Class Documentation for Inference Algorithms

Contents

Chapter 1

Overview

All models have been implemented using an object-oriented approach, with a shared set of core methods across all models. The common methods include:

- `fit()` : This method executes the training process and updates the model's learned parameters. It takes as input a `NumPy` array of shape $(N \times D)$, where N is the number of data points and D is the number of features. Additional optional arguments specific to each model may also be included and are detailed below.
- `predict()` : This method accepts a `NumPy` array of shape $(N \times D)$ as input and returns an array of component assignments corresponding to the input data points. Each model may include additional optional arguments, which are described in the following sections.
- `impute()` : This method takes an input data matrix with missing values and returns a new matrix where the missing values have been imputed. The imputation process is model-specific and can handle different approaches based on the model used.

In the documentation for each specific model below, we only include user-facing methods and have omitted internal methods.

The models in this documentation are as follows:

1. `GMMGibbs` : Gibbs sampler for GMM
2. `BMMGibbs` : Gibbs sampler for BMM
3. `GMMVBEM` : VBEM for GMM
4. `BMMVBEM` : VBEM for BMM
5. `GMMEM` : EM for GMM
6. `BMMEM` : EM for BMM

This also includes hyper-parameter classes used to instantiate prior distribution hyper-parameter values for each inference class

Chapter 2

Prior Parameter Classes

2.1 `GMMPriorParameters` Class

2.1.1 Description

`GMMPriorParameters` defines the prior parameters for a Gaussian Mixture Model (GMM). These parameters include the prior distributions for the mixture components, component means, covariances, and other hyperparameters required for fitting the model.

2.1.2 Constructor

```
def init(self, data, K):
```

Parameters:

- `data`: The data matrix (ND), where N is the number of data points and D is the number of features.
- `K`: The number of components in the Gaussian Mixture Model (GMM).

Attributes:

- `K`: The number of components in the mixture model.
- $\boldsymbol{\alpha}$`_0`: The Dirichlet prior for the component weights (K).
- `m_0`: The prior means for the components ($K \times D$).
- `S_0`: The prior covariance matrices for the components ($K \times D \times D$).
- `k_0`: The prior scaling factor for the component means (K).
- $\boldsymbol{\nu}$`_0`: The prior degrees of freedom for the component covariances (K).
- `o`, `h`: Hyperparameters for MNAR data handling using the Beta distribution ($K \times D$).

2.2 `BMMPriorParameters` Class

2.2.1 Description

`BMMPriorParameters` defines the prior parameters for a Bernoulli Mixture Model (BMM). Similar to the `GMMPriorParameters` class, this class defines priors for the mixture components, component-specific parameters, and missing data handling.

2.2.2 Constructor

`def init(self, data, K):`

Parameters:

- `data`: The data matrix (ND), where N is the number of data points and D is the number of features.
- `K`: The number of components in the Bernoulli Mixture Model (BMM).

Attributes:

- `K`: The number of components in the mixture model.
- $\boldsymbol{\alpha}$`_0`: The Dirichlet prior for the component weights (K).
- `a_0`: The prior alpha parameters for the Bernoulli distribution $(K \times D)$.
- `b_0`: The prior beta parameters for the Bernoulli distribution $(K \times D)$.
- `o`, `h`: Hyperparameters for missing data handling using the Beta distribution $(K \times D)$.

Chapter 3

Gibbs Sampler

3.1 `GibbsModel` Class

3.1.1 Description

`GibbsModel` is an abstract base class for `BMMGibbs` and `GMMGibbs` models. Thus the methods listed below are shared among the two BMM and GMM specific Gibbs sampling classes.

3.1.2 Methods

fit

```
@abstractmethod
def fit(self, X, num_iters=6000, burn=2000, mnar=False, collapse=False):
```

This method must be implemented by subclasses to perform the Gibbs sampling algorithm. It fits the model to the data and stores the posterior samples.

Parameters:

- `X`: The input data matrix, possibly containing missing values.
- `num_iters`: The number of iterations to run the Gibbs sampler.
- `burn`: The number of burn-in iterations to discard.
- `mnar`: Boolean indicating whether to model MNAR (Missing Not At Random) missingness.
- `collapse`: Boolean indicating whether to collapse the model.

get_map_params

```
def get_map_params(self):
```

Returns the parameters corresponding to the maximum a posteriori (MAP) estimate.

get_summarizing_results

```
def get_summarizing_results(self, y):
```

Computes the average and standard deviation for metrics such as Adjusted Rand Index (ARI), log-likelihood, and posterior likelihood. **Parameters:**

- `y`: ground truth labels corresponding to sample used to fit model

get_aligned_param_means

```
def get_aligned_param_means(self):
```

Returns the mean of the aligned parameters across all samples.

3.2 BMMGibbs Class

3.2.1 Description

`BMMGibbs` implements a Bernoulli Mixture Model (BMM) with missing data using Gibbs Sampling. The class is derived from `GibbsModel` and includes specific methods for handling binary data and missing values.

3.2.2 Constructor

```
def __init__(self, priorParameters: BMMPriorParameters):
```

Parameters:

- `priorParameters`: An instance of the prior distribution for the Bernoulli mixture model.

3.2.3 Methods

log_likelihood

```
def log_likelihood(self, X_new, mnar=False):
```

Computes the log-likelihood of given input data matrix using the current model parameters. **Parameters:**

- `X_new`: data to compute likelihood using current parameters
- `mnar` : boolean indicating whether to use MNAR approach

Returns:

- The total log-likelihood as a scalar

fit

```
def fit(self, X, num_iters=6000, burn=2000, mnar=False, collapse=False):
```

Performs the Gibbs sampling for the Bernoulli mixture model, including missing data imputation and parameter updates.

Parameters:

- `X`: data to fit model on
- `num_iters` : number of iterations to run for Gibbs
- `burn` : number of samples to burn
- `mnar` : boolean indicating whether to use MNAR approach
- `collapse` : boolean indicating whether to use collapsed approach

impute

```
def impute(self, X_new, sample=None, eps=1e-14):
```

Imputes missing values in the data using the fitted model parameters.

Parameters:

- `X_new`: the sample to impute
- `sample` : allows specifying what Gibbs sample to use

Returns: the input data matrix with missing entries imputed

predict

```
def predict(self, X_new):
```

Uses fitted parameters to predict component assignments for a given input

Parameters:

- `X_new`: The input data matrix with or without missing values.
- `mnar` : boolean indicating whether to use MNAR approach

Returns:

- Numpy array of component assignments $(N,)$

3.3 GMMGibbs Class

3.3.1 Description

`GMMGibbs` implements a Gaussian Mixture Model (GMM) with missing data using Gibbs Sampling. This class is derived from `GibbsModel` and uses multivariate normal distributions for modeling continuous data.

3.3.2 Constructor

```
def __init__(self, priorParameters: GMMPriorParameters):
```

Parameters:

- `priorParameters`: An instance of the prior distribution for the Gaussian mixture model.

3.3.3 Methods

log_likelihood

```
def log_likelihood(self, X_new, mnar=False):
```

Computes the log-likelihood of given input data matrix using the current model parameters. **Parameters:**

- `X_new`: data to compute likelihood using current parameters

- `mnar` : boolean indicating whether to use MNAR approach

Returns: The total log-likelihood as a scalar

fit

```
def fit(self, X, num_iters=6000, burn=2000, mnar=False, collapse=False):
```

Performs Gibbs sampling for the Gaussian mixture model, including parameter updates and missing data imputation.

Parameters:

- `mnar`: boolean indicating whether to use MNAR generative approach
- `collapse` : boolean indicator whether to use collapsed Gibbs

impute

```
def impute(self, X_new, sample=None, eps=1e-14):
```

Imputes missing values in the data using the Gaussian mixture model parameters.

Parameters:

- `X_new`: the sample to impute
- `sample` : allows specifying what Gibbs sample to use

Returns: the input data matrix with missing entries imputed

predict

```
def predict(self, X_new):
```

Uses fitted parameters to predict component assignments for a given input

Parameters:

- `X_new`: The input data matrix with or without missing values.
- `mnar` : boolean indicating whether to use MNAR approach

Returns:

- Numpy array of component assignments $(N,)$

Chapter 4

Variational Bayesian EM Algorithm (VBEM)

4.1 `BMMVBEM` Class

4.1.1 Description

`BMMVBEM` implements a Bernoulli Mixture Model (BMM) with missing data using VBEM. It is derived from the `VBEMModel` abstract class and includes specific methods for handling binary data with missing values using variational inference.

4.1.2 Constructor

```
def init(self, priorParameters : BMMPriorParameters):
```

Parameters:

- `priorParameters`: An instance of the prior distribution for the Bernoulli mixture model.

4.1.3 Methods

fit

```
def fit(self, X, mode=0, max_iters=200, tol=1e-4):
```

Performs the VBEM algorithm for the Bernoulli mixture model. The algorithm iteratively updates the parameters until convergence. **Parameters:**

- `X`: The input data matrix (N x D).
- `mode`: Determines the update mode for π. Mode 0 uses variational updates, and mode 1 uses MLE-style updates.
- `max_iters`: The maximum number of iterations to run.
- `tol`: The convergence tolerance for the ELBO.

impute

```
def impute(self, X_new, eps=1e-14):
```

Imputes missing values in the data using the fitted model parameters.

Parameters:

- `X_new`: The input data matrix with missing values.
- `eps`: A small constant to prevent division by zero.

Returns: the input data matrix with missing entries imputed

predict

```
def predict(self, X_new):
```

Uses fitted parameters to predict component assignments for a given input

Parameters:

- `X_new`: The input data matrix with or without missing values.

Returns:

- Numpy array of component assignments $(N,)$

4.2 `GMMVBEM` Class

4.2.1 Description

`GMMVBEM` implements a Gaussian Mixture Model (GMM) with missing data using VBEM. It is derived from the `VBEMModel` abstract class and uses multivariate normal distributions to model continuous data.

4.2.2 Constructor

```
def init(self, priorParameters : GMMPriorParameters):
```

Parameters:

- `priorParameters`: An instance of the prior distribution for the Gaussian mixture model.

4.2.3 Methods

fit

```
def fit(self, X, max_iters=200, tol=1e-4):
```

Performs the VBEM algorithm for the Gaussian mixture model

Parameters:

- `X`: The input data matrix (N x D).
- `max_iters`: The maximum number of iterations to run.
- `tol`: The convergence tolerance for the ELBO.

impute

```
def impute(self, X_new, eps=1e-14):
```

Imputes missing values in the data using the learned variational posteriors.

Parameters:

- `X_new`: The input data matrix with missing values.
- `eps`: A small constant to prevent division by zero.

Returns: the input data matrix with missing entries imputed

predict

```
def predict(self, X_new):
```

Uses fitted parameters to predict component assignments for a given input

Parameters:

- `X_new`: The input data matrix with or without missing values.

Returns:

- Numpy array of component assignments $(N,)$

Chapter 5

Expectation Maximization Classes

5.1 BMMEM Class

5.1.1 Description

`BMMEM` implements the Expectation-Maximization (EM) algorithm for Bernoulli Mixture Models (BMM).

5.1.2 Constructor

```
def init(self, K, complete_case=False):
```

Parameters:

- `K`: The number of components in the mixture model.

Attributes:

- `rng`: Random number generator for reproducibility.
- `K`: The number of components in the mixture model.
- `fitted`: A boolean indicating if the model has been fitted.
- `complete_case`: Whether to use complete case analysis for missing data.

5.1.3 Methods

fit

```
def fit(self, X, max_iters=200, tol=1e-4):
```

The `fit` method runs the EM algorithm to fit the BMM to the data. It iterates between the E-step and M-step until convergence, based on the provided tolerance. It also handles missing data using the `complete_case` flag.

Parameters:

- `X`: The input data matrix (N x D), where N is the number of data points and D is the number of features.
- `max_iters`: The maximum number of iterations to run the EM algorithm.
- `tol`: The convergence tolerance for the log-likelihood change between iterations.

Returns:

- A dictionary containing the following keys:
  - `z`: Cluster assignments for each data point (N).
  - $\boldsymbol{\pi}$: Mixture component weights (K).
  - $\boldsymbol{\theta}$: Component parameters (K x D).
  - `loglike`: Log-likelihood values over iterations.

compute_responsibility

```
def compute_responsibility(self, X_new, eps=1e-14):
```

Computes the responsibility matrix R for a new set of data points, which indicates the probability that each data point belongs to each component of the mixture model. This method handles missing data according to the `complete_case` argument.

Parameters:

- `X_new`: The input data matrix (N x D) with missing values (represented as NaN).
- `eps`: A small constant to prevent numerical issues.

Returns:

- `R`: The responsibility matrix (N x K), with each row summing to 1.
- `loglike`: The log-likelihood of the data given the current model parameters.

impute

```
def impute(self, X_new, eps=1e-14):
```

Imputes missing values in the data using the fitted BMM parameters. The imputation is done based on the responsibility matrix R and the model parameters $\boldsymbol{\theta}$ for each mixture component.

Parameters:

- `X_new`: The input data matrix (N x D) with missing values (NaN).
- `eps`: A small constant to prevent numerical issues during the imputation process.

Returns:

- `X_filled`: The imputed data matrix (N x D), with missing values replaced by imputed values.

log_likelihood

```
def log_likelihood(self, X_new):
```

Computes the log-likelihood of a new set of data points under the current model parameters. This is used for model evaluation and convergence checking.

Parameters:

- `X_new`: The input data matrix (N x D).

Returns:

- `loglike`: The log-likelihood of the data under the current model.

predict

```
def predict(self, X_new):
```

Predicts the cluster assignments for a new set of data points.

Parameters:

- `X_new`: The input data matrix (N x D) with missing values.

Returns:

- A vector of cluster assignments (N), indicating the predicted mixture component for each data point.

5.2 `GMMEM` Class

5.2.1 Description

`GMMEM` implements the Expectation-Maximization (EM) algorithm for Gaussian Mixture Models (GMM) with missing data.

5.2.2 Constructor

```
def init(self, K, complete_case=False):
```

Parameters:

- `K`: The number of components in the mixture model.
- `complete_case`: Boolean indicating whether to perform complete case analysis (i.e., ignore incomplete data points).

Attributes:

- `rng`: Random number generator for reproducibility.
- `K`: The number of components in the mixture model.
- `fitted`: A boolean indicating if the model has been fitted.
- `complete_case`: Whether to use complete case analysis for missing data.

5.2.3 Methods

fit

```
def fit(self, X, max_iters=200, tol=1e-4):
```

The `fit` method runs the EM algorithm to fit the GMM to the data. It iterates between the E-step and M-step until convergence, based on the provided tolerance. It also handles missing data using the `complete_case` flag.

Parameters:

- `X`: The input data matrix (N x D), where N is the number of data points and D is the number of features.
- `max_iters`: The maximum number of iterations to run the EM algorithm.

- `tol`: The convergence tolerance for the log-likelihood change between iterations.

Returns:

- A dictionary containing the following keys:
  - `z`: Cluster assignments for each data point (N).
  - $\boldsymbol{\pi}$: Mixture component weights (K).
  - $\boldsymbol{\mu}$: Component means (K x D).
  - $\boldsymbol{\Sigma}$: Component covariances (K x D x D).
  - `loglike`: Log-likelihood values over iterations.

compute_responsibility

```
def compute_responsibility(self, X_new, eps=1e-10):
```

Computes the responsibility matrix R for a new set of data points. This method handles missing data according to the `complete_case` argument.

Parameters:

- `X_new`: The input data matrix (N x D) with missing values (represented as NaN).
- `eps`: A small constant to prevent numerical issues.

Returns:

- `R`: The responsibility matrix (N x K), with each row summing to 1.
- `loglike`: The log-likelihood of the data given the current model parameters.

impute

```
def impute(self, X_new, eps=1e-14):
```

Imputes missing values in the data using the fitted GMM parameters. The imputation is done based on the responsibility matrix R and the model parameters.

Parameters:

- `X_new`: The input data matrix (N x D) with missing values (NaN).
- `eps`: A small constant to prevent numerical issues during the imputation process.

Returns:

- `X_filled`: The imputed data matrix (N x D), with missing values replaced by imputed values.

log_likelihood

```
def log_likelihood(self, X_new):
```

Computes the log-likelihood of a new set of data points under the current model parameters. This is used for model evaluation and convergence checking.

Parameters:

- `X_new`: The input data matrix (N x D).

Returns:

- `loglike`: The log-likelihood of the data under the current model.

predict

```
def predict(self, X_new):
```

Predicts the cluster assignments for a new set of data points based on the current responsibility matrix.

Parameters:

- `X_new`: The input data matrix (N x D) with missing values.

Returns:

- A vector of cluster assignments (N), indicating the predicted mixture component for each data point.