ETCAL software description

ETCAL library is a convenient tool to calibrate data obtained by any kind of eye tracker.

ETCal interface

The work starts with creation of ETCal object. Then it is necessary to feed it with data using ***add(DataUnits)*** method. It may be done multiple times and new data is added to previously added. It is also possible to delete all data using ***reset()*** method.

The next step is defining or applying filters. There are two methods: ***useFilter(f)*** that applies filter to the current data at once and ***addFilter(f)*** that stores the filter and applies it every time just before model building. It is possible to add more than one filter and they are executed in chain. It is possible to remove all filters with ***resetFilters()*** method.

When all DataUnit objects are loaded and filters are defined (or directly applied) it is possible to perform one of the three operations:

build(calibrator) – builds a regression model using the calibrator supplied as a parameter

optimize(optimizer) – uses the supplied optimizer to search for the best regression model

mapTargets(mapper) – uses the mapper supplied to find for the best mapping of targets. Only feasible when data contains DataUnit objects with multiple Target objects. mapTarget() method searches for the best sequence of targets and when it is ready this mapping sequence automatically is used by build() and optimize() method.

When build() or optimize() methods fulfill its tasks it is possible to calibrate new data. There are two methods that may be used:

get(double[] x) – for a given array of input features returns the ***Target*** object with gaze coordinates using the previously calculated regression model.

get(DataUnits input) – for a given list of DataUnit objects (each containing an array of input features) returns the list of ***DataUnit*** each containing the calculated Target using the previously calculated regression model. The Target objects in the input object are not taken into account!

There is also additional helper method that calculates errors:

checkErrors(DataUnits input) – calculates targets for every DataUnit and then compares it with targets in the input object. Returns ***Errors*** object containing: Rsquare, MSE, AbsError for both axes.

Generic objects

The library contains several ready to use filters, calibrators for model building, several optimizers and one mapper. It is possible to use them directly by initializing the proper object and providing it as a parameter of a method addFilter(filter), build(calibrator), optimize(optimizer) and mapTargets(mapper).

However, because the software is open for creation new custom filters, calibrators, optimizers and mappers it is also possible to use a more generic declaration by providing a ObjDef object instead of a specific implementation. The ObjDef object contains of *type* field which is a string with class name that should be used and *params* field which is a map of parameters for the object. When a method (addFilter(), build(), optimize() or mapTargets()) is invoked with ObjDef object it initializes the object of *type* class and tries to configure it using parameters from map by invoking proper setters.

For instance ObjDef object:

{type: “pl.cccc”, params:{expGamma:10,expCost:4}}

Is equivalent to the code:

```
Calibrator c = new pl.cccc();
c.setExpGamma(10);
c.setExpCost(4);
```

Methods build(), optimize(), mapTargets(), addFilter(), useFilter() may be invoked with proper objects (implementing Calibrator, Optimizer or Mapper or CalFilter) or with ObjDef object.

Asynchronicity

As build(), optimize() and mapTargets() methods may take some time, all these methods have both synchronous and asynchronous implementations. For instance, when buildAsync(params,callback) is executed, it starts a background process calculating a model and ends immediately returning a Future<Void> object. User may check if the model is ready by checking Future.isDone() method or by referencing directly to EtCal object’s method isCalibrating(). If an object implementing Callback interface (with ready() method) is supplied by the second parameter of buildAsync(), a ready() method of that object will be executed when model is ready.

Similarly, there are optimizeAsync() and isOptimizing() methods for optimization and mapTargetsAsync() and isMapping() methods for mapping targets.

Useful classes and interfaces

To properly use ETCAL object and interface described above it is necessary to use predefined classes.

DataUnits - data to calibrate

DataUnits object contains a collection of DataUnit objects. Every DataUnit object contains a list of features (double values) and a list of Target objects (each containing X and Y coordinates)

Additionally DataUnits object contains helper methods to load() object from BufferedReader or file in JSON format and to save() the object in JSON format. (GSON library is used for JSON manipulations)

An example of DataUnit object in JSON format:

{"variables":[20.0,13.0,10.0,20.0],"targets":[{"x":260.0,"y":260.0,"w":1.0},{"x":360.0,"y":360.0,"w":1.0}]}

Target – the result

The object representing single point's coordinates X,Y.

Errors – error report

Simple object that contains the following fields: rSquareX, rSquareY, MSEX, MSEY, absError, absErrorX, absErrorY

ObjDef – definition of the object

ObjDef is a definition how to build any other object. It contains two fields:

- String type – name of a class to be used
- Map<String,String> params – a map of parameters. Each parameter should have a corresponding setter method in the object of class type (e.g. if we have *mask* parameter ***setMask()*** method is invoked on the object)

Generally ObjDef may be used for any kind of object but in ETCAL library the pattern is used to create CalFilter, Calibrator, Optimizer and Mapper classes.

Internal objects

Filter

A new filter must implement Filter interface and implement ***filter(DataUnits data)*** method that returns DataUnits converted by the filter.

Calibration

A new calibrator must implement ***Calibrator*** interface and implement two methods: ***calculate()*** that calculates a regression model using the given data and ***get (double[] x)*** that calculates Target value for the given array of features.

When the calibrator is intended to use any regression classifier available in WEKA library it is recommended to extend ***CalibratorWeka*** class and create the classifier in the constructor (see class ***CalibratorWekaLinearRegression*** as a good example).

Optimizer

A new optimizer should implement ***Optimizer*** interface implement ***optimize(DataUnits)*** method. The method should create calibrator object return it in ***getBest()*** method. Instead of building the whole class from a scratch it is easier to extend ***AbstractOptimizer*** class that contains all useful parameters.

Mapper

A new mapper should implement Mapper interface and implement ***map(DataUnits)*** method. The method returns a List<Integer> with indexes of the chosen targets.

Web interface

Every ***ETCal*** object method has its web extension in ***pl.kasprowski.etcal.www*** package. Each servlet gets parameters in JSON format as DataUnits or ObjDef objects and executes the desired method. ETCal object is kept in user's session so it is possible to use the application by many users in the same time.

The simple single site application that uses ETCAL is developed in index.jsp file. The buttons use AJAX requests to perform specific actions on the server site.

Additional questions: pawel@kasprowski.pl