==================================

DeepRT v1.0 software mannual

==================================

DeepRT is a deep learning based software for peptide retention time prediction in LS-MS/MS based proteomics and achieves the state-of-the-art performance. And this is a brief user mannual for it. If you have any question using the software, just feel free to contact machunwei@genomics.cn or create an issue at https://github.com/horsepurve/DeepRT.

<1> The structure of DeepRT v1.0 binary backage:

The DeepRT v1.0 binary backage consists of 9 dynamic link libraries:

```
conv.so
datalayer.so
DeepRT.so
Load_data_validate.so
ModelConv.so
my_lstm.so
my_pca.so
norm.so
validate.so
```

and a python main program:

```
main.py
```

While you have insalled the dependent libraries, just type:

```
python main.py
```

to run the software. You can also specify your own parameters in main.py.

<2> Installation:

Step 1: download Intel MKL (Math Kernel Library) via:
https://software.intel.com/en-us/qualify-for-free-software, and install.

Step 2: install python2 through miniconda via:
http://conda.pydata.org/miniconda.html

Step 3: install deep learning libraries:

```
conda install anaconda-client
conda install -c jaikumarm theano=0.9.0.dev1
conda install -c jaikumarm keras=1.0.6
```

Step 4: configure Theano, add following to ~/.theanorc:

```
[blas]
ldflags = -L/PathTo/Intel_MKL/intel/mkl/lib/intel64
[global]
floatX = float32
```

<3> Datasets:

The example datasets can be downloaded at:
https://github.com/statisticalbiotechnology/GPTime/tree/master/Data
which were also been used in Elude and GPTime software. For peptides without modifications, just

leave them intact, but for peptides with modificaitons, substitute the modified amino acids with characters unused. For example, in mod.txt, we have replaced the modified amino acids with numbers:
'M[16]' -> '1',
'S[80]' -> '2',
'T[80]' -> '3',
'Y[80]' -> '4'.

<4> How to run step by step:

Step 1. import the package:

```
from DeepRT import *
```

Step 2. split the data:

```
split_dataset(dataset = 'mod.txt',
              train_ratio = 8,
              validate_ratio = 1,
              test_ratio = 1,
              seed = 42)
```

# It splits data.txt into training/validate/test sets as 8:1:1 ratio by default, and then moves input files into .\input\ and test labels into .\results\. Users can change the split ratio and seed for randomly split using parameters.

Step 3. CNN training:

```
cnn_train(learning_rate = 0.01,
            k_size = 3,
            drop_value = 0.2,
            Layers = "20,20,20,20",
            batch_size = 128,
            Flag = 1,
            log = 'log/cnn_3_20_20_20_20_0.2.txt')
```

# This step performs the CNN training with parameters:
[learning_rate]: initial learning rate set into model
[k_size]: CNN filter size
[drop_value]: dropout ratio in dropout layers
[Layers]: "20,20,20,20" as example, specifies 4 CNN layers in the model and each layer uses 20 channels
[batch_size]: batch size for SGD
[Flag]: set Flag = 1 will convert raw data into python pkl format with coding, set this to 1 in your first CNN model and to 0 otherwise
[log]: write the training outputs into log file

Step 4. CNN feature extraction:

```
CNN_feature_extractor()
```

# This step extracts features from the trained models, and puts the generated features into .\features\

Step 5. LSTM training:

```
LSTM_training(batch_size = 128,
                layer_num = 1,
```

active = 'linear')

# This step performs the CNN training with parameters:

[batch_size]: batch size of data fed into model

[layer_num]: specifies LSTM layer number

[active]: specifies the regression layer, "linear" for linear, "sigmoid" for logistic regression

Step 6. LSTM feature extraction:

LSTM_feature_extractor()

# This step extracts features from the trained models, and puts the generated features into .\features\

Step 7. Feature PCA

PCA(0.95)

# PCA_Ensemble.py accepts [components] as parameters

[components]: select the number of components such that the amount of variance that needs to be explained is greater than the percentage specified, 0< components <1, 0.95 as default.

Step 8. SVR_regression(gamma = 1e-9, C = 1e6)

# This step performs the SVR regression with [gamma] [C] as parameters

[gamma]: SVR Kernel coefficient

[C]: Penalty parameter C

Step 9. RF_regression(n_estimators = 400)

# RF_regression accepts [n_estimators] as parameters

[n_estimators]: the number of trees in the forest

Step 10. GB_regression(n_estimators = 1000)

# GB_regression [n_estimators] as parameters

[n_estimators]: The number of boosting stages to perform.

Step 11. bagging()

# This step performs the model bagging.

Step 12. Evaluation()

# This step gives some evluations of the results.