MultipleDistributionFitting Documentation

Release 0.1.1

Shirui

Apr 26, 2018

Contents

Finding optimized number of components from mixed distribution data.

Process:

1. Define target function(s)
2. Create fitting model(s)
3. Evaluation the model by AIC, AICc, BIC
4. Choose the model that minimizes the BIC, AICc or AIC

CHAPTER 1

Installation

1.1 Using pip

```
pip install MultipleDistributionFitting
```

1.2 Directly download

```
git clone https://github.com/Shirui816/MultipleDistributionFitting.git
```

or

```
wget https://github.com/Shirui816/MultipleDistributionFitting/archive/master.zip
```

1.3 Requirements

```
numpy >= 1.14
scipy >= 1.0
python3
pandas >= 0.21
```

Anaconda enviroment is recommanded.

CHAPTER 2

Theory and Usage

2.1 Theory

Our goal is to find optimized number of components in a mixture model. Assuming that a mixture of distributions are given as:

$$f(x) = \sum_i^n a_i g_i(x; \theta_i)$$

That each $g_i(x; \theta_i)$ is a distribution function with weight a_i, θ_i is the parameter vector. Usually, a mixture model data set can be fitted by arbitary number of components n, to supress overfitting, Akaike information criterion (AIC), Bayesian information criterion (BIC) and a modified AIC (AICc) is used for small sized samples to estimate the model and find out the most probable number of components n.

2.2 Usage

2.2.1 Generate mixture models for fitting

```
from utils import n_func_mix, n_func_maker
```

Make a n-fucntion mixture from a common base

```
def n_func_maker(func: callable, n: int, known: list) -> callable:
    r"""Make n-function mixture from a common base.

    Arguments:
    func: base function, the signature must start with `x`.
    n: desired number of components.
    known: a list of $n\times n_{\text{func args}}$ variables.
```

```
            None is for fitting variables and values for fixed variables.

    Returns:
    mixture: callable.
    """
```

For example, suppose that a 2-component mixture is generated by the base fuction `f(x, a, b, c)` that the `a` variable of 2nd function is equal to 2, the `n_func_maker(f, 2, known=[None, None, None, 2, None, None])` generates a mixed function with signatures `x, a0, b0, c0, b1, c1`.

Make mixture of functions

```
def n_func_mix(funcs: list of callables) -> callable:
    r"""Mixer for defining functions mixed by base function.

    For scipy.optimize.curv_fit.

    Arguments:
    funcs: A list of callables, and signatures of
           all functions must begin with `x`.

    Returns: Function that mixed n base functions.
    """
```

2.2.2 Fitting the generated models

```
from utils import FitLSQ

class FitLSQ():
    def __init__(self, func: callable):

    def set_bounds(self, bounds: list, known: list) -> self:
        r"""Set bounds for target function.

        Arguments:
        bounds: 2d-list for lower and upper bounds (lb, ub) for arguments
                of base function. +/-np.inf for no bounds.
        n: number of parameters ofl BASE functions.
        known: Known parts in functions.
        Returns:
        self
        """

    def set_p0(self, p0: list, known: list) -> self:
        r"""Set initial values for fitting.

        Arguments:
        p0: tuple or list for initial parameters.
        known: list for known components.

        Returns:
        self
        """
```

```
    def fit(self, x: np.ndarray, y: np.ndarray, **kwargs) -> self:
        r"""Fit the model.

        Arguments:
        x: np.array for x
        y: np.array for y

        Keyword Arguments:
        kwargs that fits scipy.optimize.curve_fit

        Returns:
        self
        """
```

For example, `model.set_p0([0.1, 0.002, 3.7])` and `model.set_bound([[0, -np.inf, 1], [1,np.inf, 2]])` are for a mixture of consists of 3-argument base functions with initial guess of (0.1, 0.002, 3.7) for parameters and corresponding bounds are (0, 1), (-inf, inf) and (1, 2).

Warning: `set_p0` and `set_bounds` are currently supported for the compoents in the mixture have same base function only.

2.2.3 Evaluate models.

```
from utils import Evaluation

class Evaluation():
    def __init__(self, model: FitLSQ):
        r"""Initialize with model.

        Arguments:
        model: a fit object
        """

    def aic(self, x: np.ndarray) -> np.ndarray:
        r"""Calculate AIC.

        Aho, K.; Derryberry, D.; Peterson, T. (2014), "Model selection for
        ecologists: the worldviews of AIC and BIC", Ecology, 95: 631-636,
        doi:10.1890/13-1452.1.

        AIC = 2k - 2\ln{\hat{\mathcal{L}}}, \hat{\mathcal{{L}}} is Likelihood.

        Arguments:
        samples: samples of (n_samples, n_features)

        Returns:
        aic: np.ndarray
        """

    def bic(self, x: np.ndarray) -> np.ndarray:
        r"""Calculate BIC.

        Schwarz, Gideon E. (1978), "Estimating the dimension of a model",
        Annals of Statistics, 6 (2): 461-464, doi:10.1214/aos/1176344136,
        MR 0468014.
```

```
        BIC = \ln{N}k - 2\ln{\hat{\mathcal{L}}}

        Arguments:
        samples: samples of (n_samples, n_features)

        Returns:
        bic: np.ndarray
        """


    def aicc(self, x: np.ndarray) -> np.ndarray:
        r"""Calculate AICc.

        deLeeuw, J. (1992), "Introduction to Akaike (1973) information theory
        and an extension of the maximum likelihood principle" (PDF),
        in Kotz, S.; Johnson, N.L., Breakthroughs in Statistics I, Springer,
        pp. 599-609.

        AICc = AIC + \frac{2k^2+2k}{N-k-1}

        Arguments:
        samples: samples of (n_samples, n_features)

        Returns:
        aicc: np.ndarray
        """

    @classmethod
    def make_sample(cls, n, x, pdf):
        r"""Make random sample taken from x.

        Arguments:
        n: int, sample size
        x: np.ndarray
        pdf: np.ndarray

        Returns:
        sample
        """
```

`x` is a sample data set with shape of (n_samples, n_features), samples can be generated by `Evaluation.make_sample` from the fitting data `x` and `y` if the fitting object is the pdf function.

CHAPTER 3

Lorentzian mixtures for H^1NMR

```
In [8]: from Lorentzian import NMRFitting
```

```
class NMRFitting(object):
    r"""Fitting NMR datas."""

    def __init__(self, files, components_range,
                 n_mc_trials=10, n_samples=3000, shift=0, tol=0.01):
        r"""Initialize.

        Arguments:
        files: a list of files of NMR datas
        components_range: a touple of the range of how many peaks
        n_mc_trials: default is 10. times that finding BIC
        n_samples: default is 3000. samples used to find BIC
        shift: default is 0. Set shift if you want to remove some components.
        tol: Tolerance of ratio of negative areas after shift.
        """

    def set_p0_bounds(self, p0=(0.5, 0.002, 3.7),
                      bounds=((0, 1e-4, 3.5), (1, 1e-1, 4.1))):
        r"""Set p0 and bounds, defaults are for PEG.

        Arguments:
        p0: 1-d touple or list for area, peak_width and chemical shift
        bounds: 2-d touple or list for the lower/upper value of area,
                peak_width and chemical shift. +/-np.inf for no bounds.

        Returns:
        self
        """

    def fitting(self, **kwargs):
        r"""Fitting method.

        kwargs: for `scipy.optimize.curv_fit`
```

```
    """
```

```
In [31]: a_0_5 = NMRFitting(["../data/A-0.50-fitting.txt"], (2,7))
```

```
In [32]: a_0_5.set_p0_bounds(p0=[0.5, 0.002, 3.7], bounds=[[0, 1e-4, 3.5], [1, 1e-1, 4.1]])
         a_0_5 = a_0_5.fitting()
```

```
../utils/FitLSQ.py:77: UserWarning: p0 must EXACTLY match the base function!
  UserWarning)
../utils/FitLSQ.py:50: UserWarning: Bounds must EXACTLY match the base function!
  UserWarning)
```

```
Best estamation by AIC is 3
The parameters are: [0.233335 0.001504 3.701074 0.384682 0.034882 3.699284 0.418008 0.002058
 3.698922]
Best estamation by AICc is 3
The parameters are: [0.233335 0.001504 3.701074 0.384682 0.034882 3.699284 0.418008 0.002058
 3.698922]
Best estamation by BIC is 3
The parameters are: [0.233335 0.001504 3.701074 0.384682 0.034882 3.699284 0.418008 0.002058
 3.698922]
The normalization factor is 1.0012, the original is 2.7394
```

```
In [34]: %matplotlib inline
         from pylab import *
         fig = figure(figsize=(12,4))
         ax1 = fig.add_subplot(131)
         ax2 = fig.add_subplot(132)
         ax3 = fig.add_subplot(133)
         ax1.plot(range(2,7), a_0_5[0][0], label='AIC', lw=2)
         ax1.legend()
         ax2.plot(range(2,7), a_0_5[0][1], label='AICc', lw=2)
         ax2.legend()
         ax3.plot(range(2,7), a_0_5[0][2], label='BIC', lw=2)
         ax3.legend()
```

```
Out[34]: <matplotlib.legend.Legend at 0x7fc76216fc18>
```