Image Indexing

11

Order is our favourite, but the truth is beyond order.

11.1 Numerical Indexing

11.1.1 List Indexing

In numerical indexing, each image I in the database has been represented as a k-dimensional feature vector: $\mathbf{x} = (x_1, x_2, \ldots, x_k)$. The simplest way of a numerical indexing is to create a list of (*image_id*, $\mathbf{x}$) as shown in the following:

$$\begin{aligned}(I_1, \mathbf{x}_1) &= [I_1, (x_{11}, x_{12}, \ldots, x_{1k})] \\ (I_2, \mathbf{x}_2) &= [I_2, (x_{21}, x_{22}, \ldots, x_{2k})] \\ &\vdots \\ (I_N, \mathbf{x}_N) &= [I_N, (x_{N1}, x_{N2}, \ldots, x_{Nk})]\end{aligned}$$

where N is the total number of images in the database.

A list is simple and useful for a small image database, however, it's impossible to use it for a very large commercial image database, because it would take a long time to search the entire list of millions even billions of images. Therefore, a more efficient data structure is needed to index large image databases. One of the simplest yet efficient data structure is the *k-d tree* indexing.

11.1.2 Tree Indexing

The simplest *k-d* tree is a binary tree. A binary *k-d* tree is an algorithm of repeatedly splitting the database into two subsets by cyclically dividing the k dimensions of the data. Given N number of k-dimensional feature vectors: $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$, a *k-d* tree first divides the N data into two sets or branches of approximately equal size

D. Zhang, *Fundamentals of Image Data Mining*, Texts in Computer Science, https://doi.org/10.1007/978-3-030-17989-2_11

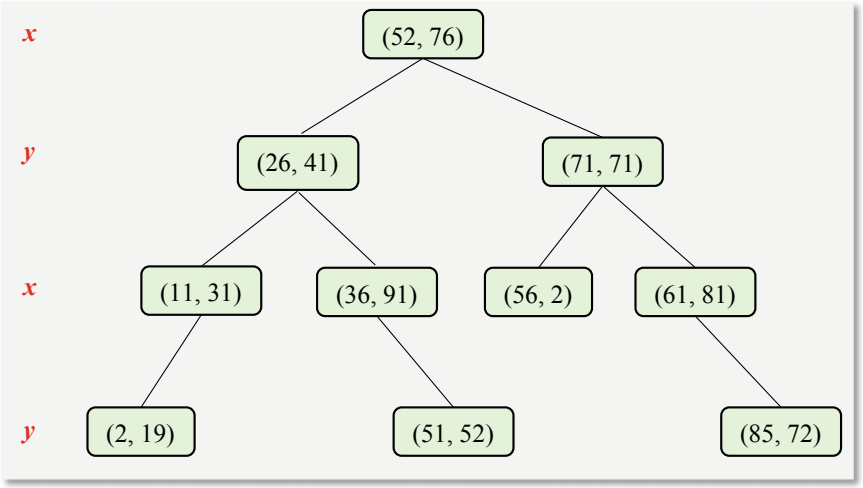

Fig. 11.1 A *k-d* tree for a 2D dataset of 10 data

according to the values of the first dimension of the *N* feature vectors. Next, each of the left and right branches are further divided into two subbranches of approximately equal size according to the values of the second dimension of the feature vectors. This division process continues until the *k*th dimension when the next division returns to the first dimension. The cycle goes on until no node is divisible any more.

Figure 11.1 shows an example of a *k-d* tree for a 2D dataset. Given a 2D dataset of 10 data: (56, 2), (36, 91), (52, 76), (2, 19), (11, 31), (61, 81), (85, 72), (71, 71), (51, 52), (26, 41), the binary *k-d* tree for this dataset is shown in Fig. 11.1. The labels on the left-hand side of the tree are the splitting criteria or the dimensions to be split.

A *k-d* tree reduces the search cost of a list of *N* data from an average O(*N*/2) to an average of O(ln *N*). For example, for a database of 10,000 images, the average search cost of a *k-d* tree is *integer* [ln (10,000)] ≈ 14, which is way smaller than 5,000, which would be needed for an exhaustive search of a data list. For very large image database, more efficient data structures can be used such as *n*-ary *k-d* tree, *quad-tree*, *octree*, *R-tree*, *cluster* tree, etc.

11.2 Inverted File Indexing

The data structures described above are for numerical data. If images in a very large database have been labeled with nominal or discrete values, they are equivalent to structured textual documents as shown in Fig. 11.2 [1]. Therefore, labeled images can be indexed using the same technique used for textual document indexing. In this section, we first review the inverted file for textual documents indexing and then introduce the inverted file for image indexing.

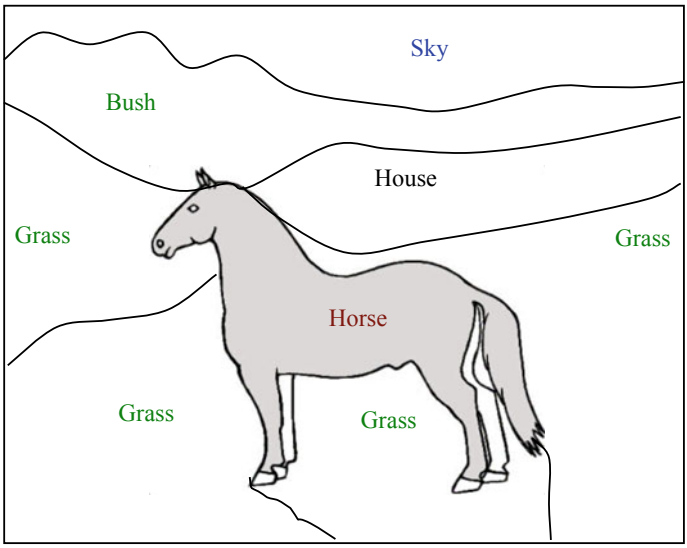

Fig. 11.2 An image with labeled regions

11.2.1 Inverted File for Textual Documents Indexing

In textual document indexing, a technique called *inverted file* is used. An inverted file is a (*terms*, *docs*) table instead of an ordinary (*docs*, *terms*) table. The reason of using an inverted file for document indexing is that a large amount of documents (millions/billions, e.g., web pages) can be indexed using a much shorter list of dictionary words (thousands). This makes the search for a large amount of documents very efficient.

The *i*th entry in an inverted file is a vector: ($term_i$, doc_1, doc_2, …, doc_n), where every document doc_j has $term_i$. Because a term carries a different amount of information in each of the documents, $term_i$ is given a different weight tw_j for each doc_j. Therefore, an inverted file can be shown in Table 11.1.

Since the documents are typically sorted in descending order of importance according to term weights (tw_j), the term weights can be omitted after sorting. Therefore, an actual inverted file is as simple as Table 11.2.

Table 11.1 A conceptual inverted file for textual document indexing

Term ID	Terms	Documents (weighted)
1	Apple	(doc_{11}, tw_{11}), (doc_{12}, tw_{12}), (doc_{13}, tw_{13}), …
2	Computer	(doc_{21}, tw_{21}), (doc_{22}, tw_{22}), (doc_{23}, tw_{23}), …
3	Tree	(doc_{31}, tw_{31}), (doc_{32}, tw_{32}), (doc_{33}, tw_{33}), …
…	…	…
n	Zebra	(doc_{n1}, tw_{n1}), (doc_{n2}, tw_{n2}), (doc_{n3}, tw_{n3}), …

Table 11.2 An inverted file after sorting

Term ID	Terms	Documents (ranked)
1	Apple	doc_{11}, doc_{12}, doc_{13}, …, doc_{1n}, …
2	Computer	doc_{21}, doc_{22}, doc_{23}, …, doc_{2n}, …
3	Tree	doc_{31}, doc_{32}, doc_{33}, …, doc_{3n}, …
…	…	…
n	Zebra	doc_{n1}, doc_{n2}, doc_{n3}, …, doc_{nn}, …

The *term weight* (*tw*) of a term t in a document d is determined by two factors: the *term frequency* of t in document d and the *inverse document frequency* of t in the entire database D. Specifically, *tw* is defined as follows:

$$tw(t,d) = tf(t,d) \times idf(t,D) \tag{11.1}$$

where *tf* stands for *term frequency* and *idf* stands for *inverse document frequency*. The *tf* and *idf* are given in (11.2) and (11.3), respectively.

$$tf(t,d) = \frac{f(t,d)}{\sum_{t_i \in d} f(t_i,d)} \tag{11.2}$$

$$idf(t,D) = \log\left(\frac{\|D\|}{df(t)}\right) \tag{11.3}$$

where

- $f(t, d)$ is the number of occurrence of term t in document d;
- $df(t)$ stands for the *document frequency* of t, it is the number of documents in D which have term t;
- $\|D\|$ is the total number of documents in D.

Recent research shows that the location where a term appears in a document is also a factor in determining the term weight, e.g., a term t in the *title* or *head* section of a web document is given a much higher score because the information of a term in a title or a head is much more important than that in the *body* text.

11.2.2 Inverted File for Image Indexing

The inverted file indexing method can also be applied to image indexing. Once the regions in an image database have been labeled with semantic concepts, images in the database are essentially translated into textual documents. Therefore, images can now be indexed and retrieved the same way as textual documents. Specifically, images in the database are indexed using an inverted file structure, where each index is a vector of the form: (*term*, *image*1, *image*2, …).

Since each term of an image is associated with a number of image regions, the term weight of an image term is determined by three factors: *area*, *position*, and spatial *relationship* [1, 2]. As a result, three corresponding weights have been defined respectively: *aw*, *pw*, and *rw*.

11.2.2.1 Determine the Area Weight *aw*

Let

- $aw(t)$ be the *area weight* of term t in image I,
- $R(t)$ be the area of a region which is labeled with term t in image I.

Then, $aw(t)$ is defined as the sum of all $R(t)$ in image I normalized by the area of the image. $aw(t)$ is equivalent to the term frequency in the textual document indexing. Mathematically,

$$aw(t) = \frac{\sum_{R \in I} R(t)}{\|I\|} \tag{11.4}$$

For example, in Fig. 11.3 [2], the 4 pink and reddish regions in the center of the image are all labeled as a term of "flower" by a classifier, therefore, the weight of the term "flower" is determined by the total area of the 4 regions, which is the area of the single flower in the image.

11.2.2.2 Determine the Position Weight *pw*

It is known that each type of objects usually has its natural position in an image, e.g., sky is naturally located at the top, grass is naturally located at the bottom, animals are naturally located at the center, etc. Based on this observation, a position weight can be defined for each term of an image document.

Fig. 11.3 A flower in the center with 4 segmented regions

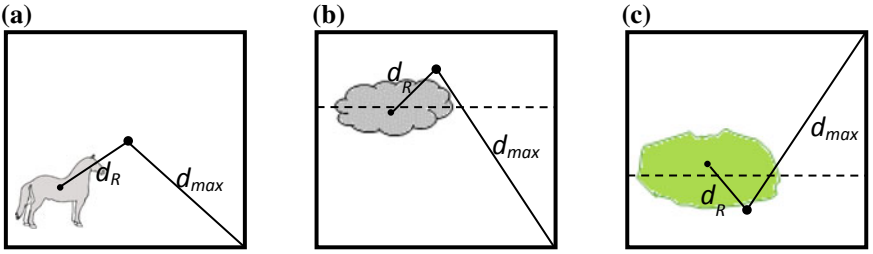

Fig. 11.4 Calculation of d and d_{max} for **a** animal, **b** sky and **c** grass regions

Let

- $R(t)$ be one of the regions in image I associated with term t;
- d_R be the distance between the centroid of the region $R(t)$ and the center of $R(t)$'s natural position in image I;
- d_{max} be the maximum distance between the center of $R(t)$'s natural position and the boundary of image I.

Then, the *position weight* of term t is defined as

$$pw(t) = \sum_{R(t) \in I} 2 \times \left(1 - \frac{d_R}{d_{max}}\right) \tag{11.5}$$

Figure 11.4 shows three examples of computing d_R and d_{max} [1, 2].

11.2.2.3 Determine the Relationship Weight *rw*

It is found that many types of objects usually go together in images such as bird and sky, computer and desk, beach and water, mammals and grass, flowers and tree, car and road, clock and wall, window and building, etc. The *co-occurrence* relationship can be used to determine the weight on a term in an image. For example, if both a "bird" term and a "sky" term are detected in an image, the weight of the "bird" term is doubled; if a "kangaroo" term appears together with a "tree" term and a "grass" term, the weight of the "kangaroo" term is tripled, so on so forth.

Let $r(t)$ be a term which co-occurs with term t in the image I, then the relationship weight rw is given as

$$rw(t) = \sum_{r(t) \in I} r(t) \tag{11.6}$$

Now that we have defined the three factor weights of a term t in image I, the final term weight is defined as follows (11.7):

$$tw(t) = aw(t) \times pw(t) \times rw(t) \tag{11.7}$$

Table 11.3 A conceptual inverted file for image document indexing

Term ID	Terms	Images (weighted)
1	Apple	(im_{11}, tw_{11}), (im_{12}, tw_{12}), (im_{13}, tw_{13}), …
2	Computer	(im_{21}, tw_{21}), (im_{22}, tw_{22}), (im_{23}, tw_{23}), …
3	Tree	(im_{31}, tw_{31}), (im_{32}, tw_{32}), (im_{33}, tw_{33}), …
…	…	…
n	Zebra	(im_{n1}, tw_{n1}), (im_{n2}, tw_{n2}), (im_{n3}, tw_{n3}), …

Table 11.4 The simplified inverted file from Table 11.3 after sorting

Term ID	Terms	Documents (ranked)
1	Apple	$im_{11}, im_{12}, im_{13}, \ldots, im_{1n}, \ldots$
2	Computer	$im_{21}, im_{22}, im_{23}, \ldots, im_{2n}, \ldots$
3	Tree	$im_{31}, im_{32}, im_{33}, \ldots, im_{3n}, \ldots$
…	…	…
n	Zebra	$im_{n1}, im_{n2}, im_{n3}, \ldots, im_{nn}, \ldots$

11.2.2.4 Inverted File for Image Indexing

Since each term in the dictionary has been given a weight in each of the images in the database, images in the database can be indexed using an inverted file the same way as in the textual document indexing. An example of an inverted file for image indexing is shown in Table 11.3.

After sorting the images at each row in descending order of importance according to the term weights, the above-inverted file is simplified as Table 11.4. The key difference between Tables 11.4 and 11.2 is that the terms in Table 11.4 are extracted from content features and by machine instead of interpretations by humans. Compared with textual documents, it's more difficult to determine the weight of a term in an image. We will show how the inverted file is used in image retrieval in Sect. 13.5.

11.3 Summary

Image indexing is about to put images in an image database into a data structure or order so that images in the database can be retrieved similar to retrieving alphabetic data from a Relational Database Management System (RDBMS). There are generally two types of approaches: numerical indexing and inverted file indexing.

If images are represented in numerical features, they can be indexed either using a list which is the simplest or using a tree structure. The list indexing is suitable for a small image database, while for a very large image database, the tree structure facilitates fast searching.