# Manual

## Brief Introduction

This document is a guide for users to get touch with our tool ‘autoBioSeqpy’ for modeling and analysis of Protein, DNA and RNA data, including a brief introduction to installation, quick start and standalone modules (a jupyter notebook example provided).

Our tool autoBioSeqpy’ is a self-made python tool which can transfer the sequence into matrix, and then use it for deep learning. In this document, users can find the data format, the internal mechanism of encoding and figure out how to use this tool.

## Installation

All the code of autoBioSeqpy’ is wrote in Python, and no mixture code (e.g. C/C++) is used in this project, so the installation is very easy. Once the dependencies are resolved, the only thing to do is to make the path as a working path or put the code into the search path.

### 2.1 Dependence

Some python modules are necessary for autoBioSeqpy, which are **re, numpy, importlib, sklearn** and **keras**. Since all modules are included in anaconda3, users could resolve module dependencies by installing anaconda3 (2 is not suggested) on their official website <https://www.anaconda.com/>. Alternatively, the users can install the module manually, for example using pip or another installer. If using pip to install the dependent modules, the command is:

**pip install numpy**

or

**pip install numpy --user**

### 2.2 Set Search Path

After installing the dependent modules, if the working path is the root directory of the extracted folder (that is, the folder where the manual is located), users can use autoBioSeqpy directly in the command line window (CMD window).

If users want to use it in their own python script, there are two ways to add modules to the search path:

1. If autoBioSeqpy is already in the python search path, adding a line to the python script is sufficient:

**import autoBioSeqpy**

or

**from autoBioSeqpy import \***

1. Otherwise, users could add the location into sys.path:

**import sys**

**libPath = /the/path/of/the/folder**

**sys.path.append(libPath)**

Then all modules are available. An example in jupyter notebook is provided, which uses the provided module for data processing and users can get it in ‘**notebook/tutorial in jupyter notebook.html**’, or see the section ‘**Using autoBioSeqpy in Other Work**’.

## 3 Quick Start

There are two ways to use autoBioSeqpy, one is to use the script running.py as a standalone application, and the other is to integrate it into a python script as a module. We will introduce both ways in separated sections.

### 3.1 Using autoBioSeqpy as Standalone Application

#### 3.1.1 Training and predict

If the dependent modules are installed (in section 2.1), a standalone script **running.py** is available. To test it, just open a command line window (or terminal in Linux) and make the working path (i.e. current folder) to the location of autoBioSeqpy. Then test:

**python running.py –help**

if the help document is showed without error, it’s available. Users can then perform a shot test:

**python running.py --dataType protein --dataEncodingType dict --dataTrainFilePaths examples/typeIIIsecretedeffectors/data/train\_pos.txt examples/typeIIIsecretedeffectors/data/train\_neg.txt --dataTrainLabel 1 0 --dataTestFilePaths examples/typeIIIsecretedeffectors/data/test\_pos.txt examples/typeIIIsecretedeffectors/data/test\_neg.txt --dataTestLabel 1 0 --modelLoadFile examples/typeIIIsecretedeffectors/model/protein\_CNN1D\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 60 --epochs 20 --spcLen 100 --shuffleDataTrain 1 --modelSaveName tmpMod.json --weightSaveName tmpWeight.bin --noGPU 1 --paraSaveName parameters.txt**

The use of parameters is in “parameters” section or in the help document generated by ‘--help’.

Since there are too many parameters to write on the command line, an alternative way is to write the parameters to a text file, for example the file parameters.txt contains the details information (All spaces below can be changed to line breaks):

**--useKMer None**

**--batch\_size 60**

**--shuffleDataTrain True**

**--dataTestLabel 1 0**

**--dataType protein**

**--dataSplitScale None**

**--noGPU True**

**--epochs 20**

**--verbose True**

**--figDPI 300**

**--paraFile None**

**--shuffleDataTest False**

**--loss binary\_crossentropy**

**--modelSaveName tmpMod.json**

**--paraSaveName parameters.txt**

**--dataTrainLabel 1 0**

**--outSaveFolderPath tmpOut**

**--spcLen 100**

**--optimizer optimizers.Adam()**

**--weightSaveName tmpWeight.bin**

**--inputLength None**

**--firstKernelSize**

**--modelLoadFile examples/typeIIIsecretedeffectors/model/protein\_CNN1D\_model.py**

**--KMerNum 3**

**--weightLoadFile None**

**--dataTrainFilePaths examples/typeIIIsecretedeffectors/data/train\_pos.txt examples/typeIIIsecretedeffectors/data/train\_neg.txt**

**--showFig True**

**--metrics acc**

**--saveFig True**

**--dataEncodingType dict**

**--savePrediction True**

**--dataTestFilePaths examples/typeIIIsecretedeffectors/data/test\_pos.txt examples/typeIIIsecretedeffectors/data/test\_neg.txt**

This file can then be used as a command line:

**python running.py –paraFile parameters.txt**

#### 3.1.2 Predict using the built model

Sometimes users will want to use the built model to predict the new data, and **predicting.py** is available. Since the data encoding during training depends on the parameters, few parameters are required during training. Using the same example as in section 3.1.1, the command line becomes:

**python running.py --dataType protein --dataEncodingType dict --dataTrainFilePaths examples/typeIIIsecretedeffectors/data/train\_pos.txt examples/typeIIIsecretedeffectors/data/train\_neg.txt --dataTrainLabel 1 0 --dataTestFilePaths examples/typeIIIsecretedeffectors/data/test\_pos.txt examples/typeIIIsecretedeffectors/data/test\_neg.txt --dataTestLabel 1 0 --modelLoadFile examples/typeIIIsecretedeffectors/model/protein\_CNN1D\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 60 --epochs 20 --spcLen 100 --shuffleDataTrain 1 --modelSaveName tmpMod.json --weightSaveName tmpWeight.bin --noGPU 1 --paraSaveName parameters.txt**

The three parameters highlighted above are necessary for making predictions, where ‘--modelSaveName’ and ‘--weightSaveName’ are the keras model files and related weights, and ‘--paraSaveName’ are the parameters used when training. Then, for prediction, the command line will become:

**Python predicting.py --paraFile tmpOut/parameters.txt --dataTestFilePaths examples/typeIIIsecretedeffectors/data/test\_pos.txt --predictionSavePath tmpout/indPredictions.txt**

That is, if the test data and parameters are sufficient (because the module and weight are recorded in parameters.txt), if the user wants to print the output to STDOUT, you can ignore **--predictionSavePath**.

### 3.2 Using autoBioSeqpy in Other Work

Because autoBioSeqpy can encode FASTA sequences into matrices, sometimes users may just want to use feature encoding instead of modeling. The autoBioSeqpy can be used as a module, so it can be used for other tasks. We provided a jupyter notebook to explain how to use it, so please open the file in ‘**notebook/ tutorial in jupyter notebook.ipynb**’ in jupyter notebook. If the jupyter notebook is not installed, users could use the HTML and PDF version alternatively (but only for reading, no interaction included in pure HTML and PDF files).

## The Design and Parameters

The autoBioSeqpy contains several parts to support automatic data transferring and modeling. Therefore, this section will introduce the framework, parameters and some important mechanisms.

### 4.1 Design

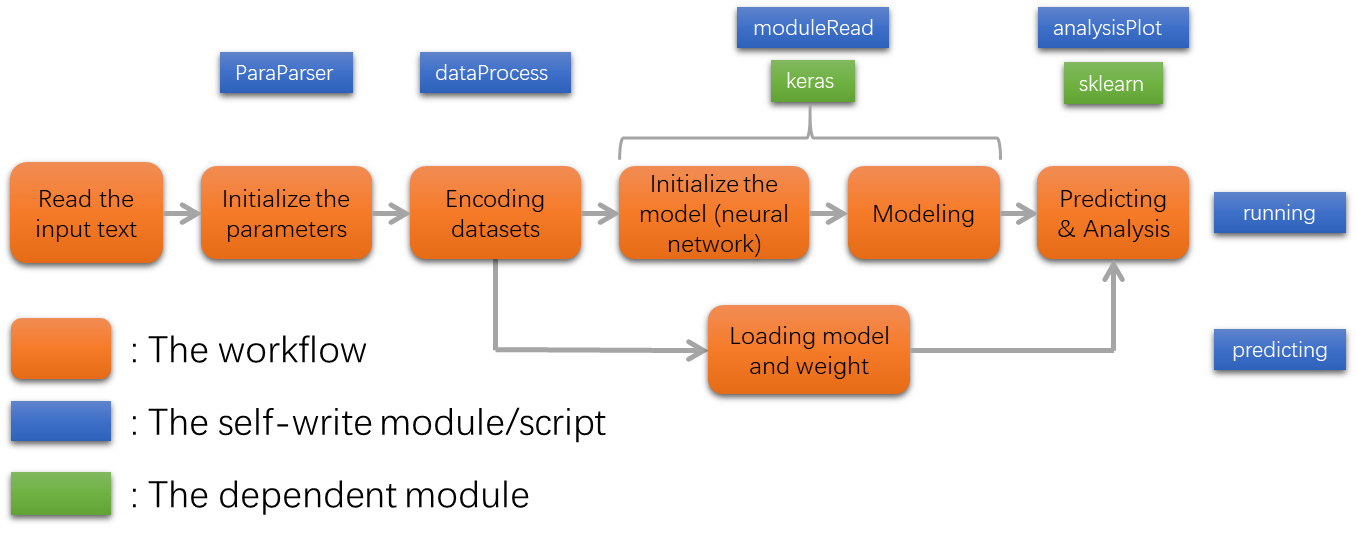


Figure 1: the design of autoBioSeqpy

The construction is listed in fig. 1, which is consisted of 6 parts (5 parts for prediction) and is supported by 4+2 modules. The workflow is very simple, so the functions of the modules are completely different. If users want to use the components of autoBioSeqpy, usually the ‘dataProcess’ and ‘analysisPlot’ are used, and sometimes ‘moduleRead’ is used. But the paraParser will be ignored unless the workflow be used directly.

### 4.2 Parameters

Because autoBioSeqpy can be modeled using keras, not only the encoding parameters, but also keras parameters will be explained here. Please note that all parameters in this section are for the standalone version, the usage of the modules are in the file ‘notebook/tutorial in jupyter notebook.ipynb’.

#### 4.2.1 Basic information

The information in this section could be got in command line window by typing:

**python running.py --help**

The details are as follows.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Parameter Name | Must be used | Default Value | Format | Description |
| --dataType | Yes | None | One unit in  {protein, dna, rna} | The type of the data, should be protein, dna or rna (upper case is supported either) |
| --dataEncodingType | No | dict | One unit in {onehot, dict} | the type for encoding the data, if dict choosed, a character (e.g. A/G/C/T for DNA) is represented as a number (such as A:1 T:2 C:3 T:4), and if onehot choosed, a character will be represented as an array (such as A:[1,0,0,0] G:[0,1,0,0] C:[0,0,1,0] T[0,0,0,1]) |
| --spcLen | No | 100 | int | The length of the input sequence which will be used for enconding. If the length of an input sequence is larger than the 'spcLen', the exceed part will be ignored, and if the length is less than 'spcLen', zeros (or zero arrays) will be added to make the length to 100. |
| --dataTrainFilePaths | Yes | None | List of paths:  [path1,path2,…] | The inputs are separated by space. FASTA data should be provided in separated files according to the labels, if two labels provided, there should be at least two FASTA files. For example, there are two files containing positive and negative samples separately, the inputs are:  --dataTrainFilePaths the/path/of/the/positive/file1.fasta the/path/of/the/negative/file2.fasta |
| --dataTrainLabel | Yes | None | List of labels:  [1, 0, 1, 2, …] | The label of each file, and the length should be the same as --dataTrainFilePaths. As the example above, two FASTA file provided, so the label could be:  --dataTrainLabel 1 0 |
| --dataTestFilePaths | No | None | List of paths:  [path1,path2,…] | Conflicting: --dataSplitScale  The data for independent test. The format and usage are the same as --dataTrainFilePaths.  NOTE: if no independent data provided, this parameter could be ignored, the dataset for testing will be generated from the training data by spliting it according to '--dataSplitScale' |
| --dataTestLabel | No | None | List of labels:  [1, 0, 1, 2, …] | Conflicting: --dataSplitScale  The format is the same as --dataTrainLabel but for the test data. The length should be the same as --dataTestFilePaths |
| --outSaveFolderPath | No | None | string | A folder path for saving the outputs, if not provide, only STDOUT will be generated. |
| --showFig | No | True | bool | Switch to show the figures |
| --saveFig | No | True | bool | Switch to save the figures to '--outSaveFolderPath' |
| --figDPI | No | 300 | int | The dpi of the figure |
| --savePrediction | No | True | bool | Switch to save the predictions to '--outSaveFolderPath' |
| --dataSplitScale | No | 0.8 | float | Conflicting: --dataTestFilePaths, --dataTestLabel  A scale for spliting the training data into two piece, one is for training and the other for independent test.  For example, if the '--dataTestLabel' is 0.8, then the training data-set is 80% and the test data-set is 20% from the provided data. |
| --modelLoadFile | Yes | None | string | Load the Keras model for modeling. Both user made model (in .py file) and keras model (in .json file) are supported. Few templates in python script (e.g. .py file) are provided in folder 'models'. |
| --weightLoadFile | No | None | string | Relating: --modelLoadFile  A built Keras model could save weight file as well, thus the weight file could be loaded when loading the model |
| --shuffleDataTrain | No | True | bool | shuffle the sequence of training data |
| --shuffleDataTest | No | False | bool | shuffle the sequence of test dataset. The default is False because the sequence will not change the modeling performance. |
| --batch\_size | No | 40 | int | The parameter for keras to decide the size of batch (e.g. the number of used data) when training |
| --epochs | No | 100 | int | The parameter for keras to decide the number of iteration of training |
| --useKMer | No | False | bool | To considering the environment of a residue. For example, if a sequence is ATTACT, and '--KMerNum' is 3, then the first A will be considered as 'ATT' and the shape of dataset will be expanded accordingly (see section ‘kmer’ for more details). |
| --KMerNum | No | 3 | int | The length of the sequence which will be taken as environment, please see the details of '--UseKMer' |
| --inputLength | No | None | int | A parameter for 2D layer. This parameter is added to modify the size of the built model before compiling. The "batch\_input\_shape" and "input\_length" will be changed according to this parameter. If not provided, program will change the size to the current shape automaticly if a 2D convolution layer is used as the first layer. |
| --firstKernelSize | No | None | int | A parameter for changing the kernel size of the first layer. Since the shape of input dataset might be not fit for the first layer, this parameter is added to modify the size of the built model before compiling. The "kernel\_size" will be changed according to this parameter. If not provided, program will change the size to the current shape automaticly. |
| --loss | No | binary\_crossentropy | string | Keras parameter, available candidates are 'mean\_squared\_error', 'mean\_absolute\_error', 'mean\_absolute\_percentage\_error', 'mean\_squared\_logarithmic\_error', 'squared\_hinge', 'hinge', 'categorical\_hinge', 'logcosh', 'categorical\_crossentropy', 'sparse\_categorical\_crossentropy', 'binary\_crossentropy', 'kullback\_leibler\_divergence', 'poisson', 'cosine\_proximity'  (reference https://keras.io/losses/) |
| --metrics | No | ['acc'] | list of the metrics [‘acc’,’mae’,…] | Keras parameters. Available candidates are 'acc', 'mae', 'binary\_accuracy', 'categorical\_accuracy', 'sparse\_categorical\_accuracy', 'top\_k\_categorical\_accuracy', 'sparse\_top\_k\_categorical\_accuracy'.  Note: The loss function is available here.  reference https://keras.io/metrics/ |
| --modelSaveName | No | None | string | Save the built model in json format. |
| --weightSaveName | No | None | string | Save the weights of built model in binary format. |
| --noGPU | No | None | bool | Only using CPU for modeling, sometimes is useful for debugging |
| --paraFile | No | None | string | Sometimes using command line is not easy for use, write the parameters into file is better for modification. The parameters in the paraFile is the same as writen in command line, such as '--noGPU 1 --figDPI 600 ...' |
| --paraSaveName | No | None | string | Save used parameters into file. Sometimes saving the parameters into a file will make the model easier for prediction. |
| --verbose | No | False | bool | See a detailed output when the script running. |

We provide as many parameters as possible to build the model explicitly, but usually few parameters are sufficient.

In the table above, the parameter ‘--useKMer’ and ‘--modelLoadFile’ will be described in the next sections.

#### 4.2.2 Sequence Encoding and KMer

In this section, we will introduce the way of sequence encoding and the mechanism of KMer. KMer is not a new concept in sequence data processing, which means using the environment for modeling, but the implementation can be misunderstood, so we have written a description of such content.

Consider that we have a 6 base DNA sequence, such as ‘ATTACG’. In autoBioSeqpy, ‘dict’ and ‘onehot’ are used for encoding. If we do not use kmer, the sequence ‘ATTACG’ will be encoded separately as 122134 for ‘dict’ with the hash table (or dict) {A:1, T:2, C:3, T:4} and as for ‘onehot’ with the hash table {A:[1,0,0,0],T:[0,1,0,0],C:[0,0,1,0],T:[0,0,0,1]}. The two encoding methods are very easy to understand, and each base in the hash table is unique. However, if we consider using 2-Mer, the ‘ATTACG’ will become 5 (6-2+1) pieces: ‘AT’, ‘TT’, ‘TA’, ‘AC’, ‘CG’, but it is currently an element increased from 4 to 16 in the hash table. If ‘dict’ is used to encode the 2-Mer case, the result is still an array containing 5 elements ranging from 1 to 16, but if ‘onehot’ is used, the shape of the matrix becomes 5x16 and each row has only one and fifteen zeros such as:

. The positions are determined in this way:

1. We have for a single base and 4 for the length
2. When 2 bases ‘’ used, the following formula is used for find the position of 1:
3. Similarly, in kmer case: ‘’, the position of ‘1’ becomes:

Using these formulas, you will find that “onehot” may not be a good idea for encoding proteins because the encoded matrix is too sparse.

#### 4.2.3 The use of keras module

In autoBioSeqpy, keras is used for deep learning, but users still have to build neural networks manually because there is no “universal” neural network for any type of data. But to make modeling (especially the construction of the framework) more simplicity, we provide some templates in .py script. Users can use them directly by using parameter ‘-- modelLoadFile’ or create model scripts based on them.

Alternatively, if a neural network is built, the parameter ‘--modelSaveName’ and ‘--weightSaveName’ are provided to save the model as JSON and the weights in binary format. Users can use them in the next experiments with ‘--modelLoadFile’ and ‘--weightLoadFile’.

## 5. Conclusion

This document is provided for users to know autoBioSeqpy. As an open source tool, we have documented all the code and function, but this document is still a better way to understand the framework.

We looking forward to receiving any bug reports and suggestions, please feel free to contact us anytime (ljs@swmu.edu.cn)