tutorial in jupyter notebook

December 11, 2019

Tutorial for using autoBioSeqpy as modules in script

Using the autoBioSeqpy via command line is a good way for data modeling, but usually users would like to use autoBioSeqpy for something more, such as converting the FASTA data into matrix or combine the modeling to another workflow.

This notebook provided the usage for using autoBioSeqpy as a library which could be imported normally. Moreover, few alternatives provided for some special case. We hope this tutorial could help user for understanding this tool deeply.

This notebook is available in jupyter notebook (editing and running is possible), but if you didn't install jupyter notebook, two copies in PDF and HTML version are available as well (read only).

1. First step: Initializing

As the first step, initializing is necessary for autoBioSeqpy by setting the search path.

The method'import' in python is a normal way for using a module in a default search path, but this time autoBioSeqpy didn't provide a way for 'install' and thus the search path is necessary for providing.

If autoBioSeqpy is already in your search path, you can import it directly.

1.1 Initializing the search path

There are several ways for importing the self-made modules/libraries. Considering that user might have different environment, please change the variable 'libPath' into the path where the tool located.

An alternative is provided, and available if uncomment it.

```
In [1]: import os, sys
        libPath = '../' #please change it into your search path if necessary
        sys.path.append(libPath)
        #alternative
        #os.chdir(libPath)
```

```
In [2]: import numpy as np #for some analysis
```

2. Data processing

   Usually, users will write a script to converting the FASTA into matrix, here autoBioSeqpy provided few ways for the matrix creation.

2.1 import the related library

Library ‘dataProcess’ is provided for matrix creation form FASTA data. The necessary function such as suffle the samples and spliting the dataset into pieces (for cross validation) are provided as well.

Since the location is added into the search path in section 1.1, here we only need to import it as a module.

```
In [3]: import dataProcess
```

2.2 Usage of module dataProcess

To load a dataset, first we need to instantiate an object and then using loading.

When intorducing it, some cases and parameters will be explained as well.

2.2.1 A detailed description for training data

dataType

Since Protein, DNA and RNA have their FASTA, we have to decide the type of this data. In our standalone script, ‘dataType’ is a parameter, but here we don’t have to determine it as a parameter directly.

dataEncodingType

This is a parameter for set the way to encoding the FASTA into matrix. Currently there are two encoding types available: ‘onehot’ and ‘dict’. 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).

Alternetivly, if choose ‘onehot’,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]).

In this example, only ‘dict’ is used since it is better for displaying.

```
In [4]: dataEncodingType = 'dict'
```

useKMer and KMerNum

Usually we would like to consider taking not only one FASTA residue for encoding, but also its neighbors. The parameter ‘useKMer’ is an implementation for the environment encoding.

For example, if a sequence is ATTACT, and ‘KMerNum’ is 3, then the first A will be considered as ‘ATT’.

Note that the shape of dataset will be expanded accordingly (see the manual for more details). And usually the ‘useKMer’ is used when ‘dataEncodingType’ set as ‘oneHot’. And thus in this notebook, we don’t use KMer since the encoding type is ‘dict’.

If you are interesting for the KMer, please change the ‘dataEncodingType’ as ‘onehot’ and turn on the ‘useKMer’ by set it into True.

```
In [5]: useKMer = False
        KMerNum = 3 #If useKMer is False, the KMerNum is inactive, and thus it doesn't matter ho
```

2.2.1.1 featureGenerator: the encoder

Now we can initialize a featureGenerator. A featureGenerator is a class for encoding the FASTA sequence.

There are three featureGenerator available: ProteinFeatureGenerator, DNAFeatureGenerator and RNAFeatureGenerator, you could use one of them according to the datatype.

In this notebook, protein data is used, thus we use ProteinFeatureGenerator as the featureGenerator.

```
In [6]: featureGenerator = dataProcess.ProteinFeatureGenerator(dataEncodingType, useKMer=useKMer
        #featureGenerator = dataProcess.DNAFeatureGenerator(dataEncodingType, useKMer=useKMer, K
        #featureGenerator = dataProcess.RNAFeatureGenerator(dataEncodingType, useKMer=useKMer, K
```

2.2.1.2 File format and class DataLoader

With the encoder, now it's possible to read the FASTA data and encode them into matrix.

autoBioSeqpy provided a class 'DataLoader' for handle all the file reading things. So here we need to introduce the format of the FASTA file.

File format

The so called 'file format' is the normal FASTA format. That is, a sequence is started by '> name&information' and then few lines of FASTA characters followed. There is no limitation of the number of characters in a line, you can try few lines with not more than 60 characters or only 1 line with all characters. For example, the both formats are supported and can even mixed in one file: >case1 only 1 line XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX... >case2 few lines XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX ... XXXXXXXXXXXXXXX

label

The only thing should be concerned is the label of a file. Usually there are two (e.g. 1/-1 or 1/0) or more (e.g. case1, case2, case3, ...) labels for a dataset, but the class DataLoader can handle one dataset with the same label, which means if a data has 3 labels, at least 3 files is necessary for reading.

In this notebook, the provided data is a binary classification, and therefore only two labels, 1 for positive samples and 0 negative samples, are used.

spcLen

Here is another problem: usually the length is not the same for different sequences.

To address it, the 'spcLen' is provided. 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 up to spcLen.

```
In [7]: spcLen = 100
```

DataLoader

The class 'DataLoader' is a class for loading a file, thus usually we need at least two DataLoader for different label.

In this notebook, two files, 'train_pos.txt' and 'train_neg.txt', provided for training dataset, which can be found in 'data' folder. The labels are '1' and '0' respectively.

```
In [8]: #the paths
        dataTrainFilePaths = ['../data/protein/train/train_pos.txt','../data/protein/train/train
        #the related labels
        dataTrainLabel = [1, 0]
        #a list for recording the DataLoader
        trainDataLoaders = []
        for i,dataPath in enumerate(dataTrainFilePaths):
            #init
            dataLoader = dataProcess.DataLoader(label = dataTrainLabel[i], featureGenerator=feat
            #file read
            dataLoader.readFile(dataPath, spcLen = spcLen)
            trainDataLoaders.append(dataLoader)
```

2.2.1.3 DataSetCreator: merge different dataLoader

After FASTA loading and encoding, now we can generate the matrix by merging the dataLoaders. The class 'DataSetCreator' is provided for the matrix merging and the necessary functions, such as sample shuffle and dataset split, are provided.

NOTE: Since the DataSetCreator is able to merge different DataLoader no matter whether the label is the same or not, thus if you have multiple files with the same label, you don't have to merge them by hand, just merger them here.

```
In [9]: #init
        trainDataSetCreator = dataProcess.DataSetCreator(trainDataLoaders)
```

Then we can generate the matrix by using the method 'DataSetCreator' if the test dataset is in other files.

The parameter 'toSuffle' is a switch to s

```
In [10]: #get dataset
         trainDataMat, trainLabelArr = trainDataSetCreator.getDataSet(toShuffle=True)
```

We can have a look of the matrix and labels, all of them are numpy array.

```
In [11]: print('Matrix with shape %d x %d:' %(trainDataMat.shape[0],trainDataMat.shape[1]))
         print(trainDataMat)
         print('\n')
         print('The labels with length %d:' %(len(trainLabelArr)))
         print(trainLabelArr)
```

```
Matrix with shape 1234 x 100:
[[13 10 16 ...  5 16  1]
 [13 10 12 ... 17  3 17]
 [13 18 18 ... 17  3  3]
 ...
 [13  4  8 ...  4 17 16]
 [13  2  8 ... 16  1 12]
 [13 10 17 ...  6  4  7]]


The labels with length 1234:
[1 1 1 ... 0 0 0]
```

Now with the use of featureGenerator, DataLoader and DataSetCreator, the dataset is generated.

2.2.2 The same process for test data

There are two way for generating the test data set: 1) from FASTA file or 2) from a built dataset.

2.2.2.1 generate test dataset from other FASTA files

Sometimes the test data if from another bath/experiment, in this case, just generate the test dataset in the same way when generating the training set.

For example, in this notebook, we can load provided test data in folder 'data/protein/test'.

NOTE: You can skip this subsection if you want to generate them by splitting. The parameter spcLen and object featureGenerator should be the same. And when generating the matrix, usually we don't have to shuffle the sample since it will not be used in training.

```
In [12]: #the paths
         dataTestFilePaths = ['../data/protein/test/test_pos.txt','../data/protein/test/test_neg
         #the related labels
         dataTestLabel = [1, 0]
         #a list for recording the DataLoader
         testDataLoaders = []
         for i,dataPath in enumerate(dataTestFilePaths):
             #init
             dataLoader = dataProcess.DataLoader(label = dataTestLabel[i], featureGenerator=feat
             #file read
             dataLoader.readFile(dataPath, spcLen = spcLen)
             testDataLoaders.append(dataLoader)

         testDataSetCreator = dataProcess.DataSetCreator(testDataLoaders)
         testDataMat, testLabelArr = testDataSetCreator.getDataSet(toShuffle=False)
```

We can have a look as well

```
In [13]: print('Matrix with shape %d x %d:' %(testDataMat.shape[0],testDataMat.shape[1]))
         print(testDataMat)
         print('\n')
         print('The labels with length %d:' %(len(testLabelArr)))
         print(testLabelArr)
```

```
Matrix with shape 308 x 100:
[[13 17 12 ... 12  6  7]
 [13  5  4 ... 19  7 10]
 [13  6  3 ...  4  7 11]
 ...
 [13 10  7 ...  5 19 15]
 [13 16  5 ...  2  3  1]
 [13  5 10 ...  6 16  6]]



The labels with length 308:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0]
```

2.2.2.2 Alternative: generate test dataset by spliting a built one

Sometimes, we only have one dataset and all the samples in it could be used for either training or test. Thus autoBioSeqpy provided a way for splitting the dataset into two.

The method is provided in class DataSetCreator. A new parameter dataSplitScale is provided to control the splitting ratio, if the 'dataSplitScale' is 0.8, then the training dataset is 80% and the test dataset is 20% from the provided dataset.

In this notebook, we use the trainDataSetCreator as the example.

NOTE: You can skip this subsection if you don't have to split dataset.

```
In [14]: dataSplitScale = 0.8
         trainDataMat, testDataMat, trainLabelArr, testLabelArr = trainDataSetCreator.getTrainTe
```

We can have a look as well

```
In [16]: print('Training:')
         print('Matrix with shape %d x %d:' %(trainDataMat.shape[0],trainDataMat.shape[1]))
         print(trainDataMat)
         print('\n')
         print('The labels with length %d:' %(len(trainLabelArr)))
         print(trainLabelArr)
         print('\n#############################################################################
         print('###############################################################################
         print('Testing:')
         print('Matrix with shape %d x %d:' %(testDataMat.shape[0],testDataMat.shape[1]))
         print(testDataMat)
         print('\n')
         print('The labels with length %d:' %(len(testLabelArr)))
         print(testLabelArr)
```

```
Training:
Matrix with shape 986 x 100:
[[13  4 14 ...  2 10  1]
 [13 14 19 ...  4 15 10]
 [13  6  5 ...  4 16 11]
 ...
 [13  4 10 ... 11  6 11]
 [13  7 15 ...  1  5  2]
 [13  1  4 ...  6 10  2]]


The labels with length 986:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]

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

Testing:
Matrix with shape 248 x 100:
[[13 15 10 ... 18  2 19]
 [13  2  2 ... 18  8 11]
 [13 10 10 ... 19  4  7]
 ...
 [13  2  1 ...  5 18 16]
 [13  4  8 ...  5 11  4]
 [13 19  7 ...  4  2 10]]


The labels with length 248:
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0]
```

3. Data Modeling and Testing

After the data generated in section 2, now the dataset is available for modeling. Since it is a matrix, the data could be used for not only deep learning but also other machine learning as well.

Here we made a brief introduce by using keras for deep learning, and provided a traditional example by using random forest at last.

3.1 Using keras for modeling

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow, CNTK, or Theano.

Keras is useful for modeling the dataset by both ‘dict’ and ‘onehot’, but the neural network and related parameters should be set carefully.

Few templates provided in the folder ‘models’, users could copy them directly and change few related parameters such as to make sure the shape of the data is the same as the kernel size of the first layer.

In this notebook, since ‘dict’ is used as the encoding, the 1D neural network is a good choice for modeling, therefore, the model in ‘model/CNN_Conv1D+GlobalMaxPooling.py’. Here the maxlen should be changed as the same with spcLen.

Here are two ways for using the model, one is write (or copy/post) the code in the script directly, another one is read a built model (in .json format) by using our provided module.

3.1 building keras model directly

3.1.1 model generating

As mentioned before, users could write any code for building keras neural network, but should modify the parameters manually.

```
In [14]: os.environ["CUDA_VISIBLE_DEVICES"] = '-1' #force using CPU, comment it for using GPU
```

```
In [15]: from keras.models import Sequential
         from keras.layers import Dense, Dropout, Activation
         from keras.layers import Embedding
         from keras.layers import Conv1D, GlobalMaxPooling1D
         from keras import optimizers



         # set parameters:
         max_features = 26
         embedding_size = 128
         filters = 250
         kernel_size = 3
         hidden_dims = 250
         batch_size = 40
         epochs = 25

         #the parameter which need to modified
         maxlen = spcLen


         print('Building model...')
         model = Sequential()
         # we start off with an efficient embedding layer which maps amino acids
         # indices into embedding_dims dimensions
         model.add(Embedding(max_features, embedding_size, input_length = maxlen))
         model.add(Dropout(0.2))
         # we add a Convolution1D, which will learn filters word group filters of
         # size filter_length:
```

```
model.add(Conv1D(filters,kernel_size,padding = 'valid',activation = 'relu',strides = 1)
# we use max pooling:
model.add(GlobalMaxPooling1D())
# We add a vanilla hidden layer:
model.add(Dense(hidden_dims))
model.add(Dropout(0.2))
model.add(Activation('relu'))
# We project onto a single unit output layer, and squash it with a sigmoid:
model.add(Dense(1))
model.add(Activation('sigmoid'))

model.compile(loss = 'binary_crossentropy',optimizer = optimizers.Adam(),metrics = ['ac

model.summary()
```

```
Using TensorFlow backend.


Building model...
WARNING:tensorflow:From C:\Users\jingr\Anaconda3\lib\site-packages\tensorflow\python\framework\o
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From C:\Users\jingr\Anaconda3\lib\site-packages\keras\backend\tensorflow_back
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 128)          3328
_________________________________________________________________
dropout_1 (Dropout)          (None, 100, 128)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 98, 250)           96250
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 250)               0
_________________________________________________________________
dense_1 (Dense)              (None, 250)               62750
_________________________________________________________________
dropout_2 (Dropout)          (None, 250)               0
_________________________________________________________________
activation_1 (Activation)    (None, 250)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 251
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 162,579
Trainable params: 162,579
```

```
Non-trainable params: 0
_________________________________________________________________
```

3.1.2 training

After the model built, now the dataset is ready for training, keras provided a framework for the training phase, just use it for the training dataset.

NOTE: The parameters batch_size and epochs are defined above.

analysisPlot

analysisPlot is a module provided for analyze the modeling process when using keras. We can import it easily since the search path is set before.

```
In [16]: import analysisPlot
```

```
In [17]: history = analysisPlot.LossHistory()
         model.fit(trainDataMat, trainLabelArr,batch_size = batch_size,epochs = epochs,validatio
```

```
WARNING:tensorflow:From C:\Users\jingr\Anaconda3\lib\site-packages\tensorflow\python\ops\math_op
Instructions for updating:
Use tf.cast instead.
WARNING:tensorflow:From C:\Users\jingr\Anaconda3\lib\site-packages\tensorflow\python\ops\math_gr
Instructions for updating:
Deprecated in favor of operator or tf.math.divide.
Train on 1110 samples, validate on 124 samples
Epoch 1/25
1110/1110 [==============================] - 1s 1ms/step - loss: 0.5961 - acc: 0.7243 - val_loss
Epoch 2/25
1110/1110 [==============================] - 1s 677us/step - loss: 0.5574 - acc: 0.7270 - val_lo
Epoch 3/25
1110/1110 [==============================] - 1s 708us/step - loss: 0.4929 - acc: 0.7532 - val_lo
Epoch 4/25
1110/1110 [==============================] - 1s 705us/step - loss: 0.4123 - acc: 0.8180 - val_lo
Epoch 5/25
1110/1110 [==============================] - 1s 708us/step - loss: 0.3591 - acc: 0.8414 - val_lo
Epoch 6/25
1110/1110 [==============================] - 1s 709us/step - loss: 0.3112 - acc: 0.8739 - val_lo
Epoch 7/25
1110/1110 [==============================] - 1s 712us/step - loss: 0.2653 - acc: 0.9000 - val_lo
Epoch 8/25
1110/1110 [==============================] - 1s 696us/step - loss: 0.2374 - acc: 0.9072 - val_lo
Epoch 9/25
1110/1110 [==============================] - 1s 722us/step - loss: 0.2035 - acc: 0.9297 - val_lo
Epoch 10/25
1110/1110 [==============================] - 1s 708us/step - loss: 0.1757 - acc: 0.9405 - val_lo
Epoch 11/25
1110/1110 [==============================] - 1s 709us/step - loss: 0.1533 - acc: 0.9414 - val_lo
Epoch 12/25
1110/1110 [==============================] - 1s 703us/step - loss: 0.1461 - acc: 0.9441 - val_lo
Epoch 13/25
```

```
1110/1110 [==============================] - 1s 713us/step - loss: 0.1130 - acc: 0.9676 - val_lo
Epoch 14/25
1110/1110 [==============================] - 1s 710us/step - loss: 0.1090 - acc: 0.9622 - val_lo
Epoch 15/25
1110/1110 [==============================] - 1s 761us/step - loss: 0.0890 - acc: 0.9703 - val_lo
Epoch 16/25
1110/1110 [==============================] - 1s 752us/step - loss: 0.0608 - acc: 0.9883 - val_lo
Epoch 17/25
1110/1110 [==============================] - 1s 712us/step - loss: 0.0637 - acc: 0.9811 - val_lo
Epoch 18/25
1110/1110 [==============================] - 1s 718us/step - loss: 0.0600 - acc: 0.9847 - val_lo
Epoch 19/25
1110/1110 [==============================] - 1s 723us/step - loss: 0.0443 - acc: 0.9901 - val_lo
Epoch 20/25
1110/1110 [==============================] - 1s 729us/step - loss: 0.0408 - acc: 0.9892 - val_lo
Epoch 21/25
1110/1110 [==============================] - 1s 719us/step - loss: 0.0280 - acc: 0.9964 - val_lo
Epoch 22/25
1110/1110 [==============================] - 1s 721us/step - loss: 0.0221 - acc: 0.9964 - val_lo
Epoch 23/25
1110/1110 [==============================] - 1s 714us/step - loss: 0.0228 - acc: 0.9946 - val_lo
Epoch 24/25
1110/1110 [==============================] - 1s 722us/step - loss: 0.0208 - acc: 0.9991 - val_lo
Epoch 25/25
1110/1110 [==============================] - 1s 697us/step - loss: 0.0142 - acc: 0.9973 - val_lo
```

```
Out[17]: <keras.callbacks.History at 0x236895849b0>
```

3.1.3 testing and output analysis

The frame for predicting is provided by keras as well, therefore we can make the predict as well.

```
In [18]: predicted_Probability = model.predict(testDataMat)
         prediction = model.predict_classes(testDataMat)
```

3.1.4 showing modeling figures and predicting preference

Usually users would like to know the predicting performance, therefore the related function is provided as well.

Users could get how the loss changes as the epoch increasing, and some metrices (ACC, Recall, MCC. . .) as well.

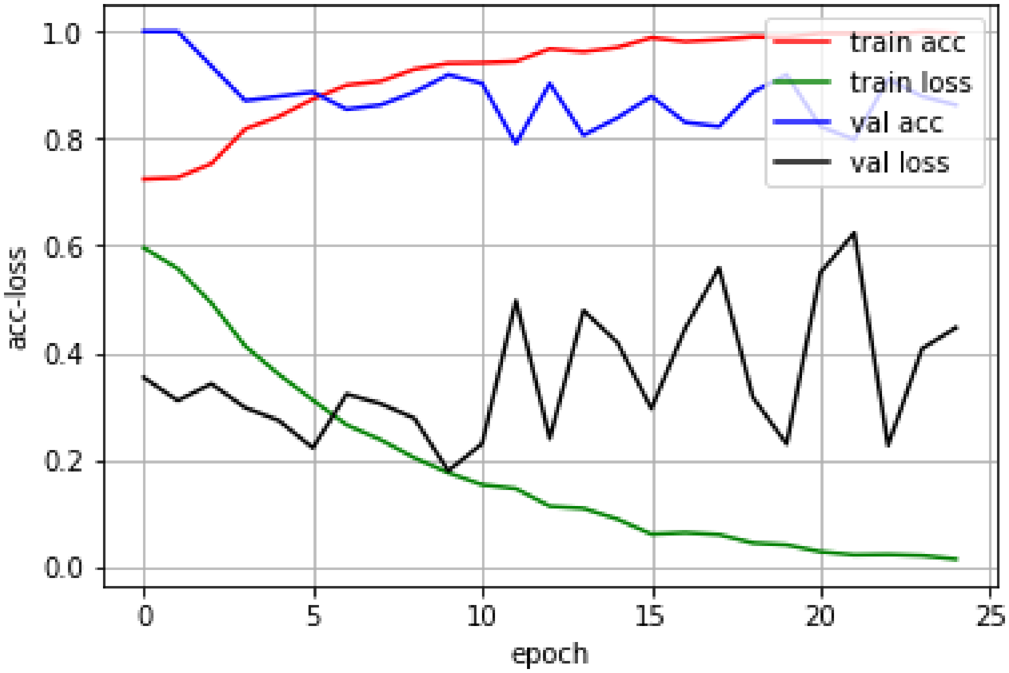
All the figure is available for save by change the parameter savePath to a real path.

This time the metrices are available in sklearn, import them at first.

```
In [19]: from sklearn.metrics import accuracy_score,f1_score,roc_auc_score,recall_score,precisio
```

The change of loss

```
In [20]: history.loss_plot('epoch',showFig=True,savePath=None)
```



confusion matrix and related metrices

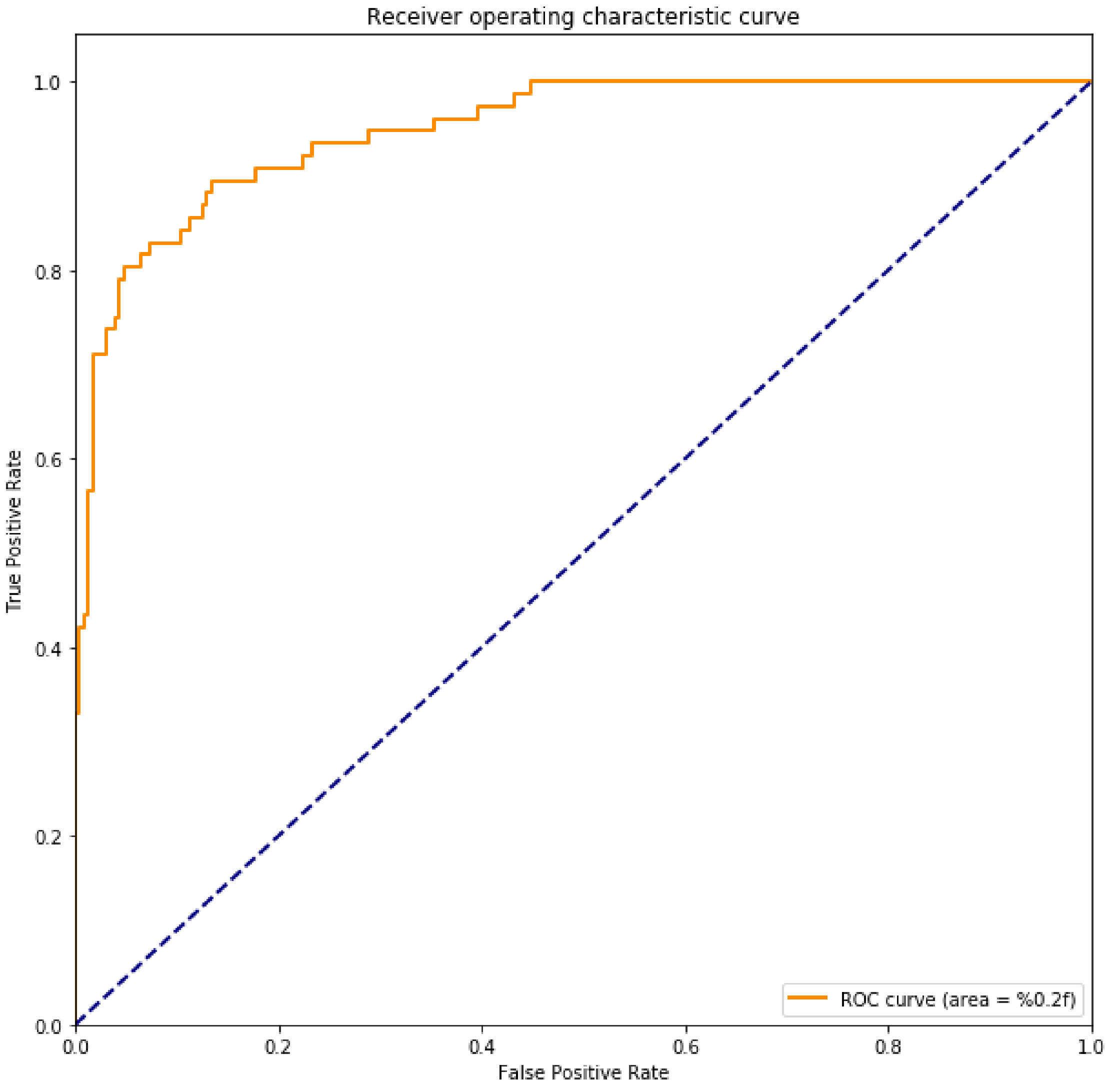
```
In [21]: cm=confusion_matrix(testLabelArr,prediction)
         print(cm)
         print("ACC: %f "%accuracy_score(testLabelArr,prediction))
         print("F1: %f "%f1_score(testLabelArr,prediction))
         print("Recall: %f "%recall_score(testLabelArr,prediction))
         print("Pre: %f "%precision_score(testLabelArr,prediction))
         print("MCC: %f "%matthews_corrcoef(testLabelArr,prediction))
         print("AUC: %f "%roc_auc_score(testLabelArr,prediction))
```

```
[[212  20]
 [ 13  63]]
ACC: 0.892857
F1: 0.792453
Recall: 0.828947
Pre: 0.759036
MCC: 0.721701
AUC: 0.871370
```

ROC curve

```
In [22]: analysisPlot.plotROC(testLabelArr,predicted_Probability,showFig=True,savePath=None)
```

```
<Figure size 432x288 with 0 Axes>
```



3.1.5 Save/Load a module (optional)

As mentioned before, keras is able to save a built model and read it again, it is available for establish a model without the data or using it for transfer learning.

Therefore, a shor part of the code (i.e. in our module 'moduleRead') is provided here for implement this function.

Model save

Not only the module, but also the weight could be saved.

```
In [23]: modelSavePath = './tmpModel.json'
         weightSavePath = './tmpWeight.bin'
```

```
In [24]: model_json = model.to_json()
         with open(modelSavePath, "w") as json_file:
             json_file.write(model_json)
         model.save_weights(weightSavePath)
```

Model Load

```
In [25]: from keras.models import model_from_json
```

```
In [26]: json_file = open(modelSavePath, 'r')
         loaded_model_json = json_file.read()
         json_file.close()
         loaded_model = model_from_json(loaded_model_json)
         if not weightSavePath is None:
             loaded_model.load_weights(weightSavePath)
```

Sometimes a loaded model should be recompiled before training

```
In [27]: model = loaded_model
         model.compile(loss = 'binary_crossentropy',optimizer = optimizers.Adam(),metrics = ['ac
         model.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 100, 128)          3328
_________________________________________________________________
dropout_1 (Dropout)          (None, 100, 128)          0
_________________________________________________________________
conv1d_1 (Conv1D)            (None, 98, 250)           96250
_________________________________________________________________
global_max_pooling1d_1 (Glob (None, 250)               0
_________________________________________________________________
dense_1 (Dense)              (None, 250)               62750
_________________________________________________________________
dropout_2 (Dropout)          (None, 250)               0
_________________________________________________________________
activation_1 (Activation)    (None, 250)               0
_________________________________________________________________
dense_2 (Dense)              (None, 1)                 251
_________________________________________________________________
activation_2 (Activation)    (None, 1)                 0
=================================================================
Total params: 162,579
Trainable params: 162,579
Non-trainable params: 0
_________________________________________________________________
```

And then, you can use the loaded model for training/predict as you want.

3.2 Modeling with other machine learning method

Since we got a matrix, it is possible for using this matrix for many training works other than deep learning. The result is available for comparison or make some further analysis.

Here we provided a brief sample for using random forest to modeling and predicting.

```
In [28]: from sklearn.ensemble import RandomForestClassifier

         #init
         rf = RandomForestClassifier(n_estimators=10, max_depth=None,min_samples_split=2, bootst

         #training
         rf.fit(trainDataMat, trainLabelArr)

         #predicting
         rfPrediction = rf.predict(testDataMat)
```

confusion matrix and related metrices

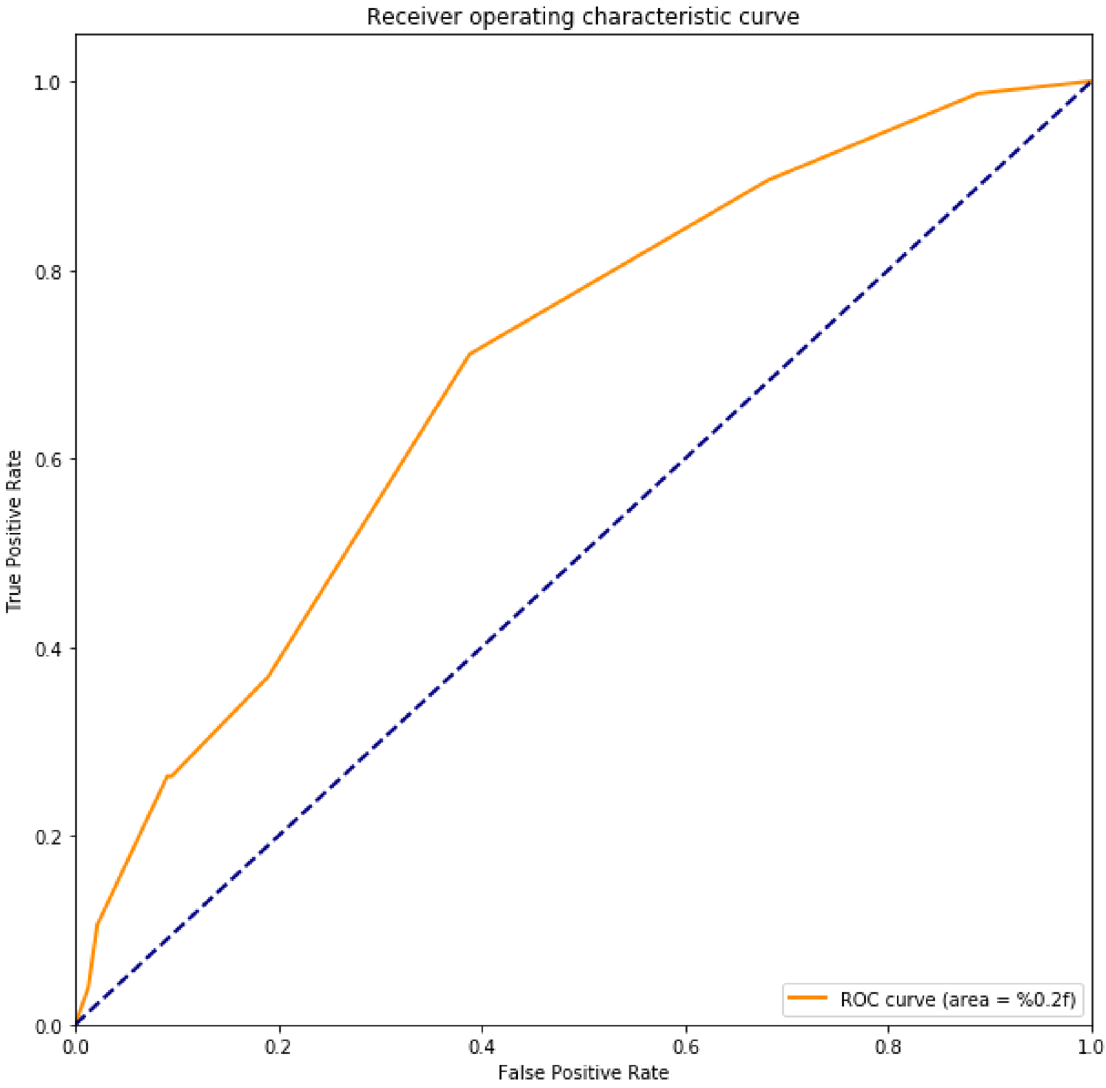
```
In [29]: cm=confusion_matrix(testLabelArr,rfPrediction)
         print(cm)
         print("ACC: %f "%accuracy_score(testLabelArr,rfPrediction))
         print("F1: %f "%f1_score(testLabelArr,rfPrediction))
         print("Recall: %f "%recall_score(testLabelArr,rfPrediction))
         print("Pre: %f "%precision_score(testLabelArr,rfPrediction))
         print("MCC: %f "%matthews_corrcoef(testLabelArr,rfPrediction))
         print("AUC: %f "%roc_auc_score(testLabelArr,rfPrediction))
```

```
[[227   5]
 [ 68   8]]
ACC: 0.892857
F1: 0.792453
Recall: 0.828947
Pre: 0.759036
MCC: 0.721701
AUC: 0.871370
```

ROC curve

```
In [30]: rfPredictedProbability = np.array(rf.predict_proba(testDataMat))
         analysisPlot.plotROC(testLabelArr,rfPredictedProbability[:,1],showFig=True,savePath=Non
```

```
<Figure size 432x288 with 0 Axes>
```



4. Conclusion

In this notebook, we introduced how to use autoBioSeqpy for file reading and engaging it into a research workflow. We hope this notebook could help users to understand the basic way for using it for data transferring and evaluating the modeling result. Then users could use it as a part of their own researches.

We are looking forward to receive any feedback and suggesting. If you have any problem in using this tool, please do not hesitate to connect us, thanks.