hwrt Documentation

Release 0.1.119

Martin Thoma

October 27, 2014

Contents

hwrt is short for 'handwriting recognition toolkit'. This toolkit allows you to download on-line handwritten mathematical symbols, view them, analyze them and train and test models to classify them automatically. The toolset offers many preprocessing algorithms and features that can be combined in many ways by YAML configuration files.

The theoretical part is covered in the bachelor's thesis 'On-line Handwriting Recognition of Mathematical Symbols' from Martin Thoma. One part of this bachelor's thesis was to create this toolkit and evaluate it.

All project source code and the source code of this documentation is at github.com/MartinThoma/hwrt. The experiments are at github.com/MartinThoma/hwr-experiments.

If you want to talk about this toolkit, you can contact me (Martin Thoma) via email: info@martin-thoma.de

Contents:

CHAPTER 1

Installation

The `hwrt` toolkit can be installed via pip:

```
# pip install hwrt
```

However, you might have to install some packages first for `scipy`.

1.1 Debian-based systems

Debian-based systems are Ubuntu, Linux Mint, and of course Debian. If you have such a Linux system, then the following commands will help you to install the `hwrt` package.

For `scipy`, `numpy` and `matplotlib` you might need these packages:

```
# apt-get install python-pip libblas-dev liblapack-dev gfortran
# apt-get install python-scipy python-numpy
# apt-get install libfreetype6-dev
# apt-get install libgeos-dev
```

If you want to use MySQL functionality, you will need

```
# apt-get install libmysqlclient-dev
```

1.2 Python packages

```
# pip install MySQL-python
```

Now you can install the remaining packages:

```
# pip install natsort matplotlib coveralls shapely
# pip install numpy
# pip install scipy
```

Now you can install pfile_utils. Some explanation of what they are can be found at my blog

As a last step, you can install hwrt:

```
# pip install hwrt
```

1.3 Test installation

You can check if it worked by

```
$ hwrt --version
hwrt 0.1.101
```

Please send me an email (info@martin-thoma.de) if that didn't work.

1.3.1 First steps

First of all, you should download the raw data. This is done by executing `download.py`.

Next, you can view a simple example by `view.py`. For example, with `view.py --list` you can view all raw data IDs of your current data. With `view.py -i 291075` you can see how the preprocessing steps and the later data multiplication steps influence the recording. If you didn't execute view from a model folder and if you didn't specify another model with `-m`, you will get the output of the small baseline model that was created in your projects root folder (`~/hwr-experiments` per default, but you can modify that with `~/.hwrtrc`). That will show 3 rotated images of π.

If you want to see more examples, have a look at https://github.com/MartinThoma/hwr-experiments

1.3.2 nntoolkit

In order to use *hwrt* completely (especially testing, training and record.py) you have to have an executable `nntoolkit` that supports the following usages:

```
$ nntoolkit run --batch-size 1 -f%0.4f <test_file> < <model>
```

has to output the evaluation result in standard output as a list of floats separated by newlines `\n+`. The evaluation result might either be the index of the neuron with highest activation or the list of probabilities of each class separated by spaces.

```
$ nntoolkit make mlp <topology>
```

has to print the model in standard output.

The *hwrt* toolset is independent of the way the training command is formatted as the training command gets inserted directly into the configuration file `info.yml` of the model.

In order to implement such a neural network executable one can use Theano, cuDNN or Caffe. Deeplearning contains example code for multilayer perceptrons written with Theano (Python).

1.3.3 Upgrading hwrt

Upgrading hwrt to the latest version is much easier:

```
# pip install hwrt --upgrade
```

CHAPTER 2

Configuration

The `hwrt` toolkit makes use of a configuration file. This file has to be in the home folder of the user an it has to be called `.hwrtrc`.

The configuration file is in YAML format. The possible values are:

- `root`: This is a required configuration entry. Its value must be a path. hwrt will look for all configuration files in this path.
- `nntoolkit`: The name of the executable in your path that does neural network training
- `preprocessing`: A path to a Python script that contains your preprocessing classes. Have a look at the official preprocessing classes to see how they should be structured.
- `features`: Just like preprocessing, this has to be a path to a Python script.

There are 3 configurations that are probably only interesting for me:

- `dbconfig`: Only important if you want to access a MySQL db to get the data
- `dropbox_app_key` and `dropbox_app_secret`: Only important if you want to upload data to DropBox

2.1 Example

The following is an example `~/.hwrtrc` configuration file:

```
root: /home/moose/GitHub/hwr-experiments
nntoolkit: nntoolkitfancyname
preprocessing: /home/moose/Desktop/preprocessing.py
features: /home/moose/Desktop/features.py
dbconfig: /home/moose/Downloads/write-math/tools/db.config.yml
dropbox_app_key: 'INSERT_APP_KEY'
dropbox_app_secret: 'INSERT_APP_SECRET'
```

CHAPTER 3

Handwritten Data

Representation of a recording of on-line handwritten data. On-line means that the pen trajectory is given (and not online as in 'Internet').

class hwrt.HandwrittenData.**HandwrittenData**(*raw_data_json*, *formula_id=None*, *raw_data_id=None*, *formula_in_latex=None*, *wild_point_count=0*, *missing_stroke=0*, *user_id=0*)

Represents a handwritten symbol.

count_single_dots()
Count all strokes of this recording that have only a single dot.

feature_extraction(*algorithms*)
Get a list of features.

Every algorithm has to return the features as a list.

get_area()
Get the area in square pixels of the recording.

get_bounding_box()
Get the bounding box of a pointlist.

get_center_of_mass()
Get a tuple (x,y) that is the center of mass. The center of mass is not necessarily the same as the center of the bounding box. Imagine a black square and a single dot wide outside of the square.

get_height()
Get the height of the rectangular, axis-parallel bounding box.

get_pointlist()
Get a list of lists of tuples from JSON raw data string. Those lists represent strokes with control points.

Every point is a dictionary: {'x': 123, 'y': 42, 'time': 1337}

get_sorted_pointlist()
Make sure that the points and strokes are in order.

get_time()
Get the time in which the recording was created.

get_width()
Get the width of the rectangular, axis-parallel bounding box.

preprocessing(*algorithms*)
Apply preprocessing algorithms.

```
>>> a = HandwrittenData(...)
>>> preprocessing_queue = [(preprocessing.scale_and_shift, []),
>>> a.preprocessing(preprocessing_queue)
```

set_pointlist(*pointlist*)

Overwrite pointlist. :param pointlist: The inner lists represent strokes. Every stroke consists of points. Every point is a dictinary with 'x', 'y', 'time'.

show()

Show the data graphically in a new pop-up window.

CHAPTER 4

Preprocessing

Preprocessing algorithms.

Each algorithm works on the HandwrittenData class. They have to be applied like this:

```
>>> a = HandwrittenData(...)
>>> preprocessing_queue = [ScaleAndShift(),
                           StrokeConnect(),
                           DouglasPeucker(epsilon=0.2),
                           SpaceEvenly(number=100)]
>>> a.preprocessing(preprocessing_queue)
```

class hwrt.preprocessing.**DotReduction**(*threshold=5*)
Reduce strokes where the maximum distance between points is below a *threshold* to a single dot.

class hwrt.preprocessing.**DouglasPeucker**(*epsilon=0.2*)
Apply the Douglas-Peucker stroke simplification algorithm separately to each stroke of the recording. The algorithm has a threshold parameter *epsilon* that indicates how much the stroke is simplified. The smaller the parameter, the closer will the resulting strokes be to the original.

class hwrt.preprocessing.**RemoveDots**
Remove all strokes that have only a single point (a dot) from the recording, except if the whole recording consists of dots only.

class hwrt.preprocessing.**RemoveDuplicateTime**
If a recording has two points with the same timestamp, than the second point will be discarded. This is useful for a couple of algorithms that don't expect two points at the same time.

class hwrt.preprocessing.**ScaleAndShift**(*center=False*, *max_width=1.0*, *max_height=1.0*, *width_add=0*, *height_add=0*, *center_other=False*)
Scale a recording so that it fits into a unit square. This keeps the aspect ratio. Then the recording is shifted. The default way is to shift it so that the recording is in [0, 1] × [0,1]. However, it can also be used to be centered within [-1, 1] × [-1, 1] around the origin (0, 0) by setting center=True and center_other=True.

class hwrt.preprocessing.**SpaceEvenly**(*number=100*, *kind='cubic'*)
Space the points evenly in time over the complete recording. The parameter 'number' defines how many.

class hwrt.preprocessing.**SpaceEvenlyPerStroke**(*number=100*, *kind='cubic'*)
Space the points evenly for every single stroke separately. The parameter *number* defines how many points are used per stroke and the parameter *kind* defines which kind of interpolation is used. Possible values include *cubic*, *quadratic*, *linear*, *nearest*. This part of the implementation relies on `scipy.interpolate.interp1d`.

class hwrt.preprocessing.**StrokeConnect**(*minimum_distance=0.05*)
StrokeConnect: Detect if strokes were probably accidentally disconnected. If that is the case, connect them. This is detected by the threshold parameter *minimum_distance*. If the distance between the end point of a stroke and the first point of the next stroke is below the minimum distance, the strokes will be connected.

class hwrt.preprocessing.**WeightedAverageSmoothing**(*theta=None*)
Smooth every stroke by a weighted average. This algorithm takes a list *theta* of 3 numbers that are the weights used for smoothing.

class hwrt.preprocessing.**WildPointFilter**(*threshold=3.0*)
Find wild points and remove them. The threshold means speed in pixels / ms.

hwrt.preprocessing.**get_class**(*name*)
Get the class by its name as a string.

hwrt.preprocessing.**get_preprocessing_queue**(*preprocessing_list*)
Get preprocessing queue from a list of dictionaries

```
>>> l = [{'RemoveDuplicateTime': None},
         {'ScaleAndShift': [{'center': True}]}
        ]
>>> get_preprocessing_queue(l)
[RemoveDuplicateTime, ScaleAndShift
 - center: True
 - max_width: 1
 - max_height: 1
]
```

CHAPTER 5

Data Multiplication

The automatic data multiplication works like this:

```
for algorithm in multiplication_queue:
    new_training_set = []
    for recording in training_set:
        samples = algorithm(recording)
        for sample in samples:
            new_training_set.append(sample)
    training_set = new_training_set
return new_training_set
```

Warning: The create_pfile procedure replaces the current set of recordings by the set returned by the data multiplication steps.

Data multiplication algorithms.

Each algorithm works on the HandwrittenData class. They have to be applied like this:

```
>>> import data_multiplication as multiply
>>> a = HandwrittenData(...)
>>> multiplication_queue = [multiply.copy(10),
                            mulitply.rotate(-30, 30, 5)
                            ]
>>> x = a.multiply(multiplication_queue)
```

class hwrt.data_multiplication.**Multiply**(*nr=1*)
Copy the data n times.

class hwrt.data_multiplication.**Rotate**(*minimum=-30.0*, *maximum=30.0*, *num=5.0*)
Add rotational variants of the recording.

hwrt.data_multiplication.**get_class**(*name*)
Get function pointer by string.

hwrt.data_multiplication.**get_data_multiplication_queue**(*model_description_multiply*)
Get features from a list of dictionaries

```
>>> l = [{'Multiply': [{'nr': 1}]},              {'Rotate': [{'minimum':-30}, {'maximum': 30}, {
>>> get_data_multiplication_queue(l)
[Multiply (1 times), Rotate (-30.00, 30.00, 5.00)]
```

CHAPTER 6

Features

Feature extraction algorithms.

Each algorithm works on the HandwrittenData class. They have to be applied like this:

```
>>> import features
>>> a = HandwrittenData(...)
>>> feature_list = [features.StrokeCount(),
                    features.ConstantPointCoordinates(strokes=4,
                                                      points_per_stroke=20,
                                                      fill_empty_with=0)
                    ]
>>> x = a.feature_extraction(feature_list)
```

class hwrt.features.**AspectRatio**
Aspect ratio of a recording as a 1 dimensional feature.

class hwrt.features.**Bitmap**(*n=28*)
n × n grayscale bitmap of the recording.

class hwrt.features.**CenterOfMass**
Center of mass of a recording as a 2 dimensional feature.

class hwrt.features.**ConstantPointCoordinates**(*strokes=4*, *points_per_stroke=20*, *fill_empty_with=0*, *pen_down=True*)

Take the first *points_per_stroke=20* points coordinates of the first *strokes=4* strokes as features. This leads to $2 \cdot extpoints\_per\_stroke \cdot extstrokes$ features.

If *points* is set to 0, the first *points_per_stroke* point coordinates and the

erb+pen_down+ feature is used. This leads to $3 \cdot extpoints_per_stroke$ features.

class hwrt.features.**FirstNPoints**(*n=81*)
Similar to the *ConstantPointCoordinates* feature, this feature takes the first *n=81* point coordinates. It also has the *fill_empty_with=0* to make sure that the dimension of this feature is always the same.

class hwrt.features.**Height**
Height of a recording as a a 1 dimensional feature.

Note: This is the current hight. So if the recording was scaled, this will not be the original height.

class hwrt.features.**Ink**
Ink as a 1 dimensional feature. It gives a numeric value for the amount of ink this would eventually have consumed.

class hwrt.features.**ReCurvature**(*strokes=4*)

Re-curvature is a 1 dimensional, stroke-global feature for a recording. It is the ratio :math:‘

rac{ ext{height}(s)}{ ext{distance}(s[0], s[-1])}‘.

class hwrt.features.**StrokeCenter**(*strokes=4*)
Get the stroke center of mass coordinates as a 2 dimensional feature.

class hwrt.features.**StrokeCount**
Stroke count as a 1 dimensional recording.

class hwrt.features.**StrokeIntersections**(*strokes=4*)

Count the number of intersections which strokes in the recording have with each other in form of a symmetrical matrix for the first *stroke=4* strokes. The feature dimension is :math:‘round

rac{ ext{strokes}^2}{2} + rac{ ext{strokes}}{2}‘,

because the symmetrical part is discarded.

•	stroke1	stroke2	stroke3	
stroke1	0	1	0	...
stroke2	1	2	0	...
stroke3	0	0	0	...

Returns values of upper triangular matrix (including diagonal) from left to right, top to bottom.

class hwrt.features.**Time**
The time in milliseconds it took to create the recording. This is a 1 dimensional feature.

get_dimension()
Get the dimension of the returned feature. This equals the number of elements in the returned list of numbers.

class hwrt.features.**Width**
Width of a recording as a 1 dimensional feature.

Note: This is the current width. So if the recording was scaled, this will not be the original width.

---

hwrt.features.**get_class**(*name*)
Get function pointer by string.

hwrt.features.**get_features**(*model_description_features*)
Get features from a list of dictionaries

```
>>> l = [{'StrokeCount': None},              {'ConstantPointCoordinates':              [{'strok
>>> get_features(l)
[StrokeCount, ConstantPointCoordinates
 - strokes: 4
 - points per stroke: 81
 - fill empty with: 0
 - pen down feature: False
]
```

CHAPTER 7

Create pfiles

Create pfiles.

Before this script is run, the *download.py* should get executed to generate a handwriting_datasets.pickle with exactly those symbols that should also be present in the pfiles and only raw_data that might get used for the test-, validation- and training set.

hwrt.create_pfiles.**get_sets**(*path_to_data*)

Get a training, validation and a testset as well as a dictionary that maps each formula_id to an index (0...nr_of_formulas).

Parameters **path_to_data** – a pickle file that contains a list of datasets.

hwrt.create_pfiles.**make_pfile**(*dataset_name*, *feature_count*, *data*, *output_filename*, *create_learning_curve*)

Create the pfile. :param filename: name of the file that pfile_create will use to create the

pfile.

Parameters

- **feature_count** (*integer*) – integer, number of features
- **data** (*list of tuples*) – data format ('feature_string', 'label')

hwrt.create_pfiles.**prepare_dataset**(*dataset*, *formula_id2index*, *feature_list*, *is_traindata*)

Transform each instance of dataset to a (Features, Label) tuple.

hwrt.create_pfiles.**training_set_multiplication**(*training_set*, *mult_queue*)

Multiply the training set by all methods listed in mult_queue. :param training_set: set of all recordings that will be used for training :param mult_queue: list of all algorithms that will take one recording and

generate more than one.

Returns mutliple recordings

CHAPTER 8

Plugins

You eventually want to create your own preprocessing steps, your own features or another implementation of the same feature. You can do so by specifying a Python script in `preprocessing` or `features`.

If a preprocessing class or a feature class exists in the official hwrt and in a plugin simultaniously, the hwrt implementation is used.

8.1 Preprocessing Classes

Every feature class must have a `__str__`, `__repr__` and a `__call__` function where

- `__call__` must take exactly one argument of type HandwrittenData
- `__call__` must call the `Handwriting.set_points`

8.2 Feature Classes

Every feature class must have a `__str__`, `__repr__`, `__call__` and `get_dimension` function where

- `__call__` must take exactly one argument of type HandwrittenData
- `__call__` must return a list of length `get_dimension()`
- `get_dimension` must return a positive number
- have a 'normalize' attribute that is either true or false

8.3 Preprocessing Plugin Example

```
#!/usr/bin/env python

import hwrt.HandwrittenData as HandwrittenData


class Nullify(object):
    def __repr__(self):
        return "Nullify"

    def __str__(self):
```

```
        return "Nullify"

    def __call__(self, handwritten_data):
        assert isinstance(handwritten_data, HandwrittenData.HandwrittenData), \
            "handwritten data is not of type HandwrittenData, but of %r" % \
            type(handwritten_data)
        # pointlist = handwritten_data.get_pointlist()
        new_pointlist = []
        new_stroke = []
        new_stroke.append({'x': 0, 'y': 0, 'time': 0})
        new_pointlist.append(new_stroke)
        handwritten_data.set_pointlist(new_pointlist)
```

8.4 Feature Plugin Example

```
#!/usr/bin/env python

import hwrt.HandwrittenData as HandwrittenData


class StrokeCountTata(object):

    """Stroke count as a 1 dimensional recording."""

    normalize = True

    def __repr__(self):
        return "StrokeCount"

    def __str__(self):
        return "stroke count"

    def get_dimension(self):
        return 1

    def __call__(self, handwritten_data):
        assert isinstance(handwritten_data, HandwrittenData.HandwrittenData), \
            "handwritten data is not of type HandwrittenData, but of %r" % \
            type(handwritten_data)
        return [len(handwritten_data.get_pointlist())]
```

CHAPTER 9

Development

The `hwrt` toolkit is developed by Martin Thoma. The development began in May 2014.

It is developed on GitHub: https://github.com/MartinThoma/hwrt

You can file issues and feature requests there. Alternatively, you can send me an email: info@martin-thoma.de

9.1 Tools

- `nosetests` for unit testing
- `pylint` to find code smug
- GitHub for hosting the source code
- http://hwrt.readthedocs.org/ or https://pythonhosted.org/hwrt for hosting the documentation

Code coverage can be tested with

```
$ nosetests --with-coverage --cover-erase --cover-package hwrt --logging-level=INFO --cover-html
```

and uploaded to coveralls.io with

```
$ coveralls
```

Binaries:

CHAPTER 10

Download Raw Data

This tool helps to get the initial raw data from the internet to your computer.

```
$ download.py --help
usage: download.py [-h]

Check if data files are here and which version they have. Contact the server
for the latest version and update them if they are outdated.

optional arguments:
  -h, --help  show this help message and exit
```

CHAPTER 11

Analyze Data

This tool helps to analyze data by features.

```
$ analyze_data.py --help
usage: analyze_data.py [-h] [-d FILE]

Analyze data in a pickle file by maximum time / width / height and similar
features.

optional arguments:
  -h, --help            show this help message and exit
  -d FILE, --handwriting_datasets FILE
                        where are the pickled handwriting_datasets? (default:
                        /home/moose/Downloads/write-math/archive/raw-
                        datasets/2014-08-26-20-14-handwriting_datasets-
                        raw.pickle)
```

CHAPTER 12

View Data

This tool lets you view a single recording. You can apply preprocessing steps by specifying a model folder.

```
$ view.py --help
usage: view.py [-h] [-i ID] [--mysql MYSQL] [-m FOLDER] [-l]

Display a raw_data_id.

optional arguments:
  -h, --help            show this help message and exit
  -i ID, --id ID        which RAW_DATA_ID do you want? (default: 279062)
  --mysql MYSQL         which mysql configuration should be used? (default:
                        mysql_online)
  -m FOLDER, --model FOLDER
                        where is the model folder (with a info.yml)? (default:
                        /home/moose/GitHub/hwr-experiments/models/small-
                        baseline)
  -l, --list            list all raw data IDs / symbol IDs (default: False)
```

CHAPTER 13

Model Training

This tool trains a model. This requires a KIT internal toolkit or your own implementation

```
$ train.py --help
usage: train.py [-h] [-m FOLDER]

Create and train a given model.

optional arguments:
  -h, --help            show this help message and exit
  -m FOLDER, --model FOLDER
                        where is the model folder (with a info.yml)? (default:
                        /home/moose/Downloads/write-math/archive/models/small-
                        baseline)
```

CHAPTER 14

Record and Evaluate New Data

The *hwrt* toolkit contains a script *record.py* that allows a client to record and evaluate. The script uses the current working directory as the recognition system. That means it uses the same preprocessing queue, the same features and the latest model file of that folder. The script gets this information by examining the *info.yml*.

CHAPTER 15

Model Testing

This tool calculates the error of a model according to different error measures.

The options *-n* and *–merge* should not be used together.

```
$ test.py --help
usage: test.py [-h] [-m FOLDER] [-s {test,train,valid}] [-n N] [--merge]

Get the error of a model. This tool supports multiple error measures.

optional arguments:
  -h, --help            show this help message and exit
  -m FOLDER, --model FOLDER
                        where is the model folder (with the info.yml)?
                        (default: current folder)
  -s {test,train,valid}, --set {test,train,valid}
                        which set should get analyzed? (default: test)
  -n N                  Top-N error (default: 3)
  --merge               merge problem classes that are easy to confuse
                        (default: False)
```

CHAPTER 16

Backup from MySQL server

This tool is only useful if you also have a write-math MySQL server running.

```
$ backup.py --help
usage: backup.py [-h] [-d FOLDER] [-s] [-o]

Download raw data from online server and back it up (e.g. on DropBox)
handwriting_datasets.pickle.

optional arguments:
  -h, --help            show this help message and exit
  -d FOLDER, --destination FOLDER
                        where do write the handwriting_dataset.pickle
                        (default: /home/moose/Downloads/write-math/archive
                        /raw-datasets)
  -s, --small           should only a small dataset (with all capital letters)
                        be created? (default: False)
  -o, --onlydropbox     don't download new files; only upload to DropBox
                       (default: False)
```

CHAPTER 17

Indices and tables

- *genindex*
- *modindex*
- *search*

h

T

W