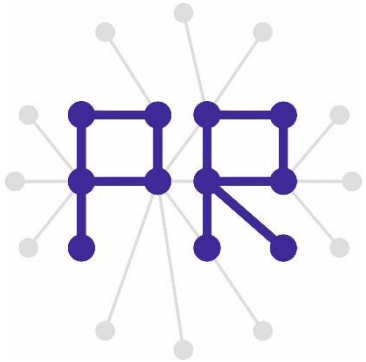
Multimedia Retrieval Exercise Course

8 Image Classification: Implementing SVM-based Classification

Kimiaki Shirahama, D.E.

Research Group for Pattern Recognition
Institute for Vision and Graphics
University of Siegen, Germany



Overview of Today's Lesson

- Implementing an image classification method using LIBSVM
  - Loading the image information
  - Preparing files for LIBSVM
  - Setting parameters of LIBSVM
  - Reading a LIBSVM result
- Suggestions to improve the classification performance

Overview of Image Classification Using LIBSVM

Purpose: Perform image classification by running LIBSVM in a C++ code

1. Load the information of all images together with their color histograms (The histogram file created for the query-by-example method is re-used)

for each object category

2. Output text files used for SVM training and test

3. Run LIBSVM commands to train and test an SVM *(Use "system" to run terminal commands in a C/C++ code)*

4. Read the result from the file output by LIBSVM

5. Sort images based on SVM outputs and make an HTML file of the result

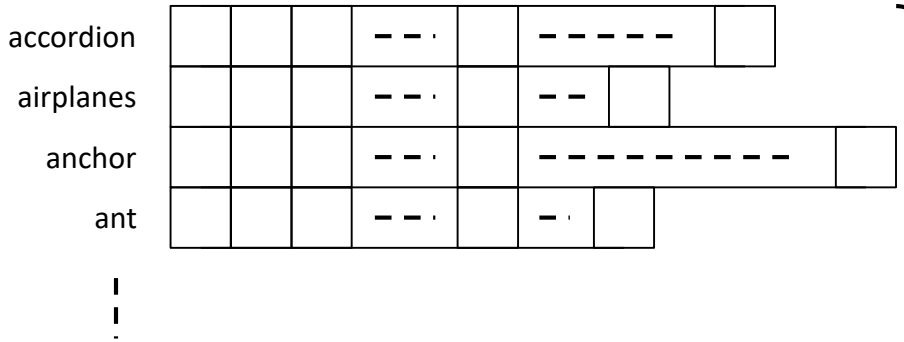
end of for

Loading the Image Information

My code stores the information of all images using a 2-dimensional vector, where each element is defined by the following structure:

```
struct ImageInfo{
    string label;       // Object class like accordion, airplanes etc.
    int id;             // XXXX in "image_XXXX.jpg"
    string filename; // Image filename
    vector<double> hist; // Color histogram
    double val; // Initialised as "-1"
};
```

vector< vector<ImageInfo> > imgs



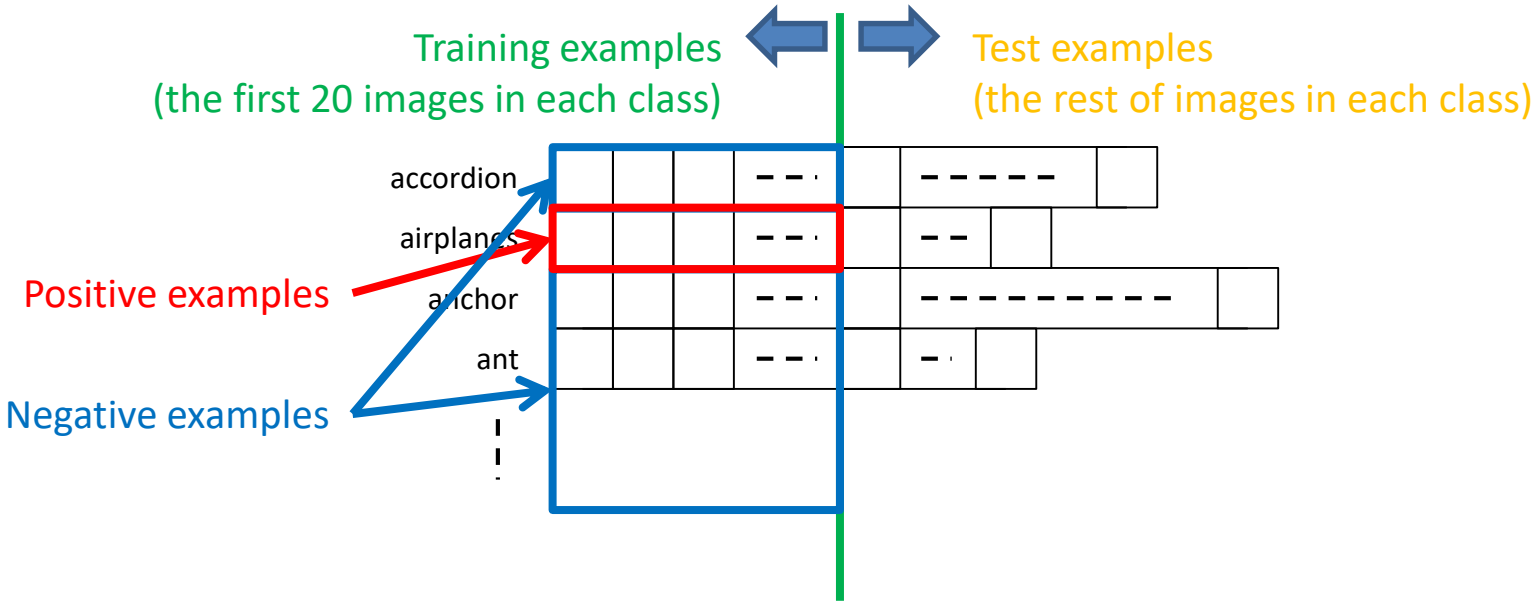
Each row is vector<ImageInfo>, and has a different number of "ImageInfo"s

Each element can be accessed by imgs[i][j]

Preparing Files for LIBSVM (1/2)

For each object class, I define training examples (positive and negative), and test examples in the following way:

Example case: Images showing airplanes are distinguished from the others



Preparing Files for LIBSVM (2/2)

One line of a LIBSVM file has the following format:

<label> 1:<1st bin value> 2:<2nd bin value> … 512:<512th bin value>

I set labels of positive, negative and test examples to +1, -1 and 0, respectively.
Any value can be used for labels of test examples, because we evaluate the performance on them using our C++ code.

(File of training examples)

Positive examples

+1 1:0.397322 2:0.0024 3:0.00174444 4:0.0015 5:0.00154444 6:0.00134444 7:0.00243333 8:0.223211 9:0 10:0 11:0 12:0 …
…

Negative examples

-1 1:0.276013 2:0.0035641 3:0.00148718 4:0.00133333 5:0.00174359 6:0.00362821 7:0.00732051 8:0.0133974 9:0.00391026 …
…

(File of test examples)

Test examples

0 1:0.261778 2:0.0669667 3:0.236444 4:0.0829667 5:0.123689 6:0.146411 7:0.0375778 8:0.0441667 9:0 10:0 11:0 12:0 …
0 1:0.2583 2:0.00811111 3:0.00641111 4:0.00527778 5:0.0055 6:0.00617778 7:0.0112889 8:0.698933 9:0 10:0 11:0 12:0 …
…

NOTE:

1. As like the query-by-example case, each histogram should be normalized so that the sum of bin values is equal to “1”
2. You don’t have to output data into a file if you use LIBSVM as a library in C++.
   (see README of LIBSVM for more details)

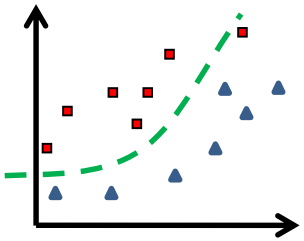
Setting Parameters of LIBSVM

I set three parameters of LIBSVM as follows:

1. γ: This determines the complexity of a classification boundary

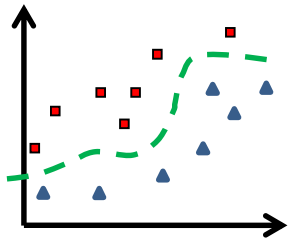
$$K(x, x_i) = \exp\left(-\frac{\| x - x_i \|^2}{\gamma}\right) = \exp(-\gamma \| x - x_i \|^2)$$

Usual formulation *LIBSVM formulation*



Small γ

Training examples which are difficult to classify, are ignored (The boundary may be too simple to accurately classify test examples)



Large γ

Training examples are forced to be correctly classified (The complex boundary does not necessarily work for test examples (*over-fitting*))

One reasonable choice is to set γ to the inverse of the average "squared" Euclidian distance among training examples

➡ For any $\| x - x_i \|^2$, its scaling by γ is probably reasonable.

2. C: This determines the tolerance of mis-classified training examples

Based on my experience, **C=2** is a reasonable choice.

3. -b 1: This is needed to let LIBSVM output continuous values for test examples

If this is not set, LIBSVM just output the predicted class label for each test example

If "-b 1" is used, LIBSVM outputs the probability that each test example belongs to a class.

This probability approximates the distance of the test example to the classification boundary.

Reading a Result by LIBSVM

(A result file output by LIBSVM)

```
labels 1 -1
-1 0.289552 0.710448
1 0.952875 0.0471254
-1 0.12747 0.87253
-1 0.0421646 0.957835
1 0.757121 0.242879
1 0.999989 1.08845e-005
...
```

- The first column represents the predicted class label for a test example:
  If the probability for the class “1” is more than 0.5, “1” is assigned, otherwise, “-1” is assigned.
- The second column represents the probability that a test example belongs to the class “1 (positive)”
- The third column represents the probability that a test example belongs to the class “-1 (negative)”

Probability that the target object is shown in a test example

Read values in the second column and set them to the “val” field of each test example.
The list where test examples are sorted in terms of “val” fields, is a recognition result.

- Please refer to the 10-th slide in the 7th lesson (p05_qbe2.pdf) for sorting a vector of ImageInfo.
- In addition, by referring to the slides in the 8th lesson (p06_qbe3.pdf), you can compute the average precision as the recognition performance.

To Improve the Classification Performance

1. Integrate a color histogram of an image with another type of feature like edge
   One approach is to extract HOG (Histogram of Oriented Gradients) feature (although I have not yet tested it)
   The following web page may be useful for understanding HOG features:
   http://www.juergenwiki.de/old_wiki/doku.php?id=public:hog_descriptor_computation_and_visualization

2. Use more training example. You can increase the number of training examples from 20 to, for example, 40. Or, you can use additional images that are collected on the Web