

BNPlib for density estimation:

A nonparametric C++ library
(part 2)

Bruno Guindani
Elena Zazzetti

January 8th, 2020



POLITECNICO
MILANO 1863

<https://github.com/poliprojects/BNPlib>

Algorithm <Hierarchy<Hypers>.Mixture>

```
unsigned int maxiter = 20000;  
unsigned int burnin = 5000;  
int num_clusters;  
  
MatrixXd data;  
vector<unsigned int> allocations;  
vector<Hierarchy<Hypers>> unique_values;
```

Neal2 <Hierarchy<Hypers>.Mixture>

```
void initialize() override;  
void sample_allocations() override;  
void sample_unique_values() override;
```

Neal8 <Hierarchy<Hypers>.Mixture>

```
unsigned int n_aux = 3;  
vector<Hierarchy<Hypers>> aux_unique_values;  
void sample_allocations() override;
```

Hierarchies

HierarchyNNIG <Hypers>

```
vector<MatrixXd> state;
shared_ptr<Hypers> hypers;
```

→ state(2, MatrixXd(1,1))

```
VectorXd eval_marg(MatrixXd datum);
VectorXd like(MatrixXd datum);
void sample_given_data(MatrixXd data);
void draw();
vector<double> normal_gamma_update(VectorXd data,
    double mu0, double alpha0, double beta0, double lambda0);
```

→ check in Algorithm Constructor if data is univariate

HierarchyDummy <Hypers>

```
vector<MatrixXd> state;
shared_ptr<Hypers> hypers;
```

```
VectorXd eval_marg(MatrixXd datum);
VectorXd like(MatrixXd datum);
void sample_given_data(MatrixXd data);
void draw();
std::vector<Eigen::MatrixXd> dummy_update(MatrixXd data,
    VectorXd mu0, MatrixXd lambda0);
```

Hypers

HypersFixedNNIG

```
double mu0, lambda, alpha0, beta0;
```

HypersDummy

```
VectorXd mu0;  
MatrixXd lambda0;
```

Mixtures

DirichletMixture

```
double totalmass;
```

```
double const prob_existing_cluster( int card, unsigned int n)  
double const prob_new_cluster( unsigned int n, unsigned int n_unique)
```

PitYorMixture

```
double strength;  
double discount;
```

```
double const prob_existing_cluster( int card, unsigned int n)  
double const prob_new_cluster( unsigned int n, unsigned int n_unique)
```

Algorithms

```
class Factory{
private:
    std::map<std::string, AlgoBuilderType> storage;
public:
    static Factory& Instance();
    void add_builder(const std::string &name, const AlgoBuilderType &b);
    auto create_algorithm(const std::string &name) const {};
    // [...]

factories
input
```

Multivariate Proto

```
message Par_Col {
    repeated double elems = 1;
}

message Param {
    repeated Par_Col par_cols= 1;
}

message UniqueValues {
    repeated Param params= 1;
}

message IterationOutput {
    repeated int32 allocations = 1;
    repeated UniqueValues uniquevalues = 2;
}

message ChainOutput {
    repeated IterationOutput chain = 1;
}
```

Collectors

