Specification of the AdaBoost IPM for use in SCENIC

Jan Šochman

jan.sochman@cmp.felk.cvut.cz
tel.: +420 2 2435 5731

Center for Machine Perception
Prague, Czech Republic

TN-eTRIMS-CMP-01-2006
version 2.1

January 11, 2007

This document describes AdaBoost image processing module (IPM) and its use for SCENIC system. Its purpose is to clarify the IPM's functionality and properties for easier incorporation into SCENIC.

1 Prague's vocabulary

In the following we use our own notions which can be slightly different from the one used in AI community. However, we observe the terminology of the initial document [1] and use the terms **entity** and **operator** accordingly.

Hence forward, the notion **measurement region** is used for the region in the image supporting the decision about the object type of the entity (e.g. area covered by the entity polygon plus small surrounding). We use the notion **classifier** whenever, given a measurement region in the image, the operator's task is to decide between two or more classes (i.e. object types plus one general class "not an object"). The notion **detector** is used in similar meaning but refers rather to repeated application of the classifier over a set of measurement regions (e.g. over different positions in the image). We call **training** the process of learning the classifier's parameters based on a set of examples. Trained classifier's parameters will be referred to as **classifier representation**. The examples used for training can be divided into positive and negative ones. **Positive examples** used in training is a set of measurement regions of the object type we are interested in (e.g. doors). **Negative examples** is a set of measurement regions which represent "everything else" except of the object type we are interested in (e.g. images of anything but doors). **Region of interest** is used when we are interested in a particular part of the image only, for instance because we want to examine certain region more closely. Since we are working with rectified images where the notion of scale is meaningful, we use **rectified image scale** for the number of pixels in the image per one metre in the scene.

2 Image description format

Recognised entities

The module is able to *learn* to recognise (classify) and detect (find in an image) any object either from eTRIMS ontology or independent one with rather fixed structure and similar (within the object type) width to height ratio. A counter example are for instance columns of different heights. Examples include, but are not limited to, particular window type, doors, cornices, chimneys, etc..

Description of entities

The IPM uses entities in two different ways. First, entities are used in the training phase as positive and negative examples given to the IPM. In this phase, a polygon representing the entity and a label (positive/negative, or ± 1) is required for each entity. The polygon has to represent a rectangular axis-parallel region in the image covering the whole entity.

Second, the output of already trained classifier (hypothesis of entities) is returned as a rectangular region and a real-valued confidence value. The region is again described by a polygon and the rectangle is axis-parallel. The confidence value is an approximation to the log likelihood ratio

$$R(x) = \log \frac{p(x|+1)}{p(x|-1)},$$

where x stands for an entity, and the $p(x|t)$ is a conditional probability that the entity x is of type t (+1 is used for the type of interest and -1 for background or "everything else"). It is clear that the confidence value can be either positive or negative and the higher the value the higher is the probability of the entity being of the object type.

Interaction with LabelMe toolbox

The AdaBoost IPM is able to return its outputs and accept the input in the LabelMe toolbox format.

Examples

See Figure 1 for an example of entities and their description used by AdaBoost IPM. Note that the regions displayed do represent the *measurement regions* and not the entities directly. The measurement regions cover also small surrounding of the entity so that the entity boundaries are included and can be used during training (they contain a lot of useful information). The enlargement of the entity polygon can be controlled by `options` attribute of the 'train' operator (see below) and differs in general for different object types. For training, each entity is assigned an object type (coded by colour in the figure). For detection/classification operators, each entity is assigned a real valued confidence value (not shown here).

3 Operators description

Prerequisites to run the IPM

The whole IPM currently works on rectified images only. It is not advised to use the IPM on non-rectified images. We will provide later an automatic rectification operator that rectifies the image.

Provided interface operators

init Creates `options` structure filled with default values. This structure can be modified and is used as one of the attributes to the operators. It controls internal parameters of the operators. Moreover, the IPM Matlab paths are set by this operator. This operator is called at system startup or during re-initialisation.

find object Given an image, a classifier representation for some object type, and a region of interest, it finds all objects of the type in the region of interest. When the region of interest is left empty, the whole image is searched.

get entity confidence Given an image, a classifier representation for some object type, and a rectangle specifying the entity, it returns a confidence that the entity is of the type.

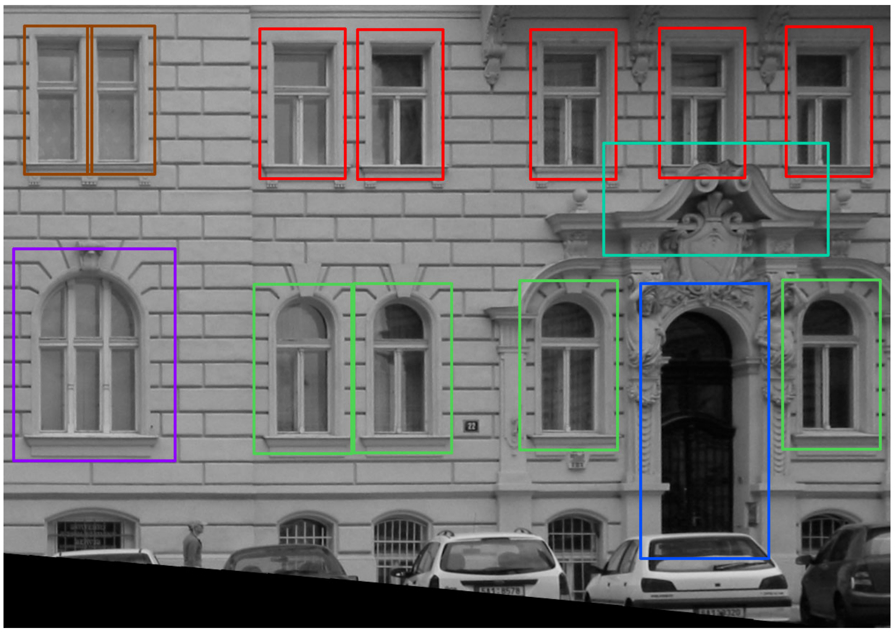

Figure 1: Examples of entities. Different colours are used for different object types.

train a classifier Given a set of positive examples representing one object type and a set of negative examples (representing “everything else”) a classifier representation for the positive examples type is trained.

update a classifier Given a classifier representation, an example and its label ('positive' or 'negative'), the classifier representation is updated. This operator has currently null functionality and is commented below.

detections filtering Given a set of hypothesised entities (outputs of previous operators) of *different* object types this operator filters out obvious mistakes. Very simple configurable rules are used (see parameters description).

draw entities Given an image and a set of hypothesised entities, the entities are drawn into the image.

shutdown This operator is called at system shutdown. It has currently null functionality and is included mainly for future enhancements and the AdaBoost IPM compatibility with other IPMs.

Important property of the AdaBoost IPM is that the same operators can be used for different object types (e.g. doors, windows, cornices, ...) by simple change of the classifier representation given to the operator.

We are also developing the `update a classifier` operator for incremental (online) training of the classifier. This means that only one example (positive or negative) is given to this operator at a time together with a classifier representation and the classifier representation is updated (instead of training it from scratch).

Besides these operators, a demonstration GUI application can be provided.

What parameters can be given to operators

Parameters given to the operators can be of several types listed in Table 1. In the following operator list, input and output parameters are specified and the parameter types are indicated in the parenthesis behind the parameters.

Type	**Description**
filename	Name of (usually) binary file (used for images, data files, etc.)
XMLfilename	Name of a text file in XML format
structure	Structure containing several fields
number	Numerical value
string	Text string
rectangle	Description of a rectangle

Table 1: Operator parameter types

The entity description contained in the XML files have to contain 1. entity rectangular polygon, 2. object type, 3. image filename to which the entity is related to (each entity can be from a different image!), and 4. rectified image scale for each image. A star (*) next to the XMLfilename type indicates that the entities in the file have filled also the confidence value tag. See Appendix A and B for XML file DTDs.

init

Input:	configuration file name (filename)
Output:	default options (structure), error message (string)

find object

Input:	image (filename), rectified image scale (number), classifier representation (filename), region of interest (rectangle), options (structure)
Output:	hypothesised entities (XMLfilename*), error message (string)

get entity confidence

Input:	image (filename), classifier representation (filename), entity (rectangle), options (structure)
Output:	confidence value (number)

train a classifier

Input:	positive examples (XMLfilename), negative examples (XMLfilename), options (structure)
Output:	classifier representation (filename), error message (string)

update a classifier

Input:	classifier representation (filename), image (filename), positive or negative example (rectangle), label (string, 'positive' or 'negative'), options (structure)
Output:	classifier representation (filename), error message (string)

detections filtering

Input:	hypothesised entities (XMLfilename*), filtering rules (array of strings), options (structure)
Output:	filtered hypothesised entities (XMLfilename*), error message (string)

draw entities

Input:	image (filename), hypothesised entities (XMLfilename*), options (structure)
Output:	image (filename), error message (string)

shutdown

Input:	options (structure)
Output:	error message (string)

The `options` parameter influences internal parameters of the operator. The `filtering rules` parameter specifies the rules which will be applied to the hypothesised entities and contain an *ordered* list of strings from pre-specified set of rule names. The available filtering rules are currently based on mutual overlap of detected entities (of a mixture of types). Each operator has the `error message` output parameter indicating success of the operator. If the operator succeeds, the `error message` ouput parameter is an empty string, otherwise it contains description of the error.

For all operators, Matlab help will be provided (use `help function_name` command in Matlab to find out the operator syntax).

Example of usage

Here, an outline of the feedback loop is sketched. Many other variations are possible. The main stress is not put on completeness but rather on the AdaBoost IPM usage understanding.

```
% initialise IPM
options = init('cfg_filename')

% initial IMP classification – collect entity hypotheses
all_hyp_entities = []
roi = []                                        % empty region of interest ⇒ whole image
for all classifiers do
   hyp_entities ⇐ find_object(img, r_img_scale, classifier, roi, options)
   all_hyp_entities = all_hyp_entities ∪ hyp_entities
end for
all_hyp_entities ⇐ detections_filtering(all_hyp_entities, rules, options)

% main feedback loop
while interpretation not good enough do
   case SCENIC_REASONING(hyp_entities)
      'I am sure that "there" must be a window':
         roi = [row, col, width, height]          % a rectangular region
         options.winclas_threshold −= 10          % higher sensitivity
         new_entities ⇐ find_object(img, r_img_scale, winclas, roi, options)
         % add new results to the hypothesis list
         hyp_entities = hyp_entities ∪ new_entities

      'Could this entity be a cornice?':
         conf_val ⇐ get_entity_confidence(img, corniceclas, entity_rect, options)

      'I have enough entities of the same type...':
         new_entity_clas ⇐ train_classifier(pos_examples, neg_examples, options)

      'I have found something, remember it':
         winclas ⇐ update_classifier(winclas, img, interesting_example, label, options)

      'I need a debugging output':
         out_img ⇐ draw_entities(img, hyp_entities, options)
   end case

   do some processing here...
end while

shutdown(options)
```

4 System requirements

- The IPM requires the Matlab environment with the Image Toolbox installed. It has been developed on Matlab 7.2. but is easy to upgrade for newer version or downgrade for an older version (not below 5.0).
- Parts of the module are written in C++, so also the compiler – gcc for Linux or Visual .NET for Windows operating system – is needed. Each Matlab version requires different gcc version so the appropriate one has to be used. The Visual .NET version 7.1 has been tested.
- Both Windows XP/2000/NT and Linux operating systems are supported. Windows XP and Gentoo Linux have been tested but the IPM should be easily transfered to similar operating systems.
- The IPM does not require any additional libraries.
- Expected computation time for training is in order of minutes for larger tasks (hundreds of positive and negative examples) and in order of seconds for detection/classification when working with images larger then 1000x1000 pixels. Usually, smaller images are sufficient for good performance.

References

[1] Lothar Hotz, *Template for describing image processing modules.* Hamburg Informatics Technology Centre, Hamburg, November 2006.

Appendix A. Found entities DTD

Following document type definition (DTD) describes XML files used by `find object`, `detection filtering` and `draw entities` operators.

```
<!ELEMENT annotation (image+)>

<!ELEMENT image (filename, rect_img_scale, entity*)>

<!ELEMENT filename (#PCDATA)>
<!ELEMENT rect_img_scale (#PCDATA)>

<!ELEMENT entity (type, confidence, polygon)>

<!ELEMENT type (#PCDATA)>
<!ELEMENT confidence (#PCDATA)>

<!ELEMENT polygon (pt, pt, pt, pt)>

<!ELEMENT pt (row, col)>

<!ELEMENT row (#PCDATA)>
<!ELEMENT col (#PCDATA)>
```

Appendix B. Training examples DTD

Following document type definition (DTD) describes XML files used by `train a classifier` operator.

```
<!ELEMENT annotation (image+)>

<!ELEMENT image (filename, entity*)>

<!ELEMENT filename (#PCDATA)>

<!ELEMENT entity (type, polygon)>

<!ELEMENT type (#PCDATA)>

<!ELEMENT polygon (pt, pt, pt, pt)>

<!ELEMENT pt (row, col)>

<!ELEMENT row (#PCDATA)>
<!ELEMENT col (#PCDATA)>
```