Lumina

PrismRCL - Technical Documentation - v2.3.1.1

List of PrismRCL parameters and their functions

For installation instructions and how to get started, please refer to the attached “Start Guide”. Please note that although PrismRCL is a native Windows application, it is designed to run from the Windows command line as a developer tool. It does not have an interactive user interface. All logs and outputs are directed to text files on the file system and are fully configurable by the user as shown in this document and in the “Start Guide”.

Each of the following parameters can be added following a command line execution of PrimsRCL.exe

All parameters should be entered in all lowercase characters. Supports .png images and .txt files.

auto-optimize

automatically sets evaluation method, rclticks and boxdown (defined below) using a “greedy validation algorithm with projection”. auto-optimize uses the train portion of the data only, the resulting model can subsequently be tested with the test portion of the data. auto-optimize is a very efficient way of finding ideal parameters for new types of data not encountered before.

Example 1:

saves the parameters to the log directory in **_optimize_summary_mm_dd_yy_hh_mm_ss.txt**.

Note: an additional parameter, called ***imaginaryslice*** *might also be generated and stored in the summary file. If it is, be sure to pass it along with the other parameters.*

```
C:\PrismRCL\PrismRCL.exe auto-optimize data=C:\data\train_data
log=c:\log_files\
```

Example 2:

creates a model using parameters from auto-optimize (evaluation method, rclticks, boxdown are stored in model).

```
C:\PrismRCL\PrismRCL.exe auto-optimize data=C:\data\train_data
savemodel=c:\models\best_model.classify log=c:\log_files\
```

Example 3:

creates a model using parameters from auto-optimize (evaluation method, rclticks, boxdown are stored in model) and evaluates model on test data, all in one pass.

```
C:\PrismRCL\PrismRCL.exe auto-optimize data=C:\data\train_data
testdata=C:\data\test_data savemodel=c:\models\best_model.classify
log=c:\log_files\
```

loadmodel

Specifies an existing model to be loaded for inference or to have other models added to it.

Example:

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\model111.classify
testdata=c:\RCLC\data\test_data
```

transferlearn

Specifies an existing model to be augmented by additional training on new data of similar shape as within the model. The settings for subsequent training will be set to the same as the transferlearn model. If the shape of the new data is different from the model specified as transferlearn the system will cast an error to the directory specified under the log parameter.

Example:

```
C:\PrismRCL\PrismRCL.exe transferlearn=c:\models\goodmodel.classify
data= C:\data\moretrainingdata savemodel=c:\models\bettermodel.classify
```

data

Specifies the folder for training data (and possibly test data). The ratio between the two can be set using testsize parameter (default testsize=0.1)

The data parameter can be applied stand-alone alone or in conjunction with the transferlearn parameter. If the data parameter is specified along with the loadmodel parameter.

```
C:\PrismRCL\PrismRCL.exe data=c:\data\train_data
savemodel=c:\models\mymodel.classify
```

From the folder specified as data the classes should be folders underneath with their respective .png or .txt files in each folder. *Please note that all file names of .png or .txt files must be unique across classes.*

addmodel

Specifies a model or set of models (separated by semicolon) to be added to an existing model. The add requires no additional subsequent training run to be functional at the accumulated models, thus the *addmodel requires a loadmodel to be in the same command*. If the training parameters for a loadmodel are different from the model specified as addmodel the system will cast an error to the directory specified under the log parameter. Individual models must also have been created using chunks from the same dataset.

Examples:

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\bettermodel.classify
addmodel=c:\models\ model222.classify
savemodel=c:\models\bettermodelx2.classify
```

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\bettermodel.classify
addmodel=c:\models\model111.classify;c:\models\ model222.classify
savemodel=c:\models\bestmodel.classify
```

savemodel

Specifies the resulting model for any of the above steps.

See example above.

testsize

As mentioned in the data section, testsize specifies the size of the split from the data folder to be served for testing of a trained model. If it is desired to test on a loadmodel (with subsequent addmodel) then use the testdata setting below. Default is testsize=0.1, but for example if a test of 20% of the data is desired, then pass the parameter testsize=0.2

testdata

This parameter specifies a folder dedicated to test data only. This is typically the parameter to use when inferring with an existing model.

Examples:

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\model333.classify
testdata=c:\data\test_data testsize=0.2
```

savedata

use this parameter to save a dedicated test set that is equivalent to that used in memory as defined by testsize. This applies only to a run where there is training involved; otherwise, the folder referred should simply be passed as testdata (see above). The files in the resulting folder will be copies of their originals.

Example:

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\model444.classify
testdata=c:\data\test_data testsave=c:\data\saveforlater
```

chisquared

Is a single parameter (no = value should be passed).

It is also applied by default if no evaluation-type is passed.

“chisquared” (Chi-squared) evaluates all patterns assuming the null hypothesis is true, i.e., all patterns are random. The class with the most “pull” away from random wins.

fractal

Is a single parameter (no = value should be passed).

“fractal” evaluates all patterns in accordance with their distance from a Probability Density Function (PDF) that assumes self-similarity (fractals) in the images. The class with the least Sum Squared Error (SSE) wins.

naivebayes

Is a single parameter (no = value should be passed).

“naivebayes” (Naive Bayes) evaluates all patterns as independent with random as contrast for all other classes. The class with the most “pull” away from other classes wins.

imaginary

Is a single parameter (no = value should be passed).

“imaginary” can be applied to all evaluation methods.

When “imaginary” is passed along with a given evaluation method, RCLC will collapse low frequency patterns into an imaginary pattern and use this as part of the given evaluation method selected. The effect of using “imaginary” will (depending on the data) result in smaller models with faster inference.

NOTE: that “imaginary” setting is lossy, and thus adding models with different settings for imaginary is not possible. “imaginary” may work well for certain types of data but can be too lossy for others.

trainacc

Is a single parameter (no = value should be passed). This setting forces RCLC to also output training accuracy. In general, it is desired only to use this, when necessary, since the training set can otherwise be freed from memory incrementally as RCLC trains.

Example:

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\model941.classify
testdata=c:\RCLC\data\test_data trainacc testsave=c:\data\saveforlater
```

channelpick

Is for image training and inference only. channelpick allows RCLC to train on specific RGB channels of an image or either average or a combination as follows:

1: red
2: green
3: blue
4: average of red, green, and blue
5: combining all (default setting)

rclticks

Is for image training and inference only. It is a setting for how granular RCLC should interpret a value from channelpick. Default setting is 10 meaning that RCLC discretizes a byte into round(255/10) values.

filter

Is the size of RCLC’s sliding window through which an image is observed. Default setting is filter=3

stride

Stride is the movement of the filter over an image. Default setting is stride=1

Interpretation of the filter:

RCLC offers two different approaches to interpreting what is "seen" in the filter: **cluster** and **directional**.

"cluster" is robust to angle and direction of an image. (This is the default setting)

"directional" is more detailed and interprets direction.

Folding of images:

PrimsRCL offers two styles of folding an image into a smaller area over which the filter is passed: boxdown and maxdown. boxdown does a pixel average for shrink whereas maxdown takes the maximum. By default, none of these techniques are applied. To activate them, use either boxdown=1 or maxdown=1 (to fold once). Any integer can be applied, but obviously the size of an image puts a limitation to how many folds make sense.

Example of the settings:

```
C:\PrismRCL\PrismRCL.exe data=c:\data\train_data boxdown=0 filter=3
rclticks=20 saveif=0.9 stride=1 testsize=0.1
```

(The settings for down, filter, stride, and testsize are defaults; but for example written out here)

inftoimage

This inference style lets you pick a model and a folder for testdata, then it classifies the images from the testdata folder and organizes them in accordance with classification. In addition, it augments the classified images to grayscale while leaving only the pixels that are the algorithms "reasons" for suggesting the image belongs to a particular class.

Example:

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\bestmodel.classify
testdata=c:\data\test_data inftoimage=c:\results\images_out
```

inftotext

This inference style lets you pick a model and a folder for testdata, then it classifies the images from the testdata and provides a list of the classes per file at the location identified by inftotext.

Example:

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\bestmodel.classify
testdata=c:\data\test_data inftotext=c:\results\output.txt
```

textinftotext

This inference style lets you pick a model and a folder for testdata, then it classifies the text files from the testdata and provides a list of the classes per file at the location identified by textinftotext.

Example:

```
C:\PrismRCL\PrismRCL.exe chisquared
loadmodel=c:\models\textmodel.classify testdata=C:\data\chat_text_test
textinftotext=c:\results\test_out.txt
```

trainthreads

trainthreads apply the resources needed under training a model (the thread count specifically).

This is a setting that gives the ability to throttle the training machine from running out of resources. It is generally recommended (based on experiments across 4 types of hardware configurations) to set trainthreads to the be equal to the number of logical processors the machine. Default is trainthreads=20

infthreads

infthreads apply the resources needed during inference with a model (the thread count specifically).

This is a setting that gives the ability to throttle the training machine from running out of resources. Since inference generally is way less computationally expensive, this number can be (much) higher than the trainthreads setting. Based on experiments across 4 types of hardware configurations we recommend setting this to either infthreads=1000 or infthreads=2000.

It is generally recommended (based on experiments across 4 types of hardware configurations) to set trainthreads to the be equal to the number of logical processors the machine. Default is infthreads=1000

maxcpu

Is another "handle" to controlling the physical machine from running out of resources. This regulates how high the CPU usage can be (in percentage) before RCLC beings to "throttle down".

The default setting is maxcpu=100

maxmem

Is the last handle that can be used to control RCLC's behavior on the physical machine. Perhaps the most critical because running out of memory and thereby initiating memory/disk swapping has the worst throughput penalty of all.

Based on experiments on a 64 GB machine has led to setting this default to maxmem=30

log

This sets the location of the logfiles that are generated during certain checkpoints to monitor progress of RCLC. Default setting is log= c:\temp\logfiles

temp

This sets the location of the temporary files writing during saving of data and models.

Default setting is temp= c:\temp

outputfile

This setting is optional and typically used for benchmarking. This setting gives a file-location where a file will summarize the following: timestamp, settings, total training time (secs), total inference time (secs), inference time per image (secs), as well as the test accuracy with untrained data.

Combined with this the parameter **saveif** can be used to control output only if test/validation accuracy is greater than this setting.

Example:

```
C:\PrismRCL\PrismRCL.exe data=c:\data\train_data saveif=0.9
rclticks=20 outputfile=c:\RCLC\results\output.txt
```

If the value for is set, RCLC will automatically shut down so multiple runs can be scripted in sequence.

stopwhendone

in case it is desired to shut RCLC down automatically after a run, passing stopwhendone will do this.

Example:

```
C:\PrismRCL\PrismRCL.exe loadmodel=c:\models\bettermodel.classify
addmodel=c:\models\model945.classify;c:\models\model555.classify
savemodel=c:\models\bestmodel.classify stopwhendone
```

promptest

This setting prompts the user before running inference, on whether to run (yes) or not to run (no).

This setting is useful for demonstrating RCLC as well as when measuring inference speeds.

silenttest

This setting will limit test output to confusion matrix only. "silenttest" is useful for demonstrating RCLC.