Framework Specification: Internals and Externals

February 13, 2013

Contents

Abstract

This is a mini-report for global architecture of MixComp. We will make use of two terms "framework" and "developer". The framework provides a unified development environment Whereas developer will refer to the coding part that will make use of the framework (without modifying the framework) to realize new mixture laws. In practice, this is just a separation of Interface (framework) and Implementation (developer).

1 Framework Introduction

The idea of unified framework is to integrate the existing and future clustering models. The things that can be expected from the framework are:

1. Unified environment for composite mixture model based on the independence assumption.
2. Abstract(Interface) plug-in class that must be derived by developer to develop a new mixture law and data handling.
3. Facilitate creation of Rpackage, Web Interface, GUI and other High level functionalities.
4. Facilitate parallelization using distributed and shared memory models.
5. Take input from user and run the whole software.

The developer is not expected to change the functionalities provided by this framework but can only provide concrete behavior to these functionalities. Any new functionalities should be first introduced in the framework. Hence the framework is expected to evolve with time. The developer is only expected to provide all the low level implementations needed by the framework without worrying about how these functionalities will be brought together to realize a composite mixture model. Hence in short, the developer can concentrate on development of existing and new mixture laws without worrying about how to run them in integrated(composite) environment. The developer is free to chose it's development environment (including numerical libraries for example STK++, Eigen, Lapack or anything more suitable to developer needs) and in no way will be restricted by framework. For example, a developer can re-factor their existing codes and fit them into the framework (Quentin existing codes will be a good test for it) or one can re-implement from scratch using this framework (development of simple models including Bernoulli and Gaussian(with diagonal co-variance matrix) with Serge on STK++ platform will be a good test for it). This is the most interesting feature of this architecture as it will allow to independently develop new models.

2 Plug-in Specification

Below are the Interface methods that will (hopefully) allow to develop all the kinds of existing and new mixture laws that are based on SEM-gibbs algorithm. The functions ending with "() = 0" must be defined by the developer of new mixture law. The functions ending with "() {}" do nothing by default and the functions ending with "()" will be implemented in framework (the developer can of-course overwrite them in some cases for performance reasons.)

- **virtual void initializeStep(double**) = 0**
  This function must be use for initialization stuffs including initialize of all the parameters. The function accepts one argument for randomly initialized class labels. This method will be called only once in the very beginning.
- **virtual void imputationStep(double*) {}**
  This function should be used for Imputation of data. The function accepts one argument for proportions.
- **virtual void samplingStep(double*,double**) = 0**
  This function must be used for simulation of all the latent variables and/or missing data excluding class labels. The class labels will be simulated by the framework itself because to do so we have to take into account all the mixture laws. The function accepts two arguments, one for proportions and one for class labels (or conditional probabilities).
- **virtual void paramUpdateStep(double**) = 0**
  This function is equivalent to Mstep. The function accepts one argument for class labels (or conditional probabilities). This function must be defined by developer to update parameters.
- **virtual void finalizeStep() {}**
  This step can be used by developer to finalize any thing. It will be called only once after we finish running the SEM-gibbs algorithm.
- **virtual double posteriorProbability(int sample_num,int Cluster_num) = 0**
  This function must be defined by developer to return the probability for corresponding sample and cluster.
- **virtual double** allPosteriorProbabilties()**
  This function will be defined in framework using **posteriorProbability** function, but developer can override this function for performance reasons.
- **virtual double logLikelihood() const = 0**
  This will return the likelihood value to be used by selection criteria.
- **virtual int freeParameters() const = 0**
  This will be used to return number of free parameters to be used by selection criteria.

I have intentionally excluded the notion of Estep, Mstep, Sstep and Cstep to avoid any misunderstandings for the developer. These names will be used inside the framework to make it generic enough to expand for future.

3 stk++ Statistical Models

A (multivariate) statistical model in stk++ is a

A Coding conventions

For the coding we use the java coding convention. All data members are defined with an underscore at the end of their name, like `data_`.

Data members referring to pointer start with `p_`

All alternatives should be enclosed in `enum`.

The notations proposed in this document can be modified if needed or enclosed in namespace in order to avoid name collision.

B Components behaviors

B.1 Algorithms

The algorithms will be:

```
enum Algo
{
  em_,
  cem_,
  sem_,
  mcem_,
  exact_
}
```

The priority is to implement the `*em` versions, the `exact_` is just there in case of, but should not be implemented. Should we add stochastic minimization algorithms ?

B.2 Stopping criteria

The stopping criteria will be:

```
enum StopCriteria
{
  deltaLnLikelihood_,
  deltaPostProbabilities_,
  deltaParameters_,
  nbIterMax_
}
```

It should be possible to mix different criteria, for example `deltaLnLikelihood_|nbIterMax_` mean that we want to stop the iteration when one of the criteria is true.

B.3 Data initialization

The initialization of the algorithm will be:

```
enum Initialization
{
  randomPartition_,
  randomParameters_,
  givenPartition_,
  givenPosteriorProbabiblity_,
```

```
  givenParameters_
}
```

Did i forget a method for initialization ? The random cases are the priority. In case of heterogeneous distributions it will be difficult to find a way for initialization with given parameters.

B.4 Strategies

A strategy is composed of three parts:

1. multiple initializations,
2. for each initialization a short run,
3. A long run.

B.4.1 short runs

A short run will be an arbitrary number of sequence `(Algo_, StopCriteria)`. For example:

```
{(sem_, 1000), (cem_, 0.01|1000)}, {(em, 0.01)}
```

mean that in a short run, there is 1000 iterations of the SEM algorithm, and at most 1000 iterations of CEM that will be stopped if some other criterion have a delta less than 0.01 and finally iterations of the EM until the delta of some criteria is less than 0.01.

B.4.2 long run

A long run is initialized with the better of the short run and a StopCriteria. If there is no short run, with an initialization.

B.5 Model Selection Criteria

The model criteria will be:

```
enum Criteria
{
  aic_,
  bic_,
  icl_,
  cv_,
  penExtern_
}
```

It should be possible to let an user to define its own penalization criteria (`penExtern_` option). Should we add cross-validation ?