Documentation of LibHIR

LibHIR is an open source library implementing the HIR model to provide a collaborative prediction interactive representation of multi-entities and context-aware scenarios for later tasks, i.e., regression, classification, ranking and so on. This This document explains the use of LibHIR.
You can get LibHIR at https://github.com/TagineerDai/LibHIR.
Please read the COPYRIGHT file before using or modifying LibHIR.

Table of Contents

Quick Start

If you are new to HIR and if the task is classification and regression, please try our command-line tool version of LibHIR. We offer a script `prepro.py` to transfer the svm+ data into HIR format data file and initialize the config and model file. Generally, SVM+ data is to concate the context embedding feature data after svm format records. For more information, view the Data Format and Transfer section.

Usage:

```
prepro.py --rawfile filename.txt
```

The preprocessing script generates four files, including `model.txt`, `config.txt`, `train.txt` and `test.txt`, which should be copied to the same direct as the program `HIRTrain` and `HIRTest` on Linux, or `HIRTrain.exe` and `HIRTest.exe` on Windows. Run the files with option `-h` and no parameter, to show the usage of them. For further information, view the Command-line Tool Usage section.

Usage:

```
HIRTrain/HIRTest [option parameter]
```

Installation

Linux

On Linux systems, type `make` in the terminal to build the `HIRTrain` and `HIRTest`, add `HIRTrain` and `HIRTest` after the `make` to build the given file.

To uninstall LibHIR, use the command `make clean`.

Windows and other platform

The executable file `HIRTrain.exe` and `HIRTest.exe` of the given command-line tool is offered. Once you are developing your own LibHIR based program, consult `Makefile` for linux platform to manually build the project, or simply use a IDE.

Data Format and Transfer

Under the `data` direction, we offered a sample python code `transformat.py` to build up the svm+ format data for our movielens sample. And `prepro.py` to transfer the svm+ format raw data into HIR format data, and preprocess by generate config file model file. View the Command-line tool usage for more detail of its usage.

SVM+ format

The HIR model is compatible to multi-entity data with context featrue and entity features. The record format of libsvm concatenated with all the context feature, i.e., the svm+ format, is the raw data input of the `prepro.py`.

The format of raw svm+ data file is:

<label> <index$_1$>:<value$_1$> ... <index$_k$>:<value$_k$> <feature$_1$> ... <feature$_k$>
.
.
.

Each line is a record instance and is ended by a '\n' character. The target value is a real value number target in regression task, or a integer of the class label in classification task. In the pair, the entity index are integer, and could be in order or after shuffled. The feature values will be saved and generated as float. Once you have more than one entities with external features and context feature of a record, concatenate them as one feature vector. It will be treated as a special feature vector in an arbitrary dimension with its value not be modified by the learning process.

HIR format

The preprocess script `prepro.py` will load in the raw svm+ format file and generate record file, model file, and config file. Use the option `-h` or `--help` to view the usage of the options and parameters.

The config file `config.txt` contains three set of parameters, the filenames includes the model file, training data and testing data file, the data specification parameters are for the HIR model and record data, and the size of parameter are index of the entity feature. The config format is `[param name] = [param value]`, and the line comment in the config is started with a `#`.

It is not suggested that you to modify the parameter in config.txt directly. All the specification of HIR model and HIR record should are generate with the config file by preprocess script. The miss matching will lead to fatal error.

The model.txt contains the item feature of each entities, the parameter of the HIR model component, such as the tensor layer, the hidden layer, the latent representations, and the target representation. If you are developing your own task with HIR representation, modifing the `prepro.py` and the `HIRModel::ModelLoading` and `HIRModel::ModelSaving` could help.

The record file, namely the `train.txt` and the `text.txt` are the data unrelated to the HIR model. It is consist with record number, item index list, feature vector list, target value list and predict value list.

User Guide

The command-line tool offers implementation of the basic regression and classification task based on the HIR representation.

Transformat Script Usage

Once you have python and numpy installed, use the command `python prepro.py [option]` to preprocess the raw svm+ format data. Usage of the options is as follows.

Option	Detail
--rawfile	the svm+ format file name.
--Rtrain	the ratio of split. Rtrain = training : total record
--category	the type of task. 1--regression 2--biClass 3..n--multiClass
--hlayer	number of the hidden layer
--dim	demension of the features, and representations.

The Rtrain is a float in (0,1). And the category, hlayer and dim are integers.

HIRTrain usage

Sample:

```
HIRTrain -r 100 -o 1 -a 0.001 -e 5
```

In the HIRTrain program, the HIR model is specified by the config file `config.txt` and trained by the data in `train.txt`. The loss will showed every epoch, and the model after training will be saved back in `model.txt`.

Option	Detail	Value
-r	epoch number	positive integer
-o	objective function	0(ABS), 1(RMSE), 2(LL), 3(GOLD)
-e	evaluation function	4(ABS), 5(RMSE), 6(AUC), 7(ACC)
-a	learning rate	float number in (0,1)
-m	learning method	0(SGD)
-h	help	the help information

The epoch is the repeating training time for every record in the `train.txt`. The learning method is Stochastic Gradient Descent. The learning rate(alpha) is a float, which represent the update ratio in the backpropagation process. Both the regression and classification task have its related objective function or evaluation, therefore missmatch task and option will cause error.

HIRTest usage

Sample:

```
HIRTest -e 5
```

In the HIRTest program, the luation function is related to choosing task, a compatible pair of task and evaluation function will cause error. The predict record is saved in a given file.

Options	Details	Value
-e	evaluation function	4(ABS), 5(RMSE), 6(AUC), 7(ACC)
-f	predict record file	filename stirng
-h	help	help information

For regression tasks, the prediction of every record is a double value in one line. For classification tasks, the prediction of every record is one list of value with its length the same as the category number. The largest value in one line indicates the predicted class.

Developer Guide

The network components are declared in the header file `HIR.h`. You need to include this file in your C++ source files and link your program with all the C++ source codes, with the guide of the Makefile. You could take the sample code `HIRTrain.cpp` and `HIRTest.cpp` as a reference on how to use them.

The config file contains model specification hyperparameters such as sizes or record numbers.

```
outconfig(filename)
```

If there are some specification parameters ofyour customer components should be included, modify this function to add this to the config file.

```
HIRConfig::HIRConfig()
```

If there are addition parameters in the config file, add declare in the HIRConfig class, and add parser.get() in the construct function of HIRConfig. The compatible config value types are integer, double, string and vector.

```
void get<int>(const std::string &key, int &value); void get<double>(const std::string &key, double &value); void get<std::string>(const std::string &key, &std::string &value); void get<int>(const std::string &key, std::vector<int> &value);
```

sample:

```
parser.get("matdim", dim);
```

After the modification of the former funcitions and including the `HIR.h`, you could use the static HIRConfig object cfg declared in `HIR.h`. Firstly you have to initialize it by:

```
cfg = HIRConfig();
```

Note that the config file `config.txt` should be in the same direction of the executable file of your program.

The HIR model parameters have been initialized in the preprocess step. If you want to add new components after the target representation of HIR, modify the prepro.py and HIRModel in core HIR library. It could help load, save, modify and clear the component at the same time of the HIR components.

The function `outmodel(filename)` write the model component parameters into the file.

```
int ModelLoading(HIRConfig cfg); int ModelSaving(HIRconfig cfg); int GradInit(HIRConfig cfg); int GradClear(HIRConfig cfg);
```

These functions will load the model component parameters from the file writen by the outmodel function in preprocess, and write it back from the HIRModel object, with the specify parameters of the HIRConfig. The GradInit and GradClear function deals with the derivation intermediate amount, respectively initialize the position and set their value to all zeros.

After the modification to add your own component into the HIRModel related parts and include the `HIR.h`, you could use the static object HIRModel HIR decleard in the `HIR.h`. And initialize it by:

```
HIR = HIRModel(cfg); HIR.GradInit(cfg);
```

Note that the model file `model.txt` should be under the same direction of the executable file of your program.

You could use the arguments defined in HIR or add your own options and define your own parameters. The related functions are listed as follows.

```
HIRTrainParam parse_train_argument(int argc, char** argv, int task);
```

The task param is related to the regression and classification mission choice. The HIRTrainParam and HIRTestParam in the file are structs, therefore you could define your own parameter struct and declare function with return value of it, and parse the argument of your program by:

```
HIRTrainParam para_train; para_train = parse_train_argument(argc, argv, cfg.task);
```

The parameter could be used by:

```
int evaluation_type = para_train.evl;
```

The regression task sample HIRTrain.cpp and HIRTest.cpp is a sample code of your own task program. The HIRtrain contains the model training process, and the HIRTest contains the predict and evaluation part of the task. You could modify this file or use it as a hint of detail in your own customerized task.

In the HIRTrain we used Stochasic Gradient Descent as learning method, the task related part related model modification is in the following functions:

```
int forward_train(int record_index, HIRConfig); int backward_train(int record_index, HIRConfig);
```

In the forward or backward functions, the parameter of the model component before the target representation, in other word, the HIR components, are modified by calling the function below:

```
int GradForward(HIRConfig cfg, int* index_list, double* feature); int GradBackward(HIRConfig cfg, HIRTrainParam para, int* index_list);
```

Considering the data flow order, you should call the GradForward function at the start of the forward_train function, then use the last item of the rlist(the target representation) to feed forward and get the predict value, or in the other word, the hat label. Samely, in the backward_train function, the backpropagation process of the addition components should be finished first. Then the last L_r component could be used by the HIR model. The backpropagation and modification of HIR related component will be finished in the function GradBackward.

The trained model should be save back in the model file `model.txt` by calling the function as the following code:

```
HIR.ModelSaving(cfg);
```

Then in the next evaluation and test process, it could be reload in with the same method of the training process.

The offered evaluation function and loss function are respectively compatible with regression task or classification task. The given choices in the HIR library is listed as follows: `enum { ABS_L, RMSE_L, LL_L, GOLD_L, HINGE_L, ABS_E, RMSE_E, LL_E, ACC_E, AUC_E };`

If you are going to use evaluation function which is not in the list, try to modify the function below and call it after the prediction and saving the predict answer into the hat_y member of the

HIRRecord.

```
int rec, train; std::string filename; std::vector&lt;int*> ilist; // index per record
std::vector&lt;double*> flist; // feature per record double** ylist; // target value
for classification double** hat_y; // predict value of the model
```

```
double evaluate(int task, int rnum, double** y, double** y_hat);
```

In the HIRTest program, we give out a sample about how to carry out a regression task. The static object HIRRecord test of testing data, and the argument structure HIRTestParam para_test should be declared. Once you want to carry out your own task, the function should be modified:

```
forward_test(int index, HIRConfig cfg);
```

The index, the same as in the HIRTrain.cpp, is the index of record in the test record. In the function, you should obey the feed-forward data flow order, and call the following function first:

```
int GradForward(HIRConfig cfg, int* index_list, double* feature)
```

Then the last item in the HIR.rlist vector is the HIR target representation generated by the HIR model, with which you could calculate the next coming component values, and got the output of your own special task.

To save the predict answer, this function could be used:

```
void predictSave(std::string pfname, HIRConfig cfg);
```

If you have designed predict answer or task, which is not a float list for every record to be print out, and want to save it down, modifing this function helps printing given label for different class label lists, doing ranking and so on.