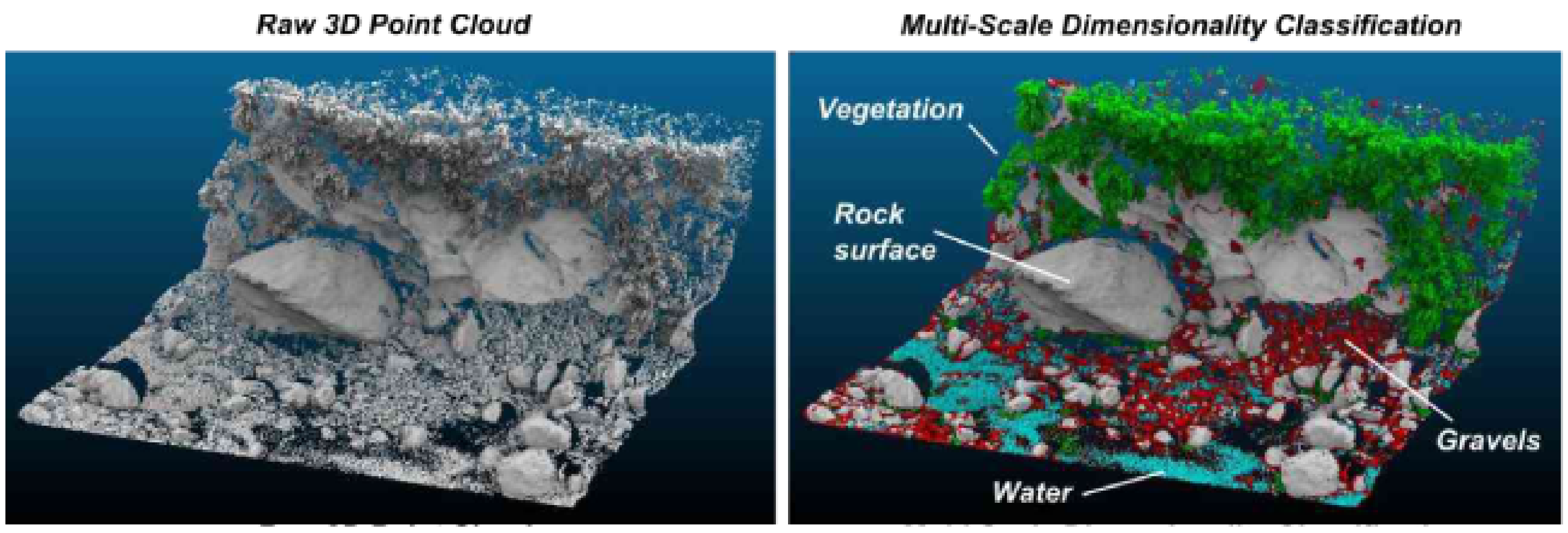


Classification of point clouds using the CANUPO software suite v1.2 (april 2013).

Dimitri Lague, Nicolas Brodu

This is the user guide for the CANUPO software suite with which you can classify 3D point cloud data. Examples are oriented towards vegetation classification in scenes measured by terrestrial LiDAR, but the technique is generic and applicable to many other contexts:

Brodu, N. and Lague, D., 3D Terrestrial LiDAR data classification of complex natural scenes using a multi-scale dimensionality criterion : applications in geomorphology, ISPRS journal of Photogrammmetry and Remote Sensing, 68, p. 121-134, 2012. doi: 10.1016/j.isprsjprs.2012.01.006.

While the software can run theoretically on any computer, we assume that you have either a workstation or a very good laptop (lot of hard drive space, a minimum of 4go of RAM and a good CPU). Note that numerous temporary files are created (and can be deleted afterwards) and the output files of the classification results are at least as large as the original file. This means that you need to be sure to have enough space available on your hard drive (but if you're working on point clouds, you know that you need a big (and fast) hard drive !).

If you have any further questions you can contact :

- regarding the step by step classification only on windows, or the application of the the method in various contexts : Dimitri Lague (dimitri.lague@univ-rennes1.fr)
- regarding the quick tutorial for classifier construction, the algorithm and the code : Nicolas Brodu (nicolas.brodu@univ-rennes1.fr)

If you find that something is unclear in this guide please let us know so that we can improve it. And don't forget to cite Brodu and Lague, 2012 if you use our software in your research.

v1.2: small update for the classification as you don't have to input the scales manually during the multiscale dimensionality computation (canupo.exe). Just input the classifier name (file.prm) and it'll work.

1. Step by step guide for classification only, on WINDOWS

After you've read this part, you can check the Example 1 : vegetation classification in the Otira gorge. It uses the batch Classification_only_Otira.bat that is ready to operate. The following text is more generic.

We assume that you have a point cloud in a format that is either an ascii file (x,y,z and eventually R,G,B,I) or in a format that can be loaded in Cloudcompare so that you can export an ascii file which is what CANUPO is using (we're discussing the possibility to switch to binary format). NOTE THAT THE BATCHES ASSUME THAT THE EXTENSION IS .txt

Optional : Cleaning the cloud

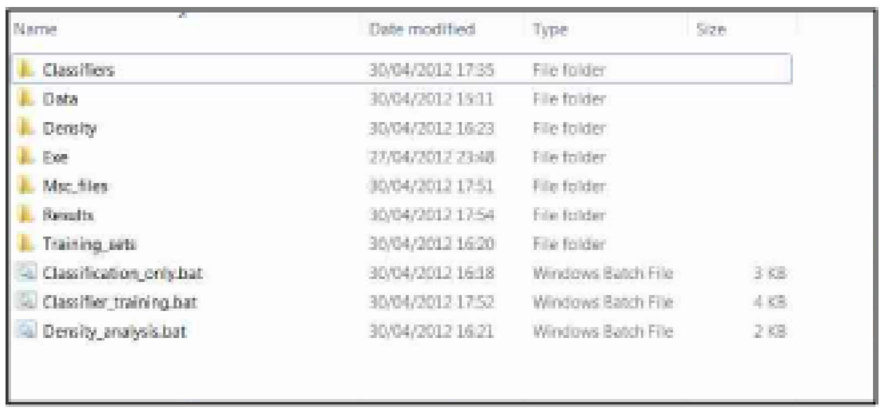
To clean the small bits of clouds which are just a few points or elements disconnected from the main scene (buildings in the back etc...), use a connected components filter in CC (I typically use octree 9 and a threshold of 20-30 points). You'll have to play with the parameters depending on the size of your cloud and your data density. Then you select all resulting clouds (or just the main one or two) (in the DB tree window of CC and using the same controls that in windows for group selection), recombine them (fuse icon in CC), clear the color field (edit->color->clear), and save this cleaned cloud. This cloud is now the raw DATA file. Having cleaned unused stuff allows to only generate core points (see next step) on data that actually matters for your application.

Optional, but quite often needed : generating a Core Points file

See Brodu and Lague, 2012 for more detail on the reason to use a CP file. Typically you subsample the DATA File down to a minimum point spacing of 5-10 cm on which the classification will be done (in CC : edit->subsample : space mode, or use subsample.exe (see tutorial in section 2)). This new dataset is the CORE file. You may also use directly the DATA file instead, but for very large scenes the classification can become quite long. The CORE file is also generally the one I'm using for displays and figures as you rarely need the full raw data file in that case.

Performing the classification

In the **PointCloud_Class** directory you should have something like that :



Name	Date modified	Type	Size
Classifiers	30/04/2012 17:35	File folder	
Data	30/04/2012 15:11	File folder	
Density	30/04/2012 16:23	File folder	
Exe	27/04/2012 23:48	File folder	
Msc_files	30/04/2012 17:51	File folder	
Results	30/04/2012 17:54	File folder	
Training_sets	30/04/2012 16:20	File folder	
Classification_only.bat	30/04/2012 16:18	Windows Batch File	3 KB
Classifier_training.bat	30/04/2012 17:52	Windows Batch File	4 KB
Density_analysis.bat	30/04/2012 16:21	Windows Batch File	2 KB

The Classifiers directory contains the classifiers .prm files. The .svg and .ppm are just used to visualize the classifier and eventually change it with a vector drawing software such as Inkscape (see the section "create your own classifier"). With each classifier should be a copy of the batch that was used to create it or a readme with a list of the corresponding scales to be used. Hence, **for a classification you need : the name of the .prm file and the corresponding set of scales (this will likely change soon)**

V1.2 : now you just have to give the name of the classifier .prm in canupo. It will automatically seek the corresponding scales. The old way of giving the scales in the command line still works. You can use prm_info.exe to get informations on the classifier scales.

The **Data directory** should contain your DATA files and CORE files.

The **Exe directory** contains the various executables

The **Msc directory** contains the multiscale characteritiscs of a point cloud. Given that in the classification script building the .msc is by far the longest part of the process, delete it only when you're happy with the classification. Indeed, if the classification is not correct, you can change the classifier (see *Building your own classifier*) and simply reapply the classifier to the .msc file by doing only the classify.exe part of the script. This is extremely fast. The only limitation is that your new classifier must use the exact same scales than the one in the .msc file.

The **Training_sets** directory is used for training the classifier (see *Building your own classifier*).

It also contains batch files (.bat) that you can modify by right clicking on it and choose edit. If you double click on it, it will actually run the executables.

The Classification_only.bat is ready to operate on the Otira dataset provided

You now need to modify the Classification_only.bat file to change the names of the DATA and CORE files, as well as the Classifier name and associated scales if needed. If this is your first time, try on a small dataset to have rapid results rather than throwing in a full scene that would require some time (up to 1 - 2 hours for very large scenes at full resolution, and depending on your computer).

Once you've done these changes, you double click on the batch and you wait...

Resulting file

In the results directory you should now have a .txt file (the name is automatically filled as a combination of the DATA and CORE names) with 8 fields that you can open in CC :

X Y Z CLASS CONFIDENCE N1 N2 ANGLE

CLASS : contains the attribute number of the class (something defined during the classifier construction). Typically this will be 1 and 2 *(it can be 0 if you have changed the arguments of classify and have introduced directly a separation of the classes with a confidence level (see description of the soft))*

CONFIDENCE : a measure of the confidence of the classification process (see Brodu and Lague, 2012 for more details). It varies between 0.5 (poor classification, i.e. random) to 1 (excellent confidence).

N1 : number of neighbouring points around the point at the minimum scale used by the classifier

N2 : number of neighbouring points around the point at the maximum scale used by the classifier

ANGLE : angle with respect to the horizontal defined at the largest scale used by the classifier

Separating the classes

Load your result file in CC. Your first scalar field will contain the attribute. Use *edit->scalar field->filter* (or directly the icon) to create subset cloud of your results. For instance vegetation layer (min:1-max :1) and no_veget layer (min:2-max:2).

Cleaning the classes

Confidence level

You can display the 2nd scalar and visually check where the classification was efficient or not. Play with the display slider to find what is the confidence level that you want to set to remove badly classified data. When you change this slider, points will turn grey when they are not in the range of the slider. Then you can directly use the scalar field filter icon to keep only the well classified points.

Density threshold

Sometimes the classification have a high level of confidence, yet it is visually wrong. This can arise in regions of the cloud where the point density was not enough to actually compute completely the multiscale parameters. In that case filtering the data as a function of N2 can be useful (say that it needs a minimum of 20 to 50 points to get a good results...it's something to try as it varies depending on the scene).

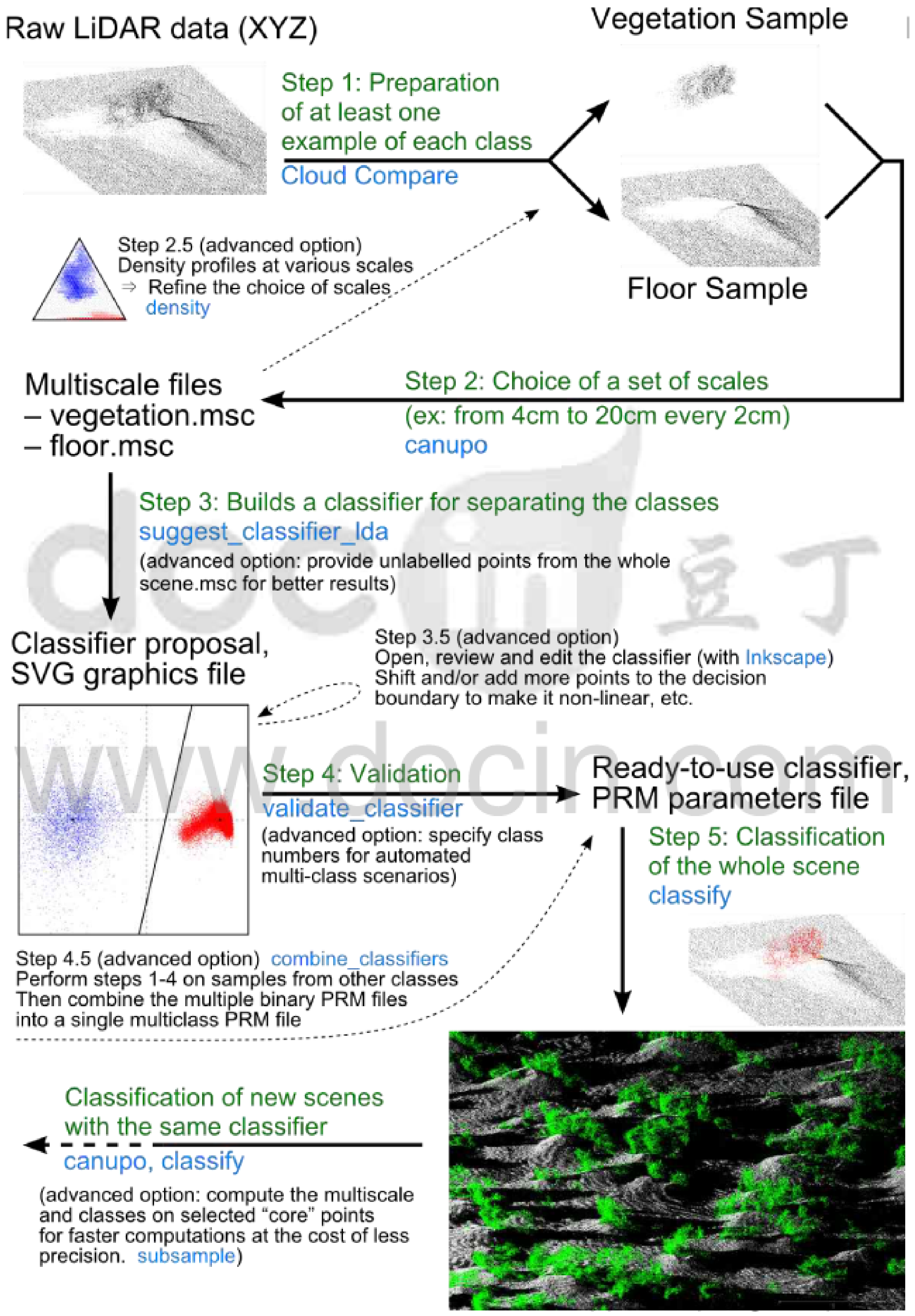
Batch processing of the result file

By using the filter.exe program you can easily filter your result file using different conditions, rather than manually going through the filters in CC. For instance, if you want to extract the class attribute 2 (rock surface for instance), with a level of confidence of 0.95, a minimum number of points of 50 at the largest scale then you can add the following command-line at the end of your batch :

```
filter.exe name_of_your_resultfile.txt filtered_file.txt 4:1.9:2.1 5:0.9:1 7:50:100000
```

Hence, assuming that you don't use a core point file, and you have tested all your parameters for filtering, the only thing you have to change is the name of the data file in your batch ! If you've learned a bit of batching, you can then launch the processing of several files automatically with exactly the same set of parameters.

Workflow for classifier construction



Raw LiDAR data (XYZ)

Step 1: Preparation of at least one example of each class

Cloud Compare

Vegetation Sample

Floor Sample

Step 2.5 (advanced option)
Density profiles at various scales
⇒ Refine the choice of scales
density

Step 2: Choice of a set of scales
(ex: from 4cm to 20cm every 2cm)
canupo

Multiscale files
– vegetation.msc
– floor.msc

Step 3: Builds a classifier for separating the classes
suggest_classifier_lda
(advanced option: provide unlabelled points from the whole scene.msc for better results)

Classifier proposal,
SVG graphics file

Step 3.5 (advanced option)
Open, review and edit the classifier (with Inkscape)
Shift and/or add more points to the decision boundary to make it non-linear, etc.

Step 4: Validation
validate_classifier
(advanced option: specify class numbers for automated multi-class scenarios)

Ready-to-use classifier,
PRM parameters file

Step 5: Classification of the whole scene
classify

Step 4.5 (advanced option) combine_classifiers
Perform steps 1-4 on samples from other classes
Then combine the multiple binary PRM files into a single multiclass PRM file

Classification of new scenes with the same classifier
canupo, classify

(advanced option: compute the multiscale and classes on selected “core” points for faster computations at the cost of less precision. subsample)

2. Tutorial for classifier construction with command line (Linux and Windows) and optional exercises

This tutorial is designed to get you quickly running with Canupo directly from the command line, and to teach you how to construct your own classifier following the workflow described in Fig. 2. The exercises are all optional but will teach you useful tricks. In section 3, you can also find a complementary user guide based on batch files rather than command line if your prefer.

Step 1: Download and unpack CANUPO, then enter the directory “tutorial”. You shall find the following files:

`floor.xyz`	This is the floor sample shown in the overview.
`vegetation.xyz`	This is the vegetation sample shown in the overview.
`scene.xyz`	A scene extract to test the classification.

They are simple text files containing x,y,z coordinates on each line, easy to work with. We recommend you to open them with the very useful CloudCompare free software that can be downloaded at http://www.danielgm.net/cc/.

Exercise: Extract another floor and vegetation sample from the scene using the scissors tool from CloudCompare.

Step 2: Choose a relevant set of scales. Think at how 1D, 2D or 3D the object looks like at various scales. In this case, above 20cm, the vegetation already looks like a big 3D ball, while the floor looks flat. We ideally want to use low scales for refining the classification (ex: detecting a patch of floor below a vegetation branch), but the floor is rippled so getting too low (<5cm) may introduce spurious 3D components. All this is approximate as it should be, rely on your intuition and do not worry, the classifiers are robust enough to small variations.

Now, extract the floor and vegetation multiscale features using canupo. Open a terminal (Unix) or dos command line (Windows) in the tutorial directory, and type the following two lines (shall work on both systems):

```
../canupo 0.04:0.02:0.2 : floor.xyz floor.xyz floor.msc
../canupo 0.04:0.02:0.2 : vegetation.xyz vegetation.xyz vegetation.msc
```

This computes the multiscale features from 4cm to 20cm every 2cm on both samples. To understand why names are duplicated type `../canupo` without arguments to see help, and read the article. This is related to core points, a reduced set of reference points at which to perform the computations when the data is too large. Here we compute the features on each point in the sample.

Exercise 1: Run the `../resample` tool without arguments for help. Then generate a set of core points for the scene.xyz file, using a spatial subsampling of 4 cm. Once this is done run the `../canupo` command on the scene, using its core points for faster computations (compare to the time used without core points).

Exercise 2: Run the `../density` *tool without arguments for help. Then generate density plots for the msc files that you generated. Look at how the vegetation and floor look like at different scales*

Step 3: Build a classifier from the samples with the following command:

```
  ../suggest_classifier_lda proposal.svg : vegetation.msc - floor.msc
```

Type `../suggest_classifier_lda` without arguments for help on the syntax. Basically, you are telling the program to write a classifier in the file `proposal.svg` that shall separate the two classes with only one sample in each class (one of vegetation, one of floor). Look at the overview sheet in the previous section for a snapshot of SVG file. You shall notice that two blobs are separated by a line. The blob on the left represents all the floor points, the one on the right all the vegetation points. You can now verify that the blobs are well separated. Read the article to understand how the transformation was performed.

Exercise 1: The SVG files are vector graphics file (and they are also text files). We recommend you to open the `proposal.svg` *file with the excellent Inkscape free software that can be downloaded at* http://www.inkscape.org/ .

Exercise 2: If you generated the multiscale features for the whole scene in Step 2, then use them as unlabelled information for improving the classifier. Refer to the help provided by `../suggest_classifier_lda` *without argument.Compare the resulting SVG files.*

Exercise 3: Run `../suggest_classifier_svm` *without arguments for help. Then run it instead of the LDA classifier in the example. Change the SVM search grid cross-validation size (the N parameter) and see how this affects the quality of the generated classifier. In practice the LDA classifier works as well if not better than the SVM and it is much faster.*

Step 4: Validate the classifier

```
  ../validate_classifier proposal.svg classifier.prm
```

This step simply means that you are happy with the default classifier.

Exercise: Edit the decision boundary path in the SVG file with Inkscape before validating it. For example by adding and moving a few points in the path. You can easily obtain powerful non-linear classifiers this way. This is also convenient if you want to favor one class over the other, for example here you may move the decision boundary away from the vegetation sample. Not really useful here, but in situations where the two blobs overlap, this is an easy way to make unbalanced classifiers (ex: you may want to be sure to remove all the vegetation even if you loose a bit more of floor at each vegetation patch base).

Step 5: Classify the scene. First, generate the multiscale representation for the whole scene if you did not do it already as an exercise in Step 2. Note that you can read the list of scales directly from the classifier file:

```
  ../canupo classifier.prm : scene.xyz scene.xyz scene.msc
```

Then classify the whole scene:

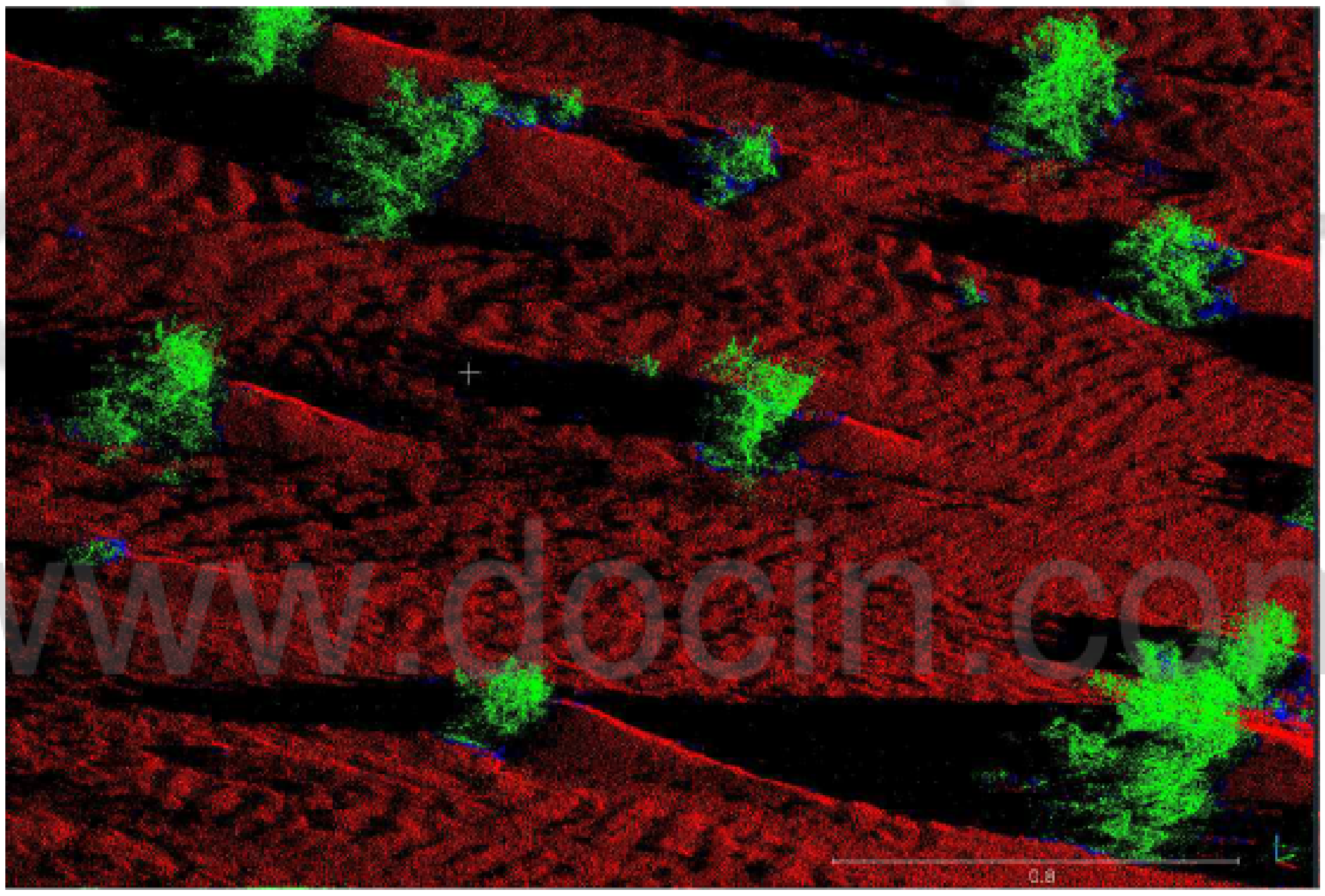
```
  ../classify classifier.prm scene.xyz scene.msc result.xyz
```

Open the `result.xyz` file with CloudCompare and specify in the open dialog that the fourth column is a “scalar>0” field. This will give each class a distinct color.

Exercise 1: Add a 0.95 confidence threshold, simply by adding 0.95 at the end of the classify command line above. Open the result file again. You shall notice that a third color was added for regions that cannot be classified with more than 95% confidence. Note how these are precisely at the base of the vegetation patches, and also how even small patches within riddles are correctly recognized.

Exercise 2: Use the scene core points from Step 2, excercise 1. Compare the computation time, but also zoom on a classified vegetation patch in CloudCompare in order to understand what tradoff was made: classification of each scene point is now assigned to that of the nearest core point.

Exercise 3: Use the `../filter` *utility in order to remove all the vegetation from the classified scene, using a condition on the fourth column that contain the class of each point.*



An extract of the example scene in this tutorial. The vegetation is displayed in green, the soil in red, zones for which the classification confidence is less than 95% are shown in blue. This is a snapshot using Cloud Compare of the result you shall obtain at Step 5, of the tutorial.

3. Building your own classifier with a Windows batch

If the vegetation classifiers that are currently provided on the website are not working in your case, or you want to try the classification for your own type of surfaces, then you have to go through the process of building the classifier. You can read the following quick description using windows batch and the tutorial with optional exercises that directly use command line instructions (in LINUX or WINDOWS) in section 3.

Selecting the training sets of the two classes

For the moment, CANUPO is designed for binary classifications. You need to choose a few representative training sets of the 2 types of surfaces you want to classify. The simplest thing to do is to create Core_Points files of 2 or 3 parts of each class in your point cloud (with something like 2000 to 5000 points per sample). These core_point files should be store in the Training_sets directory with a name typically core_toto.txt and core_tata.txt (see files already in the directory).

Selecting scales used in the classifier

Before to actually train the classifier you may want to have an idea of the dimensionality of your training sets at various scales (as in fig. 7 of Brodu and Lague, 2012). You can do that with the batch : density_analysis.bat. The text in the batch clearly highlights where to change names or scale. It's not necessary to use more than 5 or 6 scales ranging from 2-3 times your typical point spacing up to...something that depends on your application ! Then you can make sure that your choice of scales for the classifier will include at least the scales at which the density are the most different.

IMPORTANT : the computation time of the multiscale dimensionality parameters largely depends on the largest scale that you choose ! If you want the fastest classification possible, do not use very large scales compared to your typical point density (i.e. a max of ~ 1 m for min point spacing of 1 cm).

Training the classifier

Once you have chosen your set of scales (you'll probably want to try a few ranges as this can significantly change the classification result), use *batch classifier_training.bat* to create a first classifier, visualize the result (as in fig. 5) and obtain the balance accuracy (BA) and Fischer Discriminant Ratio (FDR) which are good indicators of how the classifier performs. If the BA is greater than 95 %, you have a very good classifier but it's not always possible to reach this high level (i.e., the gravel/water classifier in Brodu and Lague, 2012). To visualize the result, open the file class1.svg.

Manually modifying the classifier

Suggest_classifier.exe generates a file called CLASS1.svg in the root directory. You can open it with a browser and modify it using a free software like Inkscape. Then you just have to do the validate_classifier phase of the batch (i.e. you put "rem" on the lines you don't want to run) to recompute the classifier (creation of the .prm file) and have the ba and fdr value for your training set.

Check the success of the classifier on a subset of your data

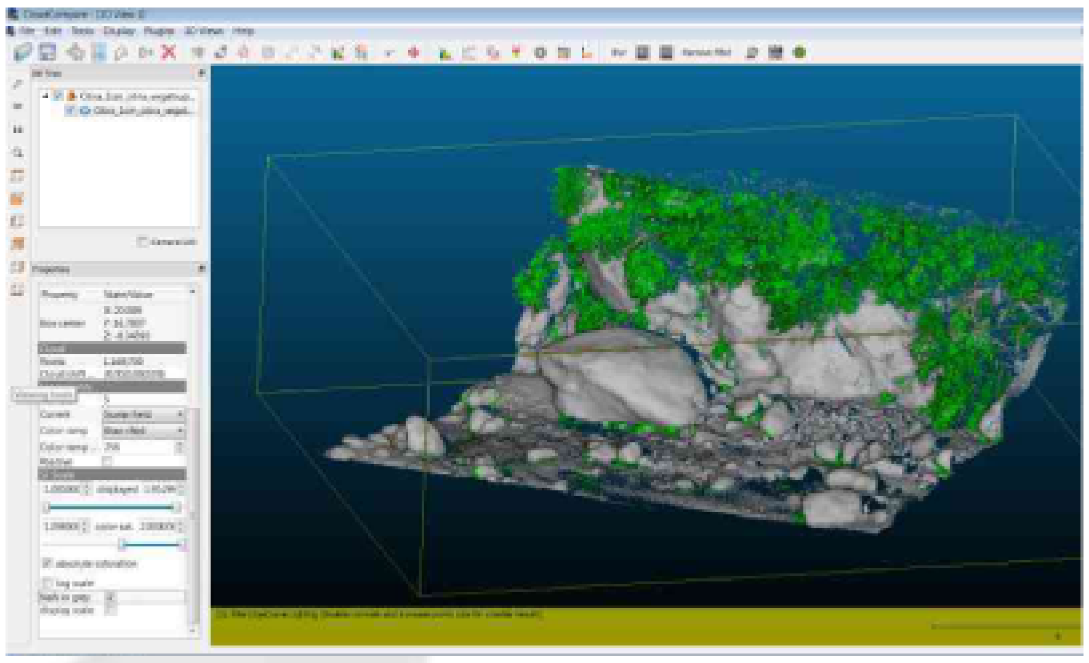
This is really the part where you'll see if the training sets where really representative or not ! If you find that the classifier does not perform as expected (lots of false positive or false negative), then a simple way to improve the classifier is to include in the training sets the false positive (i.e., vegetation classified as ground) and redo the classifier training. In my experience this is the best way to simply and rapidly improve the classifier rather than explore a different combination of scales.

Tips to build/improve your classifier

1. Choose your training sets as representative as possible of the class : this means for instance for vegetation : small and large elements, different species,....
2. Check the dimensionality content of your training set at various scales
3. Start with a typical range of scale that depends on your minimum point spacing without too many scales : for instance if your point spacing is of the order of 1-2 cm, a good starting range could be 5, 10, 15,20,30,40,50,60,70,80,90,100. The maximum scale depends on what you want to classify: if these are shrubs that are never larger than 20 cm, there's no need to go up to 100 cm ! It's more important to have more scales between in the range 3-20 cm than above. If you're interested in boulders that can be up to 1 m, then you'll need to include very large scales.
4. The best way to improve the classifier is to add to the training set the false positive and redo the training process. After that, you can manually tweak the classifier to better suit your needs, and increase the classification speed by removing the largest scales and checking whether the classification is still as good.

Example 1 : vegetation classification in the Otira gorge with Windows batch (example of Brodu and Lague, 2012)

You have nothing to do to run this example. Just make sure that you have the following files in the Data directory : Otira_1cm.txt (~ 1.1 million points) and Otira_10cm.txt (~ 0.1 million points). Then double-click on classification_only_otira.bat



BATCH to use : classification_only_otira.bat

Data_file : Otira_1cm.txt (~ 1.1 million points)

Core file : Otira_10cm.txt (~ 0.1 million points)

Classifier : otira_vegetsuper.prn

Scales (in m as in Otira_1cm.txt) : 0.02 0.04 0.06 0.08 0.1 0.125 0.15 0.175 0.2 0.225 0.25 0.275 0.3 0.35 0.4 0.45 0.5 0.6 0.7 0.8 0.9 1

Description of the batch

The first operation in the batch is the calculation of the multiscale dimensionality. The command line is the following :

```
%EXEDIR%\%CANUPO% %SCALE% : %DATADIR%\%DATA%.txt %DATADIR%\%CORE%.txt %MSCDIR%\%CORE%.msc 1
```

which means

```
.\exe\canupo.exe list_of_scales : .\Data\Otira_1cm.txt .\Datat\Otira_10cm.txt .\Msc_files\Otira_10cm.msc 1
```

The number 1 at the end of the command line specifies that the slope of the surface (from 0 to 90°) will be computed at the largest scale (here 1 m). This can be quite useful to automatically separate horizontal and vertical parts of the cloud. A new file Otira_10cm.msc has been created in the Msc_files directory. On my 2011 laptop (core i7 2720QM up to 3.4 Ghz, 7200 rpm HDD), this takes about 1 minute (the file loading is almost the longest part)

The second part is the classification :

```
%EXEDIR%\%CLASSIFY% %CLASSDIR%\%CLASSIFIER%.prm %DATADIR%\%DATA%.txt %MSCDIR%\%CORE%.msc
%RESDIR%\%DATA%_%CLASSIFIER%%GEN%.txt 0
```

which means

```
.\exe\classify.exe .\Classifiers\otira_vegetsuper.prn .\Data\Otira_1cm.txt .\Msc_files\Otira_10cm.msc .\Results\
Otira_1cm_otira_vegetsuper.txt
```

In that case, classify will use the multiscale dimensionality .msc file to assign the class at each core points (because the .msc is only computed at core points) but will apply the results on the Data file by a proximity factor : a point in the Data file get the class of the closest core point. You now have the classified file in the Results directory **Otira_1cm_otira_vegetsuper.txt.** On my laptop this takes about 1 minute.

Quick and nice visualization in Cloudcompare :

Load this file into Cloudcompare. Click on the EDL filter icon for real time shading (it's near the 'remove filter' icon).

Once loaded, click on the cloud in the DB tree panel of CC. On the SF scale part of the Properties Panel :

. Tick 'Absolute Saturation'

. Decrease the upper limit of the 'displayed level' slider to less than 2 and greater than 1.

. Increase the lower limit of the 'color sat' slider to more than 1 : the vegetation points (class_value=2) should be green, and the other points gray.

. Now tick or untick the Display NaN's, and you'll dynamically see only the vegetation layer.

Click on the min-max icon or go to menu->edit->scalar fields->filter by value. If you haven't changed anything, CC automatically use the currently displayed value (say 1 to 1.5). By clicking ok, CC automatically create a new point cloud for which the class_number is 1 (that is vegetation in our case). The old cloud is still there but not displayed anymore. Now you can change the scalar that is displayed to scalar #2 which is the confidence interval (varying between 0.5 and 1). By changing the 'displayed level' minimum you can set your appropriate level of confidence. Let's say that 0.95 is ok, now you can do a filter by value and only keep the point with a high quality of classification. You'll notice that a few points are not well classified on the river bed. Note also that because we have used a relatively coarse Core file (10 cm), if one core point is not well classified, this propagates to a lot of data points. You can redo these calculation using the data file as the core file, and you'll end up with less errors.

Batch filtering of the files :

The canupo software suites includes a parsing software that allows you to automatically extract the vegetation or non-vegetation part with a certain level of confidence.

For instance if in the previous file you want to keep only the non-vegetation points (class_value=2) with a level of confidence of 0.95, then you would write :

```
filter.exe .\Results\Otira_1cm_otira_vegetsuper.txt .\Results\Otira_1cm_classifiedcliff.txt 4:1.9:2.1 5:0.9:1
```

This added at the end of the batch with the following command :

```
rem %EXEDIR%\%FILTER% %RESDIR%\%DATA%_%CLASSIFIER%%GEN%.txt
%RESDIR%\%DATA%_class2.txt 4:1.9:2.1 5:0.95:1
```

remove the rem if you want this line to operate.

Protocol for exchanging classifiers

We hope to have an online catalogue of classifiers developed by our team and other users. We have not yet reached a final solution as to exchange all the information relevant to the classifier construction and application, but for the moment we suggest to transmit the following information :

. If possible (i.e. the original dataset is not too large or of restricted use), create a directory where you leave the batch that you have used to create the classifier, the training sets and the original .svg and .prm files. If you have manually modified the classified give the final version that you use. Explain in a readme.txt file the context for which the classifier was developped. This include :

. Typical minimum point spacing of the point cloud

. Scales used in the classifier

. Description of class 1

. Description of class 2

. Quality of classification (ba and fdr given by the validation of the classifier)

. General remarks

Help files of the main programs

Canupo.exe

```
canupo scales... : data.xyz data_core.xyz data_core.msc [flag]
  inputs: scales         # list of scales at which to perform the analysis
                         # The syntax minscale:increment:maxscale is also accepted
                         # Use : to indicate the end of the list of scales
                         # A scale correspond to a diameter for neighbor research.

  input: data.xyz        # whole raw point cloud to process\n\

  input: data_core.xyz   # points at which to do the computation. It is not necessary
that these points match entries in data.xyz: This means data_core.xyz need not be (but
can be) a subsampling of data.xyz, a regular grid is OK
                         # You can also take exactly the same file, or put more core
points than data points, the core points need only lie in the same region as data.
                         # Tip: use core points at least at max_scale distance from
the scene boundaries in order to avoid spurious multi-scale relations

  output: data_core.msc  # corresponding multiscale parameters at each core point
  input: flag            # (optional) if the flag is set to 1 then an additionnal
field is added into the output msc file for each core point: the angle (0<=a<=90°)
between the vertical and the normal of the best 2D plane fit at that core point, at
the largest given scale. 0 thus means a perfectly horizontal plane, 90 means a
perfectly vertical one
```

Classify.exe

```
classify classifier.prm scene.xyz scene_core.msc scene_annotated.xyz [pok
[usage_flag]]

  input: classifier.prm         # Classifier description

  input: scene.xyz            # Point cloud to classify/annotate with each class
                              # Text file, lines starting with #,!,;,// or with
                              # less than 3 numeric values are ignored
                              # If a 4rth value is present (ex: laser intensity) it
will be used in order to discriminate points too close to the decision boundary
                              # See also the usage_flag parameter

  input: scene_core.msc       # Multiscale parameters at core points in the scene
                              # This file need only contain the relevant scales for
classification

  output: scene_annotated.xyz # Output file containing extra columns: the class of
each point, the confidence in the classification, the number of neighbors at the min
and max scales. Scene points are labelled with the class of the nearest core point.

  input: pok                  # Some threshold, expressed as a probability to make
                              # a correct classification (0.5<pok<1). Use 0
                              # to disable the threshold, which is also the default
                              # Internally this is converted to a distance from
                              # the decision boundary matching that probability
                              # See the usage_flag argument for what pok means.
  input: usage_flag           # What to do with the perr argument if it is valid.
                              # The default is 0:
                              # - 0: mark as unclassified all points with confidence <
pok or equivalently too close to the decision boundary
```

```
                              # - 1: use the 4rth column in the data file as extra
information and train a local classifier to complement the confidence for points <
pok. This parameter has no effect if there is no 4rth value in the provided file.
```

Filter.exe

```
Arguments: input_file output_file constraint1 [constraint2 [...] ]

Files are text files with one variable per space-separated column
Constraints are written var_num:min:max
This says that the variable whose number is given (counting from one) must be in the
given range. Constraints are combined with AND, a line must respect all constraints to
be valid. The output file is written with only the valid lines from the input file.
```

Suggest_classifier_lda.exe (suggest_classifier_svm.exe)

```
suggest_classifier_lda outfile.svg [ unlabeled.msc ...] : class1.msc ... - class2.msc
...

input: class1.msc ... - class2.msc ...  # the multiscale files for each class,
separated by -

output: outfile.svg                     # a svg file in which to write a classifier
definition. This file may be edited graphically (ex: with inkscape) so as to add more
points in the path that separates both classes. So long as there is only one path
consisting only of line segments it shall be recognised.
input(optional): unlabeled.msc          # additionnal multiscale files for the scene
that are not classified. These provide more points for semi-supervised learning. The
corresponding points will be displayed in grey in the output file. If these are not
given the orthogonal to the center line is used.
```

Validate_classifier.exe

```
validate_classifier  user_modified.svg  classifier_file.prm  [ class_num_1
class_num_2 ] [: class1.msc ... - class2.msc ...]

input: user_modified.svg      # a svg file produced by suggest_classifier, possibly
updated by the user
output: classifier_file.prm   # produces a two-class classifier parameter file for use
by Classify.exe
input(optional): class_num_1 class_num_2  # the class number of the first and second
classes of this binary classifier
                                          # These can be specified for producing
multiclass classifiers by \"combine_classifier\"
                                          # otherwise they have value 1 and 2 by
default.
                                          # Class <=0 is reserved for points that
cannot be classified.
input(optional): class1.msc ... - class2.msc ...  # If these are given the classifier
performance can be estimated (see Brodu and Lague, 2012 for the signification of the
balance accuracy and the Fischer Discriminant Ratio).
```