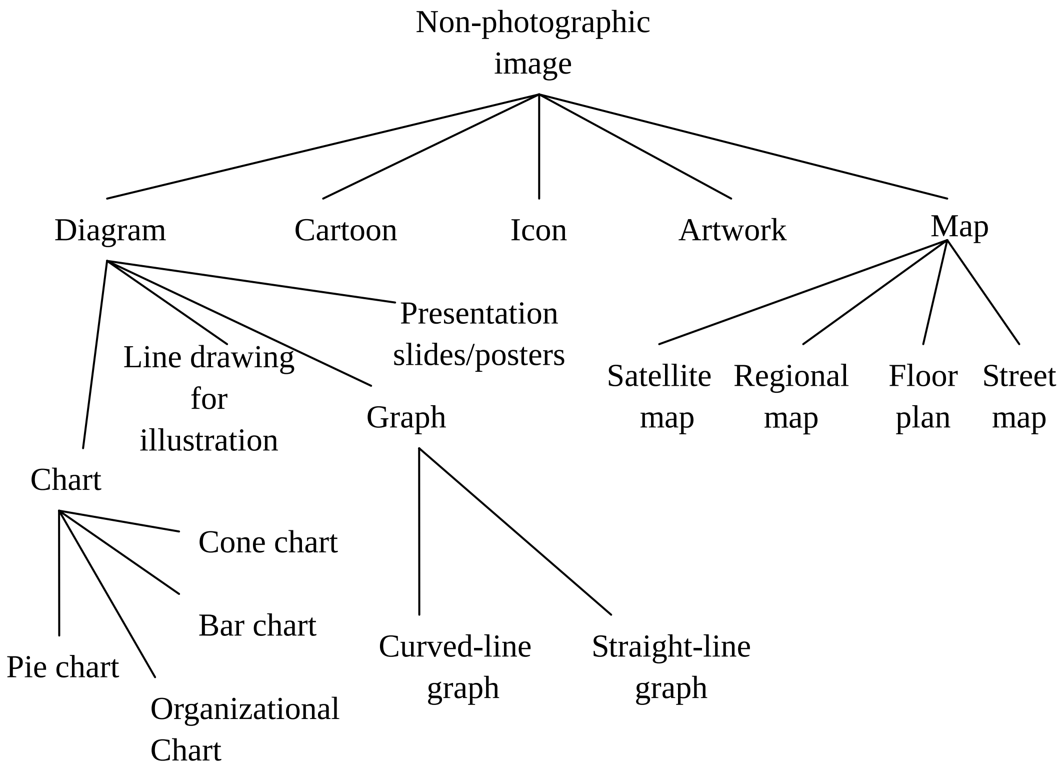

Figure 1. Image classification scheme

To come out with the above image categorization scheme, I have done a thorough study on a large number of non-photographic images on the web. My first attempt was a scheme where the first level was too detailed as it contained many of the second or even third-level categories found in the scheme presented in Figure 1; then it has been realized that depth, rather than breadth, is preferred in the categorization scheme design. This is because the system is expected to perform better if there are less available categories to choose from at each particular level. For instance, it is simpler for the system to determine that an image is a diagram rather than one of its subcategory, a cone chart.

Besides, a good categorization scheme is one that is exclusive and exhaustive, *i.e.* any given image falls into one and only one category in each level of the scheme. To achieve this, I had to verify that a randomly found image can be successfully categorized by the scheme. New categories were discovered and existing redundant categories were eliminated in this process.

In addition, although the categorization scheme as presented in Figure 1 has been designed, I have decided to pursue a prototype NPIC that only classifies images into the five categories in the first level, *i.e. icon*, *diagram*, *map*, *artwork* and *cartoon*. Higher-level categorizations may be possible with more features and larger training datasets; but my main objective in this thesis is to create a proof of concept NPIC system.

3.2 Examples and descriptions of categories

3.2.1 Icon

Icons are defined as small images (quite often square in shape) that have special uses. Examples include operating system icons, website navigation buttons and website logos. They are very distinct from images in other categories because most of them have small physical dimensions and file sizes due to their low image quality and resolution. A key differentiating feature of icons is that many of them have the file extension “.ico”, which makes them easily identifiable. Therefore, a high degree of accuracy is expected for this category.

Below are some example icons.

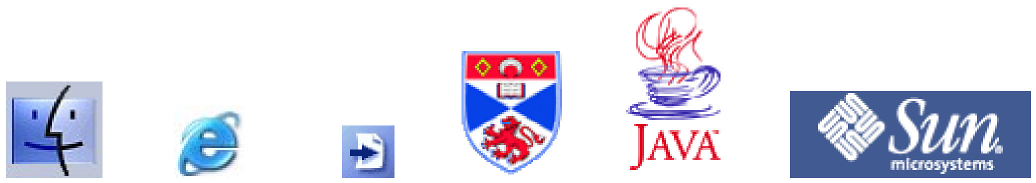

Figure 2. Example icons

3.2.2 Diagram

The category “diagram”[1] defined in this project includes non-artistic images created by image creation software such as Adobe Photoshop©, Microsoft Paint© and Microsoft Excel©. They are usually used for presentation, illustration and demonstration purposes. Charts, scientific posters, graphs, presentation slides are some examples of this category.

Diagrams are hypothesized to possess the following characteristics:

- Text blocks/regions are often found in diagrams.
- Diagrams like presentation slides usually have headings located in the topmost regions.
- Straight and curve lines are common.
- They usually have fewer colors than image of other categories.

Some example diagrams are shown here.

[1] The category “diagram” used here refers to diagrams in a broader sense, as described above.

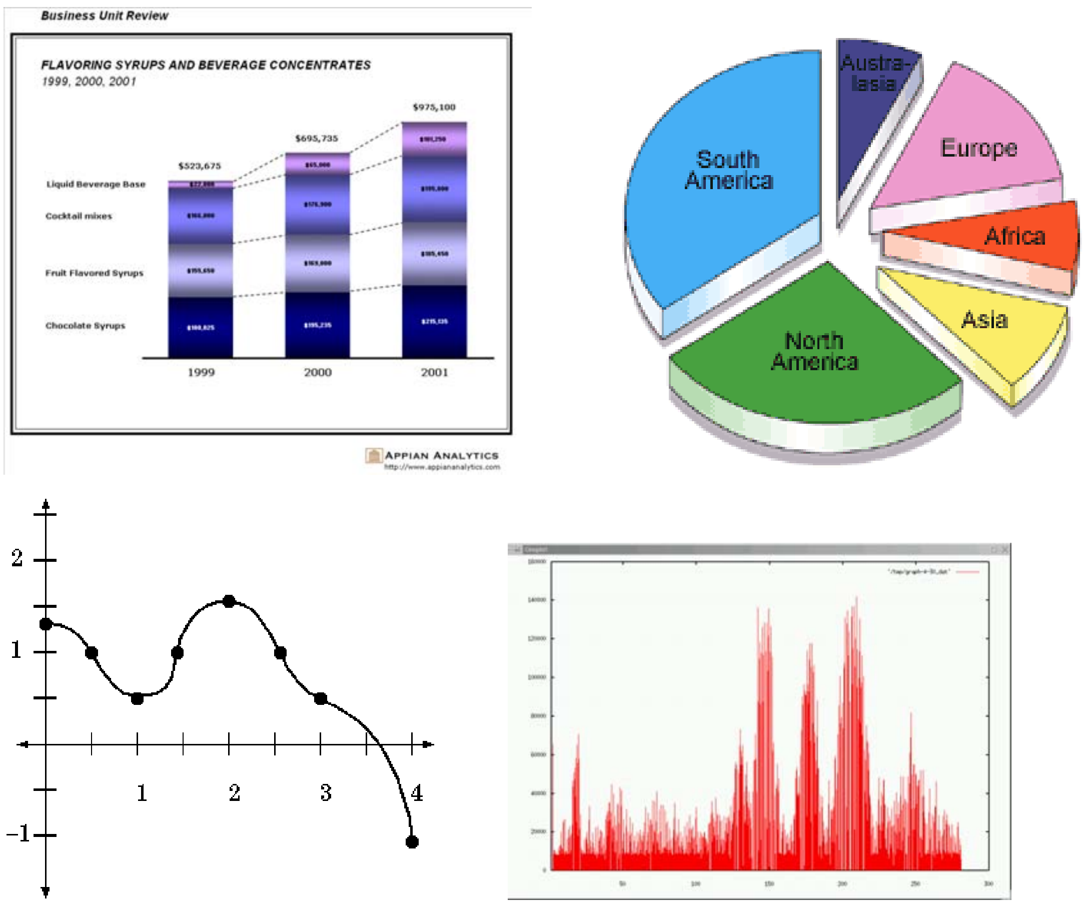

Figure 3. Example diagrams

3.2.3 Map

Maps are quite straightforward to define. They typically consist of a large number of text features and single-colored blocks, and usually have larger dimensions and resolution. Some examples are street maps and floor plans, as shown below.

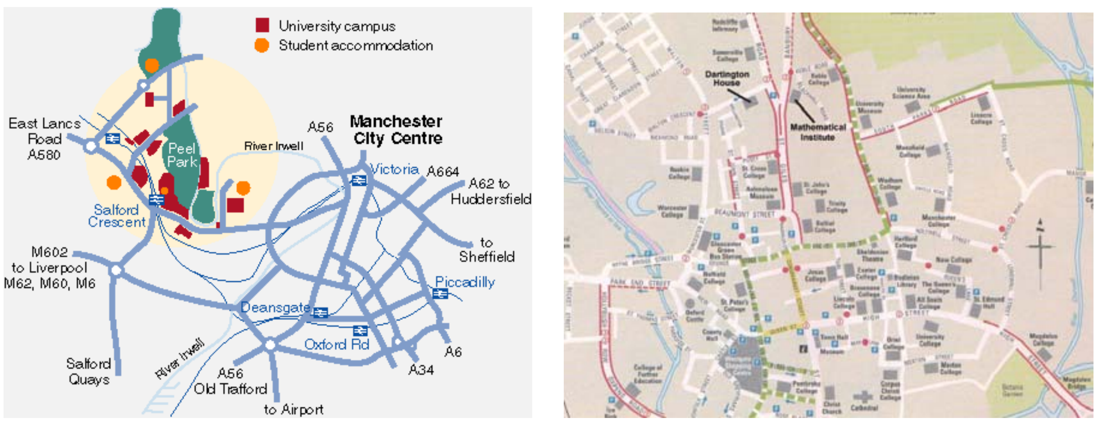

Figure 4. Example maps

3.2.4 Artwork

Artwork, as opposed to diagrams, refers to artistic drawings and computer graphics like screen captures. Generally they contain a large number of colors than diagrams and few blocks of the same color can be found.

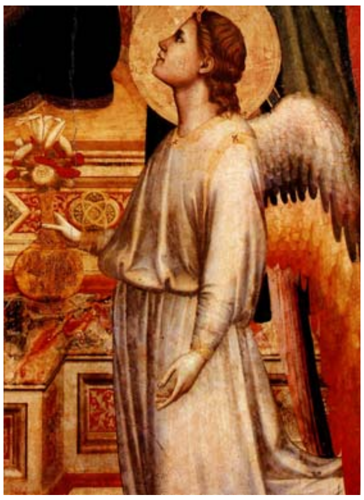
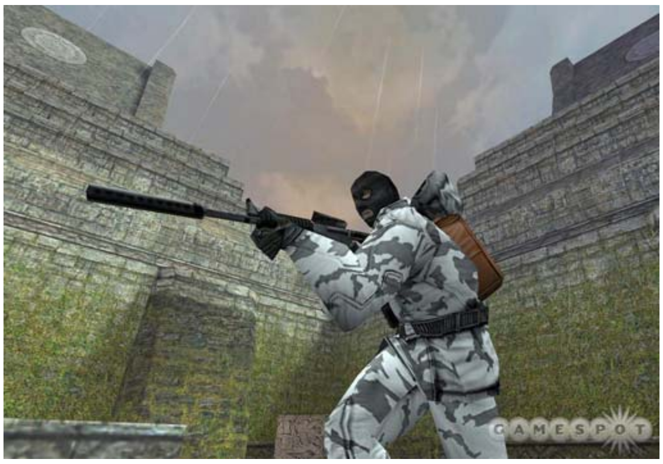

Figure 5. Example Artwork

3.2.5 Cartoon

Cartoon images are comic strips that are usually taken from the web or scanned from comic books. Often divided into rectangular blocks by vertical and horizontal borders, cartoons are sometimes seen in the form of grayscale images since many of them are acquired from “black-and-white” comic book scans. Those in color contain many large single-colored regions as cartoon images are simple color drawings and they do not have many inter-crossing lines like maps. Two cartoon examples are shown here.

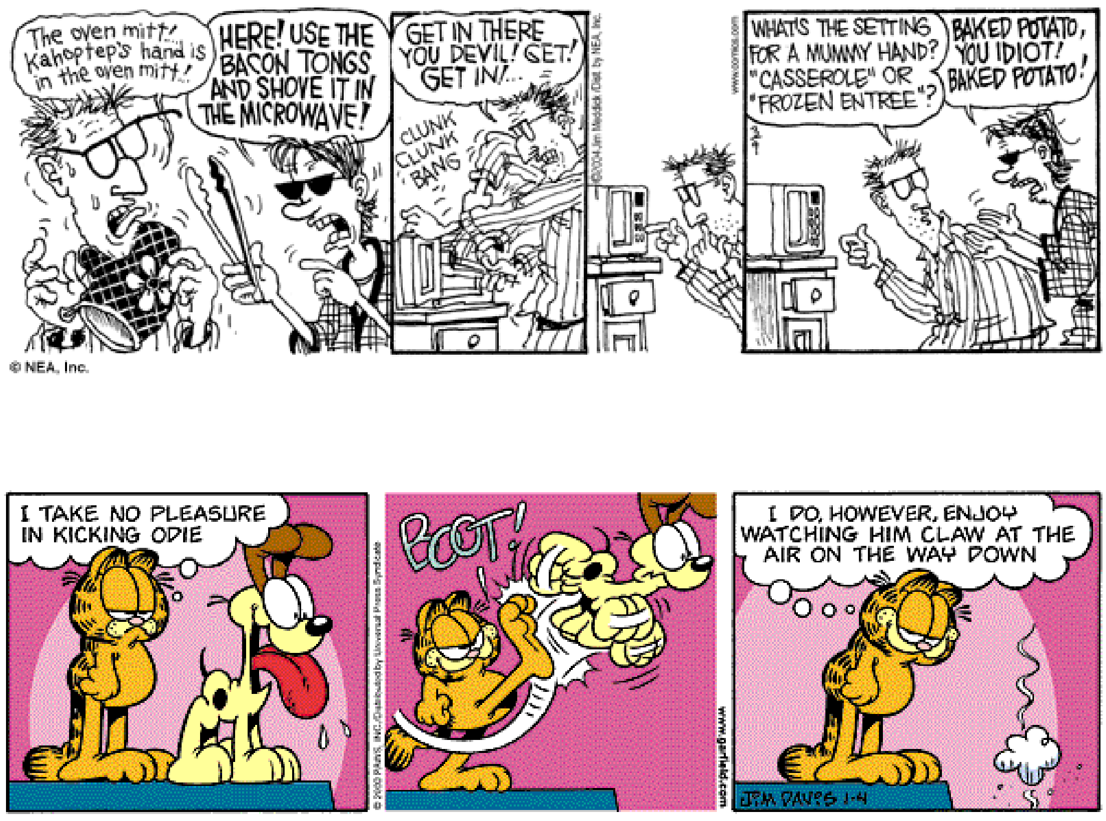
The oven mitt! Kahoptep's hand is in the oven mitt!
HERE! USE THE BACON TONGS AND SHOVE IT IN THE MICROWAVE!
GET IN THERE YOU DEVIL! GET! GET IN!...
CLUNK CLUNK BANG
WHAT'S THE SETTING FOR A MUMMY HAND? "CASSEROLE" OR "FROZEN ENTREE"?
BAKED POTATO, YOU IDIOT! BAKED POTATO!
© NEA, Inc.
I TAKE NO PLEASURE IN KICKING ODIE
BOOT!
I DO, HOWEVER, ENJOY WATCHING HIM CLAW AT THE AIR ON THE WAY DOWN
www.garfield.com
JIM DAVIS 1-4

Figure 6. Example cartoon

3.3 A brief description of the system as a whole

As mentioned before, the proposed NPIC system relies on a machine learner and a large dataset to categorize images. A machine learning program called "BoosTexter", developed by ImageMagick^{TM}, is used in this system. Basically, it works by first building up a decision tree of rules based on the training examples, and then, makes predictions on the testing examples. Each example is made up of a list of attribute values. Two types of attributes are used is this system, *i.e.* continuous attributes and textual attributes. For instance, image's height is a continuous attribute whose value might have some correlation with the image category. BoosTexter will first find a threshold, and then construct a rule, say, "images with height less than 100 pixels are likely to be icons".

To train BoosTexter, a large set of annotated images is needed. Some random 5000 images are acquired from Google image search[1], which creates a noisy corpus that has not been pre-annotated by humans. This is to say, the categories of images as assigned by Google image search are used in the training and test data. A hypothesis is made in Section 3.6 and a reliability test is conducted in Section 5.1.

3.4 Features used as classifiers

A detailed description of all image and textual features used in categorization is presented in the table below (Table 1). Although a total of 36 features are used by BoosTexter, some are just different representations of the same image property. For instance, three types of color encoding are used, *i.e.* RGB in six digit hexadecimal form (*e.g.* #fa3ce0), RGB truncated to three hexadecimal digits (#f3e for the same example) and hue value (308 for the same example). It is BoosTexter's task to decide which representation can be used to create a better classifier. This is further explained in the table below (Table 1).

Feature name		Feature type[2]	Description
Textual features	Image filename	Text	Image filename without extension.
	Filename extension	Text	e.g. jpg, gif, png, ico, etc.
	File size	Continuous[3]	File size is expressed in terms to bytes.
	Height	Continuous	Image dimensions in pixels.
	Width	Continuous	
	X-resolution	Continuous	Resolution represents the quality of an image and it is expressed in pixels-per-inch.
	Y-resolution	Continuous	
	Comments	Text	Few images have comments.

[1] Refer to Chapter 4 for a detailed description on implementation.
[2] The feature type here refers to the data type as used in BoosTexter.
[3] Please note that the field type "continuous" here refers to set of real numbers.

	Image URL	Text	URL of image where the image is found.
	Page URL	Text	URL of webpage where the image resides.
Image Features	Number of colors	Continuous	Number of distinct colors in an image.
	Color depth	Continuous	Number of bits used to represent one of the three components of RGB, e.g. color depth of 8 means 24 bits-per-pixel.
	Interlace	Text	e.g. none, plane
	Background color	Continuous/ Text	Each of these three features is represented in three different forms, *i.e.* hex6 (text), hex3 (text) and hue (continuous) as described before. The former two are just used by BoosTexter to discover repeated patterns, but the latter is hypothesized to possess similar values for visually similar colors. This will be elaborated in Section 4.2.2.
	Matte color	Continuous/ Text	
	Most prevalent color, c_1	Continuous/ Text	
	Fraction of pixels with c_1	Continuous	Equals to the number of pixels having the prevalent color / total number of pixels.
	f1, fraction of pixels with the neighbor metric[1]	Continuous	The neighbor metric denoted in the following four forms are used. $d1= \vert r_1–r_2 \vert + \vert g_1-g_2 \vert + \vert b_1-b_2 \vert$

[1] Neighbor metric is a measure of the color distance between two neighbor pixels (Lienhart and Hartmann, 2002).

	greater than zero		d2= $\mid h_1–h_2 \mid + \mid s_1-s_2 \mid + \mid v_1-v_2 \mid$
			d3= $\mid h_1–h_2 \mid + \mid s_1-s_2 \mid$
			d4= $\mid h_1–h_2 \mid$

	f2, fraction of pixels with the neighbor metric greater than a high threshold	Continuous	Represented in similar formats as f1. In this system, ¼ of the maximum is chosen as the threshold because that results in noticeable values of f2.
	f1/f2	Continuous	The ratio between f1 and f2.

Table 1. Description of features used in the system

All the text fields are tokenized into individual words by characters like “ ”, “_” and “-” that are common in filenames and URLs. For instance, a filename “Singapore_map” is broken down into two words, “Singapore” and “map”. This might make BoosTexter classify this image as map if BoosTexter, through constructing decision tree from a dataset, decides that the word “map” appearing in the filename field is a good indication to the category of maps.

It should be noted that for some images, certain features cannot be extracted (or are simply non-existent), in which case a blank (“ ”) and a number zero are used for text and continuous fields respectively as a replacement. These features are essentially omitted in the resulting feature vector.

3.5 An example feature vector

Comma-delimited feature vectors are used by BoosTexter as training data. In this section, an image found at http://home.planet.nl/~monstrous/image/tc_torques.png is used as an example. Figure 7 shows the image in its full size.

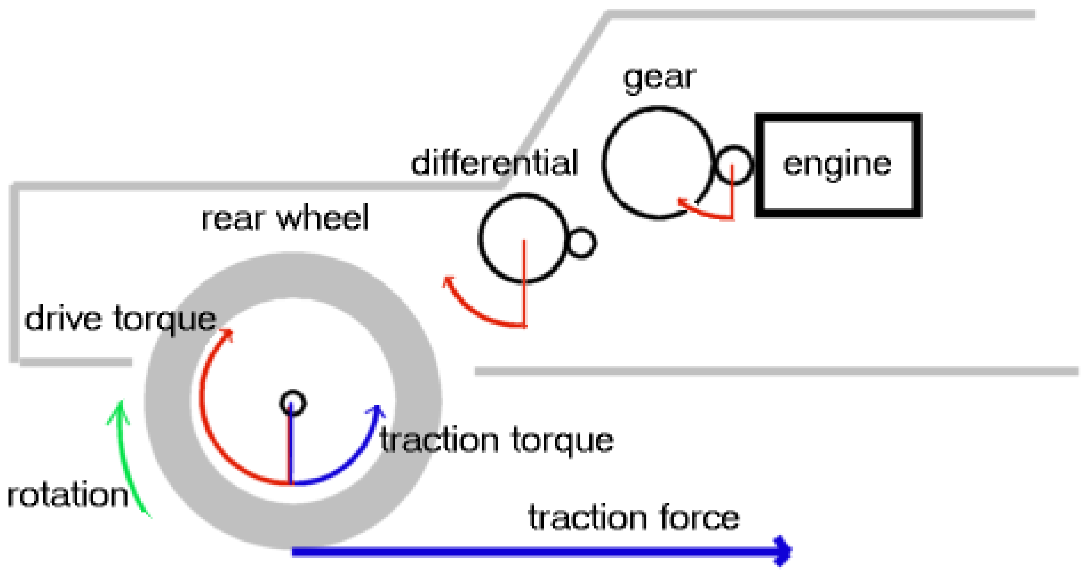

Figure 7. An image example,
tc_torques.png

The feature vector of the above image is shown in Figure 8. It can be seen that the fields conform to the order and format as described in Section 3.4. It can be observed that the fields are separated from one another by commas with the last field being the hypothesized category[1]. In this example, the class "diagram" is assigned.

In addition, all the text fields in the feature vector have been tokenized and standardized to lower case. URLs (the second and third last fields) thus appear as a sequence of individual words.

[1] Refer to Section 3.6 for the hypothesis and Section 5.1 for a reliability test.

```
tc torques, png, 19604, 266, 485, 28.34, 28.34, 660, png, 16, , 8, none,
ffffff, fff, 300.00000, bdbdbd, bbb, 330.00000, fefefe, fff, 0, 0.84841,
0.91454, 0.91454, 0.91454, 0.92218, 0.02538, 0.02219, 0.03967, 0.05166,
0.02775, 0.02426, 0.04338, 0.05602, home planet nl monstrous image,
home planet nl monstrous tutcar html, diagram.
```

Figure 8. Content of the feature vector
of image tc_torques.png

3.6 Hypothesis on classification based on Google image search

Since the NPIC system relies on machine learning to classify images, a large dataset is required. Fortunately, Google image search is a good source to obtain images based on the given queries despite the existence of some noisy data. For instances, image returned to queries like “Venn” or “chart” will probably be diagrams, though some might be noisy data, *e.g.* the photo of the mathematician John Venn might be returned.

An exception is the icon category. A large reliable source of icons has been found on the web[1], their classification is therefore 100% accurate, *i.e.* all the images downloaded (in a zip file) are icons.

However, as a large set of images are used in the system, it is unrealistic to expect each one to be manually annotated. Therefore, in the implementation of the system, it is hypothesized that images retrieved in this way conform to their actual classifications. A reliability test of this hypothesis is given in Section 5.1 where a small sample of images in each category is taken and manually annotated, and then the annotation results are compared with the categorizations given by Google. This test agrees with the hypothesis to a high degree of accuracy.

[1] http://www.wrclub.net/down/listdown.aspx?id=610

4.

4. System implementation and methodology

The NPIC system is broken down into three dependent modules, namely image retrieval, feature extraction and cross-validation. They will be discussed in the three consecutive sections, 4.1, 4.2 and 4.3 in detail.

4.1 Image retrieval using Google image search queries

The process of retrieving the 5000 images from Google is done in two stages.

First, image URLs and image page URLs are obtained from Google image search based on query keywords. Basically, I use an automated program to send out search keywords to Google as queries, and then it extracts the addresses of the image and the webpage from the returned search results. This process is repeated to retrieve more addresses. They will be saved in a file together with the category name. An example line is shown in Figure 9.

```
(garfield OR tintin OR archie) AND (comic OR cartoon)
http://www.bregenznet.at/woal/grafik/g_lauf0.gif
http://www.bregenznet.at/woal/html3.html
```

Figure 9. An example of retrieved image and page URLs

As mentioned in Chapter 2, Google image search retrieves relevant images based primarily on textual features like image's filename, ALT tag and URL. However, image category-indicating words do not always appear in these textual features. For example, a cartoon whose filename is "garfield.jpg" found at http://www2.udec.cl/~fcardena/graf/garfield.jpg will be not returned by searching keyword "cartoon" though that image is indeed one.

Therefore, to extend the range and variety of retrieved images, keywords that are correlated with categories are used. For instance, the word "Venn" is often used to refer to a particular type of logic diagram, and images returned to this query may not contain the word "diagram" in any of its textual features. Searching using these keywords is likely to slightly decrease the test accuracy of the resulting system, as the category name (*e.g.* "diagram") – a word that is likely to discriminate images of its selfsame category – does not appear in the feature vector. However, this makes the system more robust: such a modification makes the system less dependent on instances drawn only from these category names. This extends system coverage so that it has a better chance of performing well when tested on random images. Table 2 lists some of the keywords used during image searching and retrieval. After the completion of this first stage, I have a list of URLs from Google that indicate the location of images that contain the respective image search keywords.

Category	Examples keywords
Map	"(world OR city OR street OR topographic) AND map", "floor plan", "carte" (French word for "map"), "ditu" (Chinese word for "map")
Diagram	"diagram", "pie chart", "Venn", "graph"
Artwork	"artwork", "painting", "artistic drawing"
Cartoon	"cartoon", "comics", "Garfield", "Calvin and Hobbes"
Icon	Icons are not obtained from Google image search[1].

Table2. Categories and keywords used in searching

The second stage involves the downloading of the images given their URLs. I retrieve all images that can be accessed (a timeout parameter set to 5 seconds as some URLs

[1] Icons were obtained from http://www.wrclub.net/down/listdown.aspx?id=610.

are broken and cause the program to stall for a long time before continuing).

Downloaded images are assigned a unique identifier and saved. I store the identifier and other information, like image and page URLs, along with the category names in a mapping file. An example line from the mapping file is shown below.

```
1699  tc_orques.png      http://home.planet.nl/~monstrous/image/tc_torques.png
http://home.planet.nl/~monstrous/tutcar.html      diagram
```

Figure 10. An example matching file entry

4.2 Feature extraction methodology

Various features are extracted from the images using programs contained in the ImageMagick and Perl libraries.

4.2.1 Extraction of textual features using "identify"

ImageMagick is a robust collection of tools and libraries that can be used to manipulate images in various formats. The "identify" program (with -*verbose* option on) describes the characteristics of images and outputs them in a simple text format. For example, running -*verbose* on the image "middle_east_95.jpg" yields the information presented in Figure 11. I extract the image feature values from the output and canonicalize them where necessary (e.g., "10kb" and "1.2mb" are converted to "10240" and "1258291" respectively).

```
Image: middle_east_95.jpg
  Format: JPEG (Joint Photographic Experts Group JFIF format)
  Geometry: 1443x1699
  Class: DirectClass
  Type: true color
  Depth: 8 bits-per-pixel component
  Colors: 261308
  Profile-iptc: 424 bytes
  Resolution: 200x200 pixels/inch
  Filesize: 423.3kb
  Interlace: None
  Background Color: grey100
  Border Color: #DFDFDF
  Matte Color: grey74
  Dispose: Undefined
  Iterations: 0
  Compression: JPEG
  Tainted: False
  User Time: 0.180u
  Elapsed Time: 0:01
```

Figure 11. Output of *identify -verbose*

4.2.2 Extraction of color features using Image::Xpm[1]

Other image features are derived from another utility, Image::Xpm, a widely-available perl library. This module enables accessing xpm image pixels given a pair of x and y coordinates. It makes possible the traversal of all pixels of an image through simple looping.

Image::Xpm accepts only images in xpm format. I store a converted copy of an image in xpm format for this purpose. With the Image::Xpm library it is easy to find the most prevalent color, c_1, by keeping a color counter for each unique color found in the target image and returning the color with the maximal count.

[1] The motivation to introduce some of the image features described in this section is due to Lienhart and Hartmann (2002).

Calculating features f1, f2 is more complicated. As explained in Table 1, I define the neighbor metric to be the color distance between two neighboring pixels. Four different forms have been proposed to calculate it, as shown in Figure 12. f1 and f2 are the fractions of pixels in the image with a distance metric greater than zero and 3/4 of the maximum achievable distance respectively.

$d1= | r_1 – r_2 | + | g_1 - g_2 | + | b_1 - b_2 |$

$d2= | h_1 – h_2 | + | s_1 - s_2 | + | v_1 - v_2 |$

$d3= | h_1 – h_2 | + | s_1 - s_2 |$

$d4= | h_1 – h_2 |$

where r = red, g = green, b = blue, and
h = hue, s = saturation, v = value

Figure 12. Four ways to calculate the distance metric

In the process of calculating f1 and f2, a total of $w(h – 1) + h(w – 1)$ comparisons have to be made since the distance between each pair of neighboring pixels has to be evaluated, where w and h are the width and height of the image respectively. As depicted in the figure below, 12 comparisons have to be made for a three-by-three image.

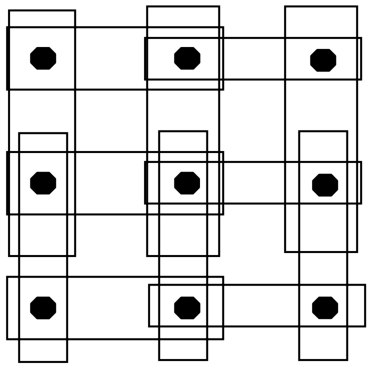

Figure 13. 12 comparisons have to been made for a three-by-three image.

The standard RGB format is normally used to represent the color of a pixel, and is usually expressed as a six-digit hexadecimal number. Each of the three primary colors, *i.e.* red, green and blue, is represented by two hexadecimal digits. However, the problem arises when comparing colors of different pixels represented in RGB format. For example, we have three pixels with colors #FFFFFF (white)，#FFFFFC (grey, but almost white) and #000000 (black). If they are simply taken as text field entries, the information that the first pixel has a very similar color as the second than the third is lost. This is because that these three values are mere encodings of colors, and BoosTexter is only able to find out a pattern if repeated values are detected.

On the other hand, if a color is to be represented in a one-dimensional scalar, what value should be used to represent the visual differences of different colors? The scheme whereby the scalar is calculated by adding up each of the three RGB values is does not prove to be a good. The two color encodings #0000FF and #FF0000 have exactly the same scalar value, *i.e.* 255, under this scheme despite the fact that they are visually very different because the first one is blue whereas the second one is red.

After some investigation into different color encoding schemes, the HSV color space scheme seems to be a suitable choice. This scheme defines a color using three different axes, *i.e.* hue, saturation and value. They are defined as follows[1]:

- Hue refers to a particular color within the visible spectrum, as defined by its dominant wavelength. Its value ranges from 0 to 360, and two pixels with similar hue values have similar colors as perceived by human beings.

- Saturation refers to the intensity of a specific hue. It is based on the color's purity, *i.e.* a highly saturated hue has a vivid intense color, while a less saturated hue appears more muted and grey. With no saturation at all, the hue becomes a shade of grey. Its value ranges from 0 to 1 as defined in the system.

[1] Definitions quoted from www.wordiq.com/definition/HSV_color_space.

- The third coordinate, “value”, is the brightness of a color. For example, the yellow color is brighter than the blue color, so it has higher “value” than blue. The “value” ranges from 0 to 1 as well.

During feature extraction, some of the output obtained from Image::Xpm is in RGB, and a conversion to HSV is made possible via the library Graphics::ColorObject. As shown in Figure 14, in the three-dimensional color space cone, the nearer any two chosen points, the more similar their colors are. However, compacting the three coordinates, H, S and V into a single scalar is non-trivial and beyond the scope of this project. On the other hand, hue, among the three components of HSV, gives the best measure of a color, hence, many features, like background color and the most prevalent color, are represented using hue values.

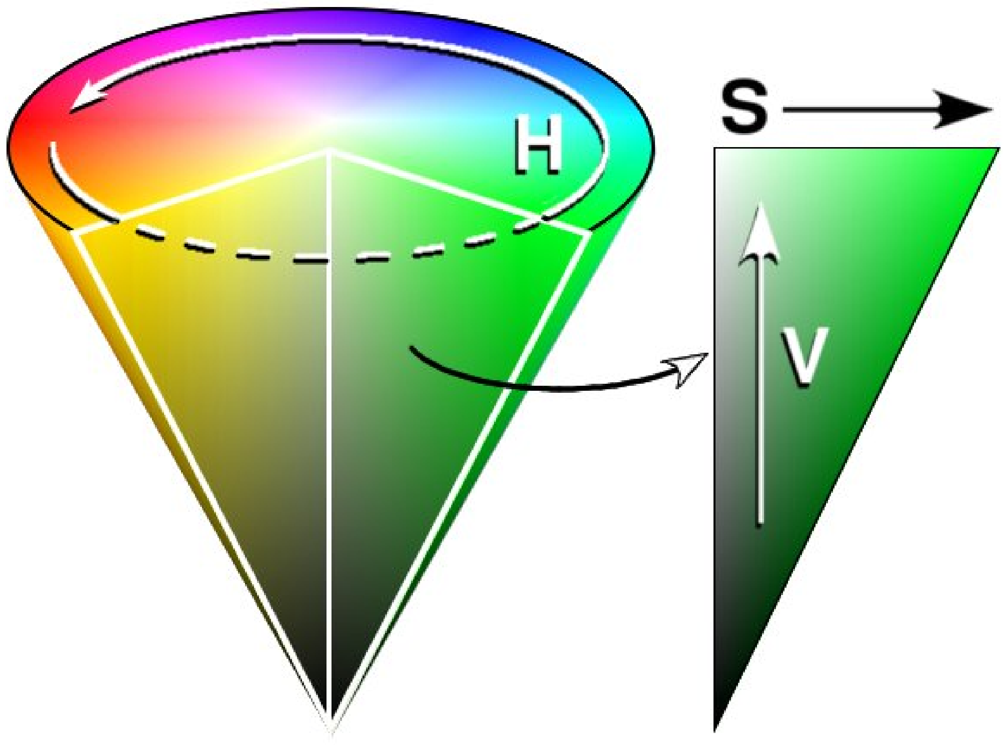

Figure 14. HSV color space cone

A difficulty has been encountered in the processing of image feature extraction when pixel colors are returned from the Image::Xpm module in different forms. Some are in six-digit hexadecimal format whereas the others are expressed by color names, e.g. grey100. This inconsistency is resolved by using a matching table that maps colors from one format to another.

4.3 Cross validation

After the feature vectors of all the 5000 images have been calculated, a decision tree is built by machine learning. As mentioned above, I chose to use the BoosTexter decision list algorithm. It uses strong hypothesis file (.shyp) to store the list of weak classifiers and their weights that it has generalized from the training data.

Cross validation is employed in the testing process. The basic idea of cross validation is to obtain more accurate estimations about how well the machine learner predicts unseen data. This is done by setting aside some fraction of the known data and using it to test the prediction performance of the hypothesis induced from the remaining data. K-fold cross-validation means that k-experiments are run, each time setting aside a different 1 / k of the data to test on. The final average results reflect more stable performance measures than any single split of the labeled data into training and testing data.. An algorithm is illustrated in Figure 15.

1. Given N examples, divide them into k equal portions, each portion containing roughly equal number of examples in each category.

2. Repeat for i = 1 to k
   a) Put i-th portion into the BoosTexter test file.
   b) Put the rest of the N examples into the training file.
   c) Run BoosTexter.
   d) Retrieve the test accuracy for each boosting round.

3. Find the average test accuracy of each boosting round across the k-runs.

4. Output the best average test accuracy across different boosting rounds.

Figure 15. Algorithm for cross validation

5. Evaluation and testing

In all experiments reported in this thesis, I have used 5-fold cross validation. In evaluating my NPIC system, the following three questions are of special interest:

1) How does the test accuracy change with increasing data size?
2) How does the test accuracy change with different set of text/image features?
3) What are the features that are most frequently used by BoosTexter in building the strong hypothesis? And what are words (for text fields) and threshold values (for continuous fields) that are selected by BoosTexter?

The above three questions will be addressed in Section 5.2, 5.3 and 5.4 respectively. A series of testing are carried out while the noisy corpus automatically retrieved from Google image search, *i.e.* the 5000 images, is used as the training/testing dataset. Therefore, Section 5.1 describes a test that is conducted to check how accurately image categories are constructed by Google.

5.1 Reliability of Google image retrieval

In Section 3.6, the image categories I constructed using Google image search were hypothesized to be accurate. I manually annotated a subset of these images and assess the reliability of using Google as a substitute. To do this, a small sample of 200 images, of which 50 are from each category (except "icon", because all icon images are acquired from a large reliable source found on the web[1], and therefore there is no need to test for its accuracy which is guaranteed to be 100%) is taken from the set of 5000 images. The number of correctly categorized images are recorded and shown in Table 3.

[1] See Section 3.6.

Category	No. of images	No. of correctly labeled images	Accuracy
Map	50	50	100%
Diagram	50	49	98%
Artwork	50	46	92%
Cartoon	50	44	88%
Total	200	189	94.5%

Table 3. Accuracy of image category labels

Hence, an agreement of 94.5% is achieved for this random sample of 200 images. It should be noted that the accuracy is relatively low for categories “artwork” and “cartoon” mainly because some photographic images have been returned by Google’s search. This significantly reduces the accuracy of correctly labeled images.

5.2 Results of the first round of cross validation

In the first round of testing, cross validation is run on different data sizes with all features (both text and image) included. Data size is incremented by 25 each time, and the average test accuracy is plotted against data size as shown in the following graph in Figure 16.

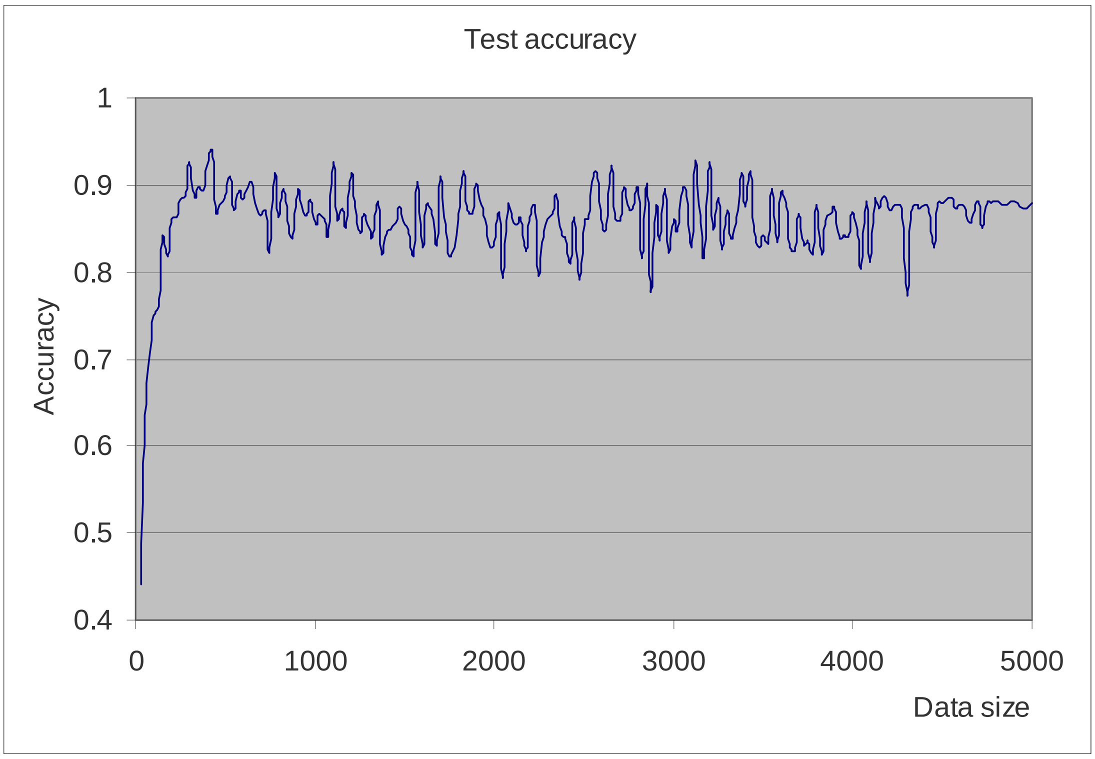

Figure 16. Test accuracy with different data sizes

The following observations can be drawn from the graph shown in the above figure.

1) Test accuracy increases significantly when data size increases from 25 to around 500.
2) After that, test accuracy seems to stabilize and fluctuate between 0.8 and 0.9.

These are good signs that there is a learnable pattern in the input data and that BoosTexter is capable of inducing it. But why does the accuracy fluctuate so much? I hypothesize that since the data sets were constructed by search engine image retrieval, the dataset may contain a high level of noise introduced. This noise can cause BoosTexter to find meaningless regularity in the data and cause overfitting. This may explain why maximum accuracy occurs for at the data size of 525 where an accuracy of 0.910 is observed.

5.3 Results of the second evaluation

I now answer how test performance is affected by using different sets of features to create the model. Four different tests have been conducted. We test performance on different data sizes by increasing the dataset size in logarithmic increments.

1) Test 1: All text/image features. These results are the same as in Section 5.2 and are restated here for convenience.
2) Test 2: All text/image features except filename, imageURL and pageURL. These three attributes are hypothesized to be used by Google to locate the image, and thus I hypothesize that they are likely to be the most frequently re-used by the BoosTexter learner. To have an unbiased test of performance, these features need to be removed in assessment.
3) Test 3: Only text features.
4) Test 4: Only image features. I expect this run to give the poorest performance among the tests.

Test results are presented in Table 4 and plotted in Figure 17.

Data size	Test with all features (Test 1)		Test with all features except filename and URLs (Test 2)		Test with only text features (Test 3)		Test with only image features (Test 4)	
	Test accuracy	No. of Rounds	Test accuracy	No. of Rounds	Test accuracy	No. of Rounds	Test accuracy	No. of Rounds
25	0.440	6	0.400	1	0.440	42	0.320	3
50	0.620	59	0.460	11	0.520	37	0.360	2
100	0.750	41	0.590	23	0.760	240	0.460	151
250	0.884	39	0.764	167	0.884	93	0.704	305
500	0.886	179	0.794	174	0.924	368	0.738	353

1000	0.854	117	0.668	257	0.894	175	0.596	488
2500	0.860	321	0.605	413	0.864	428	0.575	282
5000	0.889	500	0.586	120	0.863	380	0.542	207

Table 4. Results of tests with different sets of features

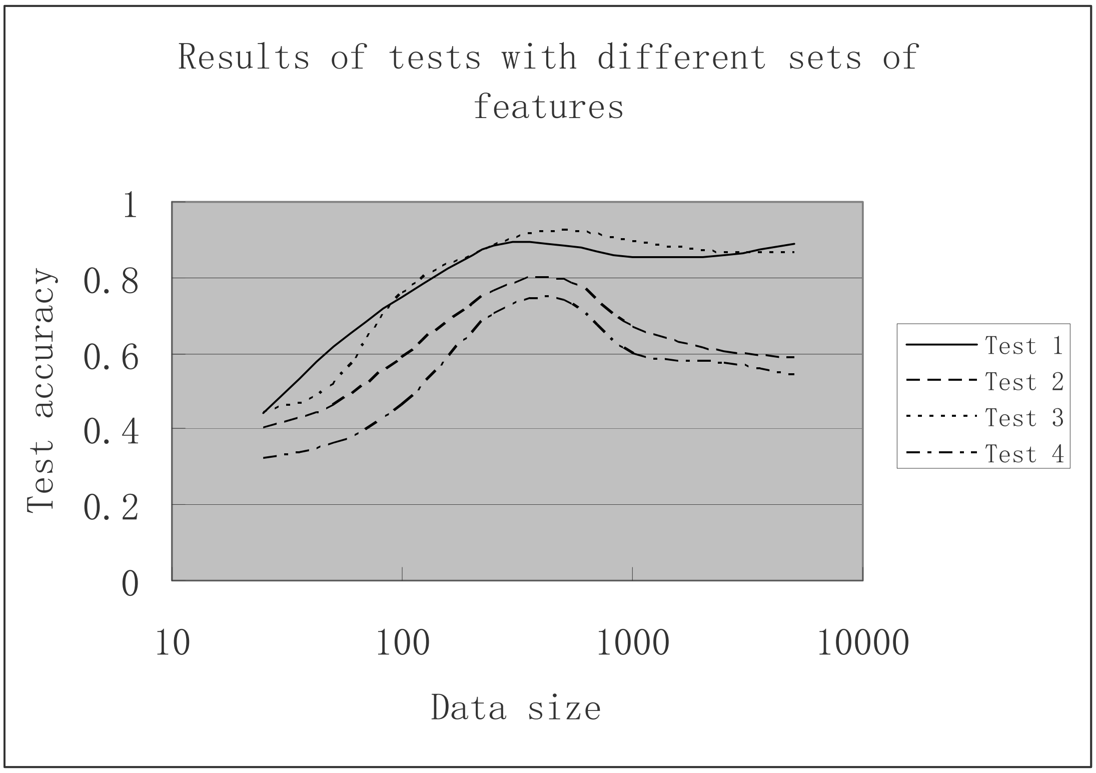

Figure 17. Four test results plotted in graph

It is evident from both the result table and graph that BoosTexter’s performance drops when fewer features are used as input. Test 1 (all features) and Test 3 (text features only) give satisfactory and almost similar results. However, when all text features are removed from the feature vectors, as in Test 4, or when the filename and URLs are removed, as in Test 2, performance drops significantly. This shows that BoosTexter depends more heavily on text features than image features, but running on only image features still gives a best accuracy of 0.738, which is still quite satisfactory.

5.4 Analysis of features selected by BoosTexter

In this section, I investigate the features selected by BoosTexter for classification. The test is done by running BoosTexter on the 5000 images for 100 boosting rounds using all the textual and image features. The table below shows the frequencies of each feature being used for a decision rule and their values or thresholds.

Feature		Frequency	Values (or thresholds)
Textual features	Filename	29	map, cartoon, painting, diagram, venn, graph, pie, disney, drawing, anime, artwork, world, street, directory, chinese, garfield, 1, 2520painting, chart, newyork
	Extension	3	Ico, jpeg
	Size	3	9.28e+04, 8.04e+03, 3.53e+05
	Height	2	180, 194
	Width	3	130, 125, 592
	X-resolution	0	
	Y-resolution	0	
	Comments	0	
	ImageURL	20	artwork, anime, maps, cartoon, directory, physics, painting, disney, cartoons, comics, stanford, nl, art, uk, zone, center, woodsigns, gifs, demon
	PageURL	25	artwork, www, map, art, city, au, maps, world, planning, venn, tutorial, garfield, chemistry, us, maths, chinese, children, cards, cn, tamu, hunt, design, drawing, asp, promotionlist

Image features	Colors	1	1.7e+0.5
	Depth	0	
	Interlace	0	
	Background_hex6	0	
	Background_hex3	0	
	Background_hue	2	337, 22.6
	Matte_hex6	0	
	Matte_hex3	0	
	Matte_hue	0	
	PrevalentColor_hex6	0	
	PrevalentColor_hex3	0	
	PrevalentColor_hue	0	
	Fraction_with_ prevalent_color	1	0.0634
	f1d1	1	0.0536
	f1d2	0	
	f1d3	2	0.75, 0.475
	f1d4	4	0.051, 0.966, 0.313, 0.942
	f2d1	3	0.028, 0.21, 0.0824
	f2d2	0	
	f2d3	0	
	f2d4	0	
	f1_to_f2_d1	1	0.0209
	f1_to_f2_d2	0	
	f1_to_f2_d3	0	
	f1_to_f2_d4	0	

Table 5. Frequencies of attributes and their values/thresholds

Observations are consistent with expectations that filename, imageURL and pageURL are chosen most frequently by BoosTexter. It can also be seen that many of their selected values are meaningful, *i.e.* they are indicative of certain image categories. For instance, the word “graph” in the filename probably infers that the image is a diagram. On the other hand, there are meaningless attribute values as well, like the “www” in the pageURL.

5.5 Test on 250 correctly labeled images

Due to the existence of some wrongly annotated images in the data set, one likes to see how well BoosTexter performs on a set of images whose categories are known (not noisily labeled by search engine retrieval). Therefore, 250 images known to have correct labels (50 from each category) are randomly selected from our corpus of 5000 images and the rest are used for training. Results by category are shown in Table 6.

Category	No. of images	No. of images correctly categorized by BoosTexter	Test accuracy
Map	50	47	94%
Diagram	50	48	96%
Artwork	50	50	100%
Cartoon	50	50	100%
Icon	50	50	100%
Total	250	244	97.6%

Table 6. Test results on manually annotated images

The results are very good. However, this sample of test data comes from the 5000 images, and probably a slightly lower test accuracy should be expected for a random sample of images from the web. The results indicate that the low performance reported in the earlier tests may actually be due to noise in the test data category labels. Although this noise level is reported to be low (from the first test), the compounded error suggests that the classification accuracy of the NPIC system is likely to be higher than what our automated tests have shown.

To illustrate the NPIC system, a demonstration program has also been written. It can be run using either a URL (if the test image is on the web) or a filename (if the test image is stored locally) as input. The predicted image category is displayed.

6. Conclusion

In this project, I have developed an image categorization scheme, although only its first level categories have the focus in the current NPIC system. Salient text and image features have been extracted from images and used as training data of machine learning. The resulting NPIC system predicts the category of a random image with fairly good performance of around 97.6%.

.

Further improvements can be made in the following areas:

- A better and larger source of images should be found, since the Google image search does not always give correct category labels. Future work can examine how to integrate both supervised, manually annotated data with the semi-supervised approach taken in this thesis.

- More discriminative features are to be introduced, especially in the image domain.

- The semantic relationship of words found in the text fields can be analyzed through natural language processing. Using general lexical resources, the system can generalize that "rabbit" is a type of "animal", which may help bring in more diverse training images. I believe that this improvement will have a significant impact on test accuracy.

7. References

1. Chapman, N. (1997). Perl: The Programmer's Companion. John Wiley & Sons Ltd, 1997.

2. CPAN, "Comprehensive Perl Archive Network", <http://www.cpan.org>

3. Egressy, G. "HTML COLORS and the X-Windows names", <http://www.vmunix.com/~gabor/html_colors.html>

4. Frankel, C., Swain, M., and Athitsos, V. Webseer: an image search engine for the world wide web. University of Chicago Technical Report TR96-14, 1996.

5. Gorkani, M. M. and Picard R. W. (1994). Texture orientation for sorting photos "at a glance". Proc. ICPR, Oct 1994 pp. 459 – 464.

6. Guelich, S., Gundavaram, S. and Birznieks, G. (2000). CGI Programming with Perl. Second Edition, O'Reilly, 2000.

7. Hu, J. and Bagga, A. (2003). Functionality-based web image categorization. WWW2003.

8. Huang, W. H. (2004). Document image analysis and recognition. PhD thesis, National University of Singapore, Singapore; 2004.

9. Li, J., Najmi, A., and Gray, R.M. (2000). Image classification by a two dimensional hidden Markov model. IEEE Transactions on Sig. Proc., vol. 48, no. 2, pp. 517-533, February 2000.

10. Miano, J. (1999). Compressed Image File Format: JPEG, PNG, GIF, XBM, BMP. Addison Wesley Longman, Inc., 2000.

11. Schapire, R.E and Singer, Y. (2000). BoosTexter: A boosting-based system for text categorization. Machine Learning (2000), 39(2/3), 135-168.

12. Shin, C. K. and Doermann, D. S. (2001). Classification of document page images based on visual similarity of layout structures. In Proceedings of the SPIE Document Recognition and Retrieval VII, 2001.

13. Soffer, A. Image Categorization Using Texture Features (1997). Proc. Int'l Conf. Document Analysis and Recognition, pp. 233-237 1997.

14. WordIQ, "Definition of HSV color space",

<www.wordiq.com/definition/HSV_color_space>

15. Yiu, E. (1996). Image classification using color cues and texture orientation. M.Sc Thesis, Department of Electrical Engineering and Computer Science, MIT, USA.; 1996.