Introduction

The functions of the code are sequence data collecting,sequence data conversion and the construction and validation of the neural network model. All of the code is written in python and should work with version 2.7. Some parts of the code require matplotlib, numpy, scipy, and sklearn. We have packed the code as a *structural_protein_package.py* file, and you can use the following statement to import the package:

```
from structural_protein_package import *
```

All the data we used are stored in the *Task1 dataset* folder.

Sequence Data

All sequence data were obtained from PLOS, supporting information, Dataset S3 ~ S5. We downloaded the xls and xlsx files. Then to get the GI information of the protein in the Excel files, you can call the following function in our package to convert the Excel files to .txt files:

```
ConvertXlsToGIFIle(XlsFilename='Dataset_S5.xls')
```

The *ConvertXlsToGIFIle* function will read data from the Excel file and output the GI information to .txt file. The parameter of the function is the name of Excel file and the output files are *posSetGI.txt* and *negSetGI.txt* ,which storing the GI information of two kinds of protein.

The next step is to get sequence data based on the GI information that we have obtained in the previous step. The following function is designed to search the blast database for protein sequences based on GI information in the files.

```
convertGItoSeq(posGIFilename="posSetGI.txt",negGIFilename="negSetGI.txt")
```

posGIFilename and *negGIFilename* are the name of the GI information files. The function will read the GI information in the two files and then query the protein sequence data in the blast database. These protein sequence data will be output to *posSet.txt* and *negSet.txt*.

Through the above two steps, we have obtained the sequence data, which is stored in *posSet.txt* and *negSet.txt*.

Converting Protein Sequences to Neural Network Data Sets

We have written a function to convert protein sequences to neural network data sets. You can use the following statement to call the function, which can convert a protein sequence to a vector.

```

Vec,Y,posTestset,negTestset=data_processing(posSetFilename="posSet.txt",negSetFilename="negSet.t
xt")
```

The two parameters of the function is the name of files that store the protein sequences. The function will return four objects, *Vec*, *Y*, *posTestset* and *negTestset*. Here are the detailed descriptions of these 4 objects.

- Vec : A python list, which is composed of vectors converted from the protein sequences.
- Y: A python list, which is composed of labels of the vectors in Vec. The value of label can only be 0 or 1.
- posTestset : A python list. This is the test set composed of vectors converted from positive protein sequences and theirs labels.
- negTestset : A python list. This is the test set composed of vectors converted from negative protein sequences and theirs labels.

Our training set is composed of *Vec* and *Y*. Our test set is composed of *posTestset* and *negTestset*.

Model construction and validation

Model construction

We can call the *classify* function to construct our neural network model.

```
clf = classify(Vec,Y)
```

The parameters of the function is similar to return value of *data_processing* function. *Vec* is a python list,which is composed of vectors converted from the protein sequences and is a part of out training set. *Y* is also a python list, which is composed of labels of the vectors in *Vec*. *Vec* and *Y* made up our training set. The function will return a classifier which is trained with the training set composed of *Vec* and *Y*.

We can use the classifier that we construct to predict the kind of a new protein. The *predict* function are used to predict.

```
predict(clf,predictsetFilename='predictset.txt')
```

The first parameter of the function is a classifier that we construct in the previous step. The second parameter of the function is the name of the file that stores the sequences that need to be classified. The prediction result will be output to the *predictresult.txt* file.

Model validation

We can call the *testing* function to test our neural network model.

```
posPre=clf.predict(posTestset)
negPre=clf.predict(negTestset)
xx,yy,ACC,MCC=testing(posPre,negPre)
```

The fist parameter of the *testing* function is *posPre*. *posPre* is the prediction result of *posTestset*. Similarly, *negPre* is the prediction result of *negTestset*. The definition of *posTestset* and *negTestset* are as described above. *clf* is the model to be tested. The return values of *testing* function are *xx*, *yy*, *ACC* and *MCC*. Here are the detailed descriptions of these 4 objects.

- xx: A python list, which stored the false positive rate(FPR) of the model.

- yy: A python list, which stored the True positive rate(TPR) of the model. The elements in xx and yy correspond one by one.
- ACC: A float number. The accuracy of the model.
- MCC:A float number. The Matthews correlation coefficient of the model.

  You can draw ROC curve using the *plot* function in the package of matplotlib. Here is an example.

  ```
  plt.plot(xx,yy) #draw ROC curve
  plt.legend()
  plt.show()
  ```

  If you want to do k-fold cross validation, you can call the following function. Here is an example.

```
ACC = k_fold_Cross_Validation_classifier(X,Y,k)
```

X is a python list, which stores the vectors in our data set. Y is also a python list, which stores the labels of the vectors in X. k is the parameter of k-fold cross validation. The function will return a float number, which is the accuracy of the k-fold cross validation of our model that is trained by the data set composed of X and Y.

You can also use the *get_k_fold_Cross_Validation_classifier* function to do k-fold cross validation. Here is an example.

```
clfs,testset=get_k_fold_Cross_Validation_classifier(X,Y,k)
```

The parameters of the function are the same with those of the *k_fold_Cross_Validation_classifier* function. The return values are *clfs* and *testset*. *clfs* is a set of classifiers that were built in k-fold cross validation. *testset* is a set of test set that every classifier in k-fold cross validation uses. The size of *clfs* and *testset* is k.

There are also some functions in the package that can be used in model validation. The *calcACC* function are used to test the accuracy of a model in a given test set. The *calcFPR_TPR* function are used to calculate the Matthews correlation coefficient of a model and FPR-TPR in a given test set.