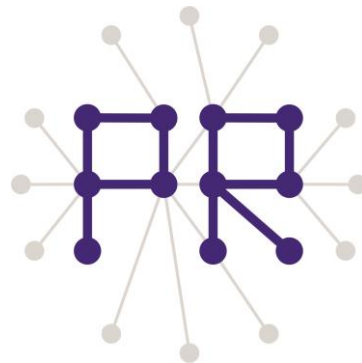


Multimedia Retrieval Exercise Course

12 Classification by Support Vector Machine

Kimiaki Shirahama, D.E.

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



Overview of Today's Lesson

- ❑ Support Vector Machine (SVM) on BoVWs
 - Characteristics of SVM
- ❑ LIBSVM (famous SVM software)
 - How to use LIBSVM

(Probably, some of the following issues will be done in the next week)

- ❑ Finishing a generic object recognition method by applying LIBSVM to BoVWs
 - Loading the information of images
 - Preparing files for LIBSVM
 - Setting parameters of LIBSVM
 - Reading a LIBSVM result
- ❑ Suggestions to improve the recognition performance

Overview of Generic Object Recognition Using Bag of Visual Words (BoVW)

1. Visual word extraction: Organise local features into groups of similar features

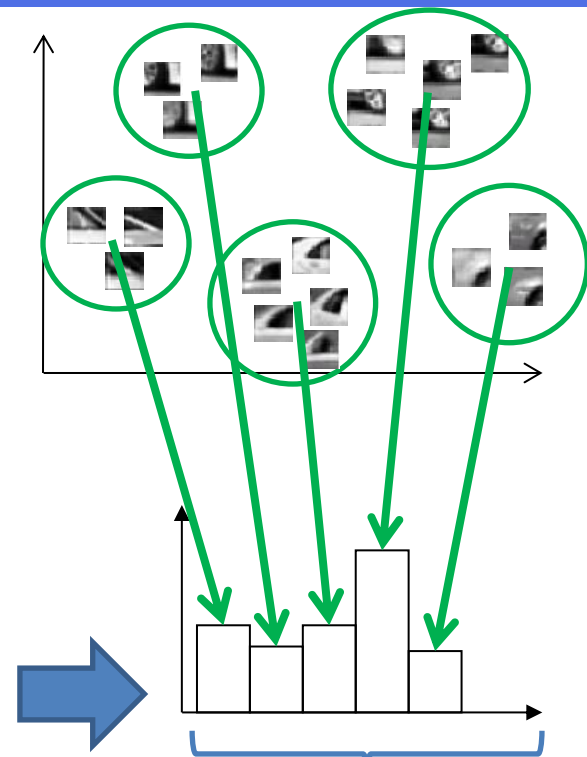
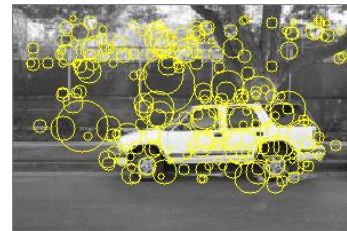
➡ *The center of each group becomes a visual word*

More than 100,000 local features are organised into more than 1,000 groups. In other words, more than 1,000 visual words are extracted

2. BoVW representation: Assign local features in each image to the most similar visual words

➡ For each visual words, the number of assigned local features becomes the value of a bin

The number of dimensions of a histogram (vector) is more than 1,000.

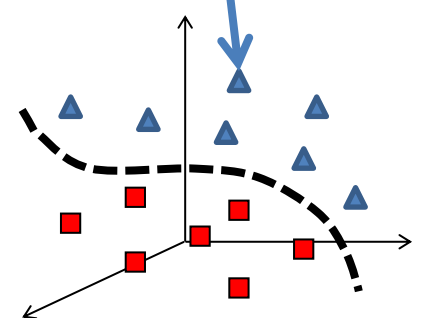


3. Classification: Extract the boundary between images where a certain object is shown and images where it is absent

➡ **One image represented by BoVW is represented as a point in the high-dimensional vector space**

Because of the high-dimensionality, simple similarity measures (e.g., Euclidian distance and cosine distance) do not work. Support Vector Machine (SVM) or other effective classifiers for high-dimensional data must be used.

I spent more than one year to find this point ☹



Today

Support Vector Machine on BoVWs

- My Personal Interpretation of SVM -

SVM extracts a useful similarity measure which combines weighted similarities of a test example to training examples

(Decision function of SVM)

$$h(\mathbf{x}) = \sum_{i=1}^N \alpha_i y_i K(\mathbf{x}, \mathbf{x}_i) - b$$

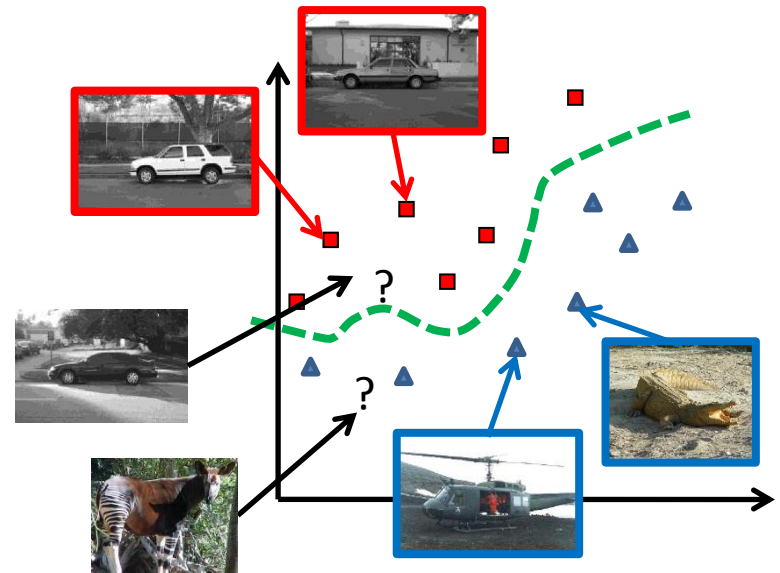
Weight

Kernel function (similarity between x_i and x)

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

Euclidian distance

The optimal set of weight is computed using training examples



Training examples

(having the information from humans: A machine learns from these examples)

- Positive examples: Images where a target object is shown
- Negative examples: Images where the target object is not shown
- Test examples: Images where displayed objects are unknown (not-having the information from humans)

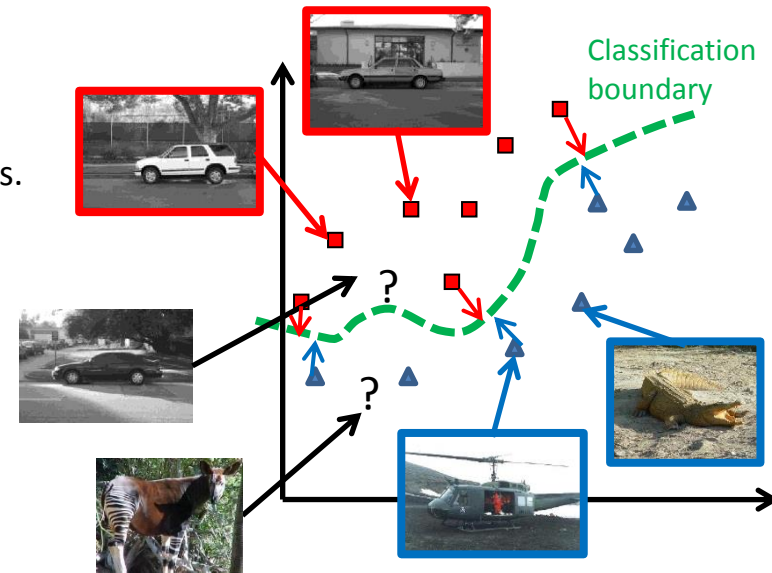
Characteristics of SVM

Advantage

- 1. Effectiveness for high-dimensional data:** Placing a classification boundary in the middle between positive and negative examples, is theoretically proven to be independent of the number of dimensions.
- 2. Effectiveness for a small number of training data:** A classification boundary is not probabilistically determined, but determined based on the geometry of positive and negative examples.
- 3. Convergence to the optimal solution:** The objective function for SVM training is convex.
- 4. Reduction of the computational cost for SVM test:** Only training examples selected as support vectors are needed to test the SVM.

Disadvantage

- 1. Quadratic increase of the computation time and the memory space depending on the number of training examples:** In the case of extracting a non-linear classification boundary (non-linear SVM), it is needed to compute the similarity for each pair of training examples.
- 2. Difficulty of understanding results:** Because of the high-dimensionality, it is difficult to know why good or bad classification is obtained. Especially, this is much difficult for non-linear SVMs.



LIBSVM

- ❑ One of the most famous SVM software (<http://www.csie.ntu.edu.tw/~cjlin/libsvm/>)
- ❑ A lot of SVM variations and options are supported
- ❑ Fast SVM training technique is implemented
- Another most famous software is SVM^{light}
- To my best knowledge, over-flow sometimes happens in SVM in OpenCV

How to Use LIBSVM

1. Download LIBSVM from <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
2. Run LIBSVM

Windows: Run executable files in the folder “windows”

Linux and Mac: Compile the source codes with “make” to create executable files

Open a command prompt or terminal

:SVM training svm-train (or ./svm-train) [options] training_set_file [model_file]

training_set_file: A file where training examples are written

model_file: A file where a trained SVM is output (if this is not specified, “training_set_file.model” is generated)

Example: svm-train ..\heart_scale

SVM test: svm-predict (or ./svm-predict) [options] test_file model_file output_file

test_file: A file where test examples are written

model_file: A file generated by SVM training

output_file: A file where classification results of test examples are written

Example: svm-predict ..\heart_scale.t ..\heart_scale.model ..\heart_scale.t.res.txt

NOTE: I created two files, the one named “heart_scale” contains the first 180 examples in the original “heart_scale”, the remaining 90 examples are contained in “heart_scale.t”.

Overview of a Code Using LIBSVM

Purpose: Perform classification (generic object recognition) by running LIBSVM in a C++ code

1. Load the information of all images together with their BoVW representations

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 or 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

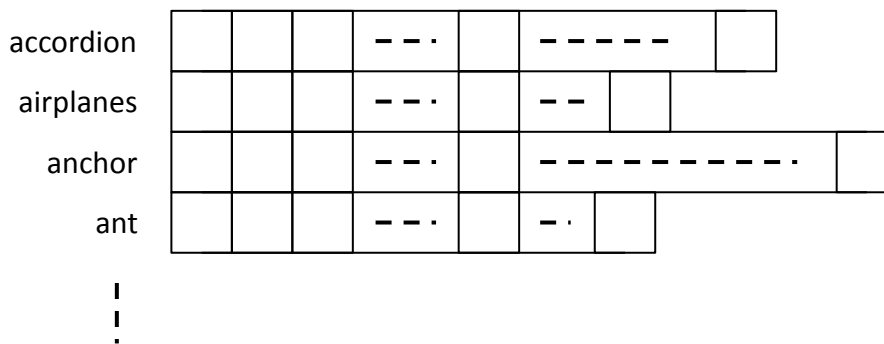
end of for

Loading the Information of All Images

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 such as accordion, airplanes etc.
    int id;         // XXXX in "image_XXXX.jpg"
    string filename; // Image filename
    vector<double> bovw; // BoVW representation
    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]

Setting Parameters of LIBSVM

I set three parameters of LIBSVM as follows:

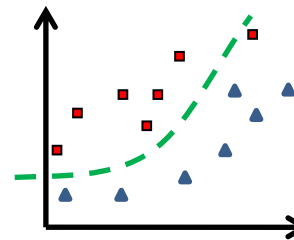
1. γ : This determines the complexity of a classification boundary

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

Usual formulation LIBSVM formulation

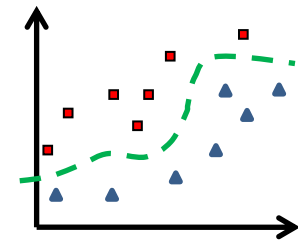
One reasonable choice is to set γ to **the inverse of the average (squared) Euclidian distance among training examples**

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



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*))

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

In 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”
- The third column represents the probability that a test example belongs to the class “-1”



Probability that a target object is shown in a test example

Read values in the second column and set them to the “val” fields for test examples. 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 4-th lesson for sorting a vector of ImageInfo. In addition, by referring to the slides in the 6-th lesson, you can compute the average precision as the recognition performance.

To Further Improve the Recognition Performance

1. Currently, we only use the first 10 images for visual word extraction.

➡ Extract visual words using more than 1,000,000 SURF features, which are randomly sampled from all of 9,144 images.

2. Change SURF features to Color SIFT features, which are more robust to illumination changes.

➡ The “colordescrptor” on the following URL is very useful for this purpose.

<http://koen.me/research/colordescriptors/>

In my experience, I recommend you to use Opponent SIFT or RGB SIFT features for densely sampled local regions

3. Use more training examples. You can use addition images that are collected on image sharing sites like Flickr.

➡ I recommend to use more than 10,000 training examples. To reduce the computational cost, it is necessary to use LIBSVM with the pre-computed kernel in the library mode

Thank you!

All contents in this course (Multimedia Retrieval Exercise) have been finished.