# Manual

## Brief Introduction

This document is a guide for users to get touch with our tool ‘autoBioSeqpy’ for modeling and analyzing Protein, DNA and RNA data, including installation, quick start and a brief introduction for using the modules separately (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 could find the data format, the inner mechanism of encoding and figure out how to use this tool.

## Installation

All the code for autoBioSeqpy’ is wrote in Python, and no mixture coding (e.g. C/C++) used in this project, thus the installation is quite easy, after the dependence solved, the only thing to do is to make the path as work path or put the code into a search path.

### 2.1 Dependence

Few python modules are necessary for out tool, they are **re, numpy, importlib, sklearn** and **keras**. Since all the modules are included in anaconda3, users could solve the module dependence by installing anaconda3 (2 is not suggested) in its official site <https://www.anaconda.com/>. Alternatively, users could install the modules manually such as using pip or other installer. If using pip for installing the dependent module, the command would be:

**pip install numpy**

for root, or

**pip install numpy --user**

for a normal user.

### 2.2 Set Search Path

After the dependent module installed, autoBioSeqpy could be used in the command line window (CMD window) directly if the work path is the root of the extracted folder (i.e. the folder where the manual located).

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

1. If autoBioSeqpy is already in the python search path, then add a line in the python script is enough:  
   **import autoBioSeqpy**  
   or  
   **from autoBioSeqpy import \***
2. Otherwise, users could add the location into sys.path:  
   **import sys  
   libPath = /the/path/of/the/folder  
   sys.path.append(libPath)**  
   then all the modules are available. An example in jupyter notebook is provided for using the provided module for data processing, users could get it at ‘**notebook/tutorial in jupyter notebook.html**’. Or see the section ‘**Using autoBioSeqpy in Other Work**’.

## 3 Quick Start

There are two ways for using autoBioSeqpy, one is to use the script running.py as a standalone application, another is integrating it into a python script as a module. We will introduce the two ways in separated sections.

### 3.1 Using autoBioSeqpy as Standalone Application

#### 3.1.1 For training and predicting

If the dependent modules are installed (in section 2.1), the standalone script **running.py** is available. To test it, just open a command line window (or terminal in Linux), make the work path (i.e. current folder) to the location of autoBioSeqpy. Then have a test:  
 **python running.py –help**  
if the help document is listed without error, then it’s available. Then users could have a brief test:

**python running.py --dataType protein --dataEncodingType dict --dataTrainFilePaths data\\TypeIIIprotein\\train\_pos.txt data\\TypeIIIprotein\\train\_neg.txt --dataTrainLabel 1 0 --dataTestFilePaths data\\TypeIIIprotein\\test\_pos.txt data\\TypeIIIprotein\\test\_neg.txt --dataTestLabel 1 0 --modelLoadFile models\\Bi\_LSTM.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 60 --epochs 20 --spcLen 100 --shuffleDataTrain 1**

if figures throwed out and a folder tmpOut\_rna with 2 text files and 2 figures generated, the quick start is successed. The usage of the parameters is in section ‘parameters’, or in the help document generated by ‘--help’.

Since the parameters would be too many to write in command line, an alternative way is to write the parameters in a text file, for example the file parameters.txt contain the details (all the spaces below are allowed to change into line breaks):

**--dataType protein**

**--dataEncodingType dict**

**--dataTrainFilePaths data\\TypeIIIprotein\\train\_pos.txt data\\TypeIIIprotein\\train\_neg.txt**

**--dataTrainLabel 1 0**

**--dataTestFilePaths data\\TypeIIIprotein\\test\_pos.txt data\\TypeIIIprotein\\test\_neg.txt**

**--dataTestLabel 1 0**

**--modelLoadFile models\\Bi\_LSTM.py --verbose 1**

**--outSaveFolderPath tmpOut --savePrediction 1**

**--saveFig 1 --batch\_size 60 --epochs 20 --spcLen 100 --shuffleDataTrain 1**

Then the file could be used as the command line:

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

#### 3.1.2 For predicting via built model

Sometimes users will want to use the built model for predicting the unknow data, this time the **predicting.py** is available. Since data encoding is depend on the parameters when training, thus few parameters are necessary when training. Using the same example as in section 3.1.1, the command line become to:

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

The highlighted three parameters above are necessary for predicting, where the ‘--modelSaveName’ and ‘--weightSaveName’ are the keras model file and related weights, the ‘--paraSaveName’ is the parameters used when training. Then for predicting, the command line becomes to:

**python predicting.py --paraFile parameters.txt --dataTestFilePaths data\\CRISPRCas9\\high\_activity\_sgRNA.txt data\\CRISPRCas9\\low\_activity\_sgRNA.txt --predictionSavePath tmpout/indPredictions.txt**

That is, providing the test data and the parameters are enough (since the module and weight were recorded in parameters.txt), the **–predictionSavePath** could be ignored if user wants to print the output to STDOUT.

### 3.2 Using autoBioSeqpy in Other Work

Since autoBioSeqpy could encoding the FASTA sequence into matrix, sometimes users might only want to use encoding instead of modeling. autoBioSeqpy is able to be used as a module, thus it is available for engaging other work. We provided a jupyter notebook for explaining how to use it, thus 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

autoBioSeqpy contain few parts for supporting the automatic data transferring and modeling. Thus the framework, parameters and some important mechanisms will be introduced in this section.

### 4.1 Design

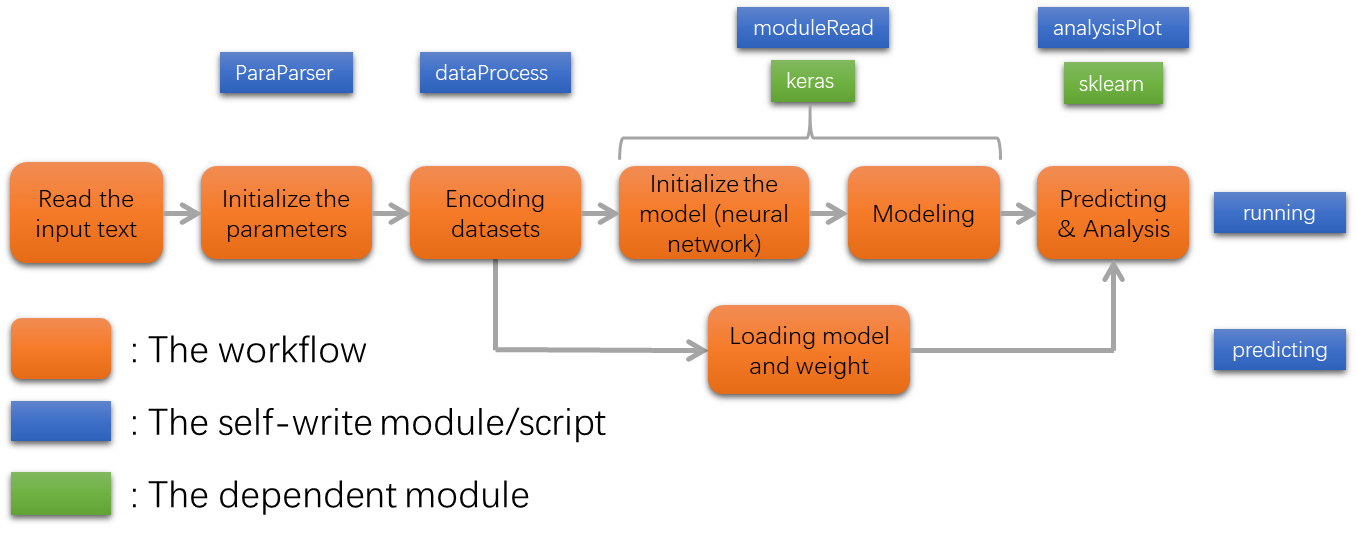


Figure 1: the design of autoBioSeqpy

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

### 4.2 Parameters

Since autoBioSeqpy could using keras for modeling, not only the parameters for encoding, but also the parameters for keras will be explained here. Note that all the parameters in this section is for the standalone version, the usage for 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 is 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 provided as many parameters as possible for making a model explicitly, but usually few of the parameters is enough.

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

#### 4.2.2 Sequence Encoding and KMer

In this section, the way autoBioSeqpy encoding and the mechanism of KMer will be introduced. KMer is not a new concept in sequence data processing, which means using the environment for modeling, but the implementation might cause misunderstanding, therefore we wrote this section for explaining the such things.

Considering that we have a DNA sequence with 6 base, such as ‘ATTACG’. In autoBioSeqpy, ‘dict’ and ‘onehot’ is used for encoding. If we do not use kmer, the sequence ‘ATTACG’ will be encoded separately to 122134 for ‘dict’ with the hash table (or dict) {A:1, T:2, C:3, T:4} and to 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 way for encoding is quite easy for understand and every base is unique in the hash table. However, if we considering the 2-Mer, then the ‘ATTACG’ become 5 (6-2+1) pieces: ‘AT’, ‘TT’, ‘TA’, ‘AC’, ‘CG’, but currently the elements in a hash table raised to 16 from 4. If ‘dict’ used for encoding the 2-Mer case, the result is still an array with 5 elements in the range from 1 to 16, but if ‘onehot’ used, the shape of the matrix becomes to 5x16 and in every row there is only one ‘1’ and fifteen ‘0’ such as . The positions are decided as 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:

With the formulas you can find that the ‘onehot’ might not be a good idea for encoding protein since there are usually more than 20 residues an the matrix will be too sparse.

#### 4.2.3 The use of keras module

In autoBioSeqpy, keras is used for deep learning, but user still has to construct the neural network manually because there is not a ‘all-purpose’ neural network for any type of data. But to make the modeling (especially the construction of the framework) more simplicity, we provided few templates in .py script, users could use them directly by using parameter ‘-- modelLoadFile’ or make a model script based on them.

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

## 5. Few modeling examples

We provided few examples to show the way of using the provided models, users could use the command lines below as a reference for their own works.

##################################################################

# protein\_BiLSTM\_model.py

python running.py --dataType protein --dataEncodingType dict --dataTrainFilePaths data\\TypeIIIprotein\\train\_pos.txt data\\TypeIIIprotein\\train\_neg.txt --dataTrainLabel 1 0 --dataTestFilePaths data\\TypeIIIprotein\\test\_pos.txt data\\TypeIIIprotein\\test\_neg.txt --dataTestLabel 1 0 --modelLoadFile models\\protein\_BiLSTM\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 60 --epochs 20 --spcLen 100 --shuffleDataTrain 1

##################################################################

# protein\_CNN1D\_model.py

python running.py --dataType protein --dataEncodingType dict --dataTrainFilePaths data\\TypeIIIprotein\\train\_pos.txt data\\TypeIIIprotein\\train\_neg.txt --dataTrainLabel 1 0 --dataTestFilePaths data\\TypeIIIprotein\\test\_pos.txt data\\TypeIIIprotein\\test\_neg.txt --dataTestLabel 1 0 --modelLoadFile models\\protein\_CNN1D\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 60 --epochs 20 --spcLen 100 --shuffleDataTrain 1

##################################################################

# protein\_Conv1D+BiLSTM\_model.py

python running.py --dataType protein --dataEncodingType dict --dataTrainFilePaths data\\TypeIIIprotein\\train\_pos.txt data\\TypeIIIprotein\\train\_neg.txt --dataTrainLabel 1 0 --dataTestFilePaths data\\TypeIIIprotein\\test\_pos.txt data\\TypeIIIprotein\\test\_neg.txt --dataTestLabel 1 0 --modelLoadFile models\\protein\_Conv1D+BiLSTM\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 60 --epochs 20 --spcLen 100 --shuffleDataTrain 1

##################################################################

#protein\_CNN2D\_model.py

python running.py --dataType protein --dataEncodingType onehot --dataTrainFilePaths data\\TypeIIIprotein\\train\_pos.txt data\\TypeIIIprotein\\train\_neg.txt --dataTrainLabel 1 0 --dataTestFilePaths data\\TypeIIIprotein\\test\_pos.txt data\\TypeIIIprotein\\test\_neg.txt --dataTestLabel 1 0 --modelLoadFile models\\protein\_CNN2D\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 60 --epochs 20 --spcLen 100 --shuffleDataTrain 1 --firstKernelSize 20 5

##################################################################

#prtein\_Conv2D+BiLSTM\_model.py

python running.py --dataType protein --dataEncodingType onehot --dataTrainFilePaths data\\TypeIIIprotein\\train\_pos.txt data\\TypeIIIprotein\\train\_neg.txt --dataTrainLabel 1 0 --dataTestFilePaths data\\TypeIIIprotein\\test\_pos.txt data\\TypeIIIprotein\\test\_neg.txt --dataTestLabel 1 0 --modelLoadFile models\\prtein\_Conv2D+BiLSTM\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 60 --epochs 20 --spcLen 100 --shuffleDataTrain 1 --firstKernelSize 20 5

##################################################################

#protein\_MLP\_model.py

python running.py --dataType protein --dataEncodingType onehot --dataTrainFilePaths data\\TypeIIIprotein\\train\_pos.txt data\\TypeIIIprotein\\train\_neg.txt --dataTrainLabel 1 0 --dataTestFilePaths data\\TypeIIIprotein\\test\_pos.txt data\\TypeIIIprotein\\test\_neg.txt --dataTestLabel 1 0 --modelLoadFile models\\protein\_MLP\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 60 --epochs 20 --spcLen 100 --shuffleDataTrain 1

##################################################################

protein\_CNN1D+GlobalMaxPooling\_model.py

python running.py --dataType protein --dataEncodingType dict --dataTrainFilePaths data\\TypeIIIprotein\\train\_pos.txt data\\TypeIIIprotein\\train\_neg.txt --dataTrainLabel 1 0 --dataTestFilePaths data\\TypeIIIprotein\\test\_pos.txt data\\TypeIIIprotein\\test\_neg.txt --dataTestLabel 1 0 --modelLoadFile models\\protein\_CNN1D+GlobalMaxPooling\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 60 --epochs 20 --spcLen 100 --shuffleDataTrain 1

##################################################################

protein\_LSTM\_model.py

python running.py --dataType protein --dataEncodingType dict --dataTrainFilePaths data\\TypeIIIprotein\\train\_pos.txt data\\TypeIIIprotein\\train\_neg.txt --dataTrainLabel 1 0 --dataTestFilePaths data\\TypeIIIprotein\\test\_pos.txt data\\TypeIIIprotein\\test\_neg.txt --dataTestLabel 1 0 --modelLoadFile models\\protein\_LSTM\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 60 --epochs 20 --spcLen 100 --shuffleDataTrain 1

##################################################################

protein\_Conv1D+LSTM\_model.py

python running.py --dataType protein --dataEncodingType dict --dataTrainFilePaths data\\TypeIIIprotein\\train\_pos.txt data\\TypeIIIprotein\\train\_neg.txt --dataTrainLabel 1 0 --dataTestFilePaths data\\TypeIIIprotein\\test\_pos.txt data\\TypeIIIprotein\\test\_neg.txt --dataTestLabel 1 0 --modelLoadFile models\\protein\_Conv1D+LSTM\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 60 --epochs 20 --spcLen 100 --shuffleDataTrain 1

##################################################################

DNA\_CNN2D\_model.py

python running.py --dataType dna --dataEncodingType onehot --dataTrainFilePaths data\\CRISPRCas9\\high\_activity\_sgRNA.txt data\\CRISPRCas9\\low\_activity\_sgRNA.txt --dataTrainLabel 1 0 --dataSplitScale 0.8 --modelLoadFile models\\DNA\_CNN2D\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 40 --epochs 60 --shuffleDataTrain 1 --spcLen 30 --firstKernelSize 4 3

##################################################################

#DNA\_2mer\_CNN2D\_model.py

python running.py --dataType dna --dataEncodingType onehot --dataTrainFilePaths data\\CRISPRCas9\\high\_activity\_sgRNA.txt data\\CRISPRCas9\\low\_activity\_sgRNA.txt --dataTrainLabel 1 0 --dataSplitScale 0.8 --modelLoadFile models\\DNA\_2mer\_CNN2D\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 40 --epochs 60 --shuffleDataTrain 1 --spcLen 30 --firstKernelSize 16 3 --useKMer 1 --KMerNum 2

##################################################################

#DNA\_3mer\_CNN2D\_model.py

python running.py --dataType dna --dataEncodingType onehot --dataTrainFilePaths data\\CRISPRCas9\\high\_activity\_sgRNA.txt data\\CRISPRCas9\\low\_activity\_sgRNA.txt --dataTrainLabel 1 0 --dataSplitScale 0.8 --modelLoadFile models\\DNA\_3mer\_CNN2D\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 40 --epochs 60 --shuffleDataTrain 1 --spcLen 30 --firstKernelSize 64 3 --useKMer 1 --KMerNum 3

##################################################################

#DNA\_CNN1D\_model.py

python running.py --dataType dna --dataEncodingType dict --dataTrainFilePaths data\\CRISPRCas9\\high\_activity\_sgRNA.txt data\\CRISPRCas9\\low\_activity\_sgRNA.txt --dataTrainLabel 1 0 --dataSplitScale 0.8 --modelLoadFile models\\DNA\_CNN1D\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 40 --epochs 60 --shuffleDataTrain 1 --spcLen 30

##################################################################

#DNA\_2mer\_CNN1D\_model.py

python running.py --dataType dna --dataEncodingType dict --dataTrainFilePaths data\\CRISPRCas9\\high\_activity\_sgRNA.txt data\\CRISPRCas9\\low\_activity\_sgRNA.txt --dataTrainLabel 1 0 --dataSplitScale 0.8 --modelLoadFile models\\DNA\_2mer\_CNN1D\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 40 --epochs 60 --shuffleDataTrain 1 --spcLen 30 --useKMer 1 --KMerNum 2

##################################################################

#DNA\_3mer\_CNN1D\_model.py

python running.py --dataType dna --dataEncodingType dict --dataTrainFilePaths data\\CRISPRCas9\\high\_activity\_sgRNA.txt data\\CRISPRCas9\\low\_activity\_sgRNA.txt --dataTrainLabel 1 0 --dataSplitScale 0.8 --modelLoadFile models\\DNA\_3mer\_CNN1D\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 40 --epochs 60 --shuffleDataTrain 1 --spcLen 30 --useKMer 1 --KMerNum 3

##################################################################

#DNA\_Conv1D+BiLSTM\_model.py

python running.py --dataType dna --dataEncodingType dict --dataTrainFilePaths data\\CRISPRCas9\\high\_activity\_sgRNA.txt data\\CRISPRCas9\\low\_activity\_sgRNA.txt --dataTrainLabel 1 0 --dataSplitScale 0.8 --modelLoadFile models\\DNA\_Conv1D+BiLSTM\_model.py --verbose 1 --outSaveFolderPath tmpOut --savePrediction 1 --saveFig 1 --batch\_size 40 --epochs 60 --shuffleDataTrain 1 --spcLen 30

## 6. Conclusion

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

We are looking forward for receiving any bug reporting and suggesting, please contact us anytime.