FCUBE: A Platform for Collaborative Learning Documentation

I. Factoring Options

Factoring is what we call the *generation* of multiple learners each with its own subset of data, explanatory variables, and parameters. This process is governed by the *parameters options* (see Listing 1 for an example).

Data factoring: The *parameters options* file provides the path to the remote data storage directory (NFS or S3 bucket name) where the training data is stored. FCUBE's factoring service randomly picks one training split and copies it to the instance local disk. Then a stochastic sampling process takes place where both exemplars and explanatory variables are factored according according to the *data_sample_rate* and *variable_sample_rate* parameters. The first specifies the ratio of exemplars that will be sampled from the data, while the second specifies the ratio of the explanatory variables. It is important to note that we keep track of the variables used for training in each of the instances (see Listing 2). This information will be necessary later to evaluate the generated models on unseen data.

Parameter factoring: The user decides what *parameters* will be factored (parameters for which a value will be sampled stochastically from the possible ranges/choices) and what *parameters* will be set to their default value. As mentioned in the previous section each FCUBE instance has built-in factoring functionality that will parse the *parameters options* file and generate a configuration (properties) file for the learner during the deployment phase (see Listing 3).

A. Parameters option syntax

Fixed parameters

First, we need to specify the values of fixed arguments (such as the number of CPU threads). Note that the path to the dataset must be specified as a fixed parameter.

Fixed parameters: The syntax employed to declare fixed parameters is:

fixed param_name value

Parameters Options Specification

The following aspects are declared 1) Name 2) a type, 3) a default value, and 4) a discrete set, range, or subset of possible values.

Types: F^3 supports 4 basic types: **int, float, bool, string**

Default values: The default value will be used if the parameter is not factored. The following syntax has to be added after the type of the parameter:

default (value **)**

Discrete sets: The instruction **discreteSet** is used to specify a set of discrete options for a given parameter. The following notation must be used:

par_name type **discreteSet default**(optiond){opt$_1$**;**...**;**opt$_n$}

Ranges: This option only applies for the numerical types **int** and **float**. A range is characterized by a minimum value, a step value (difference between consecutive numbers of the set), and a maximum value. F^3 syntax does not provide explicit support for continuous values. However, an easy approximation can be obtained by setting a small step. To specify a range of values for a given parameter, the following notation must be used:

par_name type **range default**(val)**[**min**:**step**:**max**]**

Subsets of different sizes: This instruction allows to select a subset of a varying size from a set of items. For instance, this instruction is helpful to select a subset of explanatory variables. The original set from which we will sample is specified with *par_name* while the second parameter specifies the size of the subset. To specify a random subset of varying size, the following notation must be used:

par_name type **subset default**(1){val$_1$**;**...**;** val$_n$}

where the values between brackets correspond to percentages $(0 < val_i \leq 1)$.

Specifying Factored parameters

Finally, we need to list the the parameters that will be factored, i.e. the parameters that will be explored in the cloud runs.

Factored parameters: The following syntax is employed to set the parameters that will be factored:

factoredParams{par_name$_1$**,**...**,**param_name$_N$}

B. Parameters Options Examples

In this section we show an example of *parameters options* and the two configuration files generated with FCUBE factoring service. Let is consider a dataset composed of 20 features for this example.

```
fixed data fcube/higgs/train/
fixed threads 2

data_sample_rate float discreteSet default ( 1 ) { 0.1 ; 0.2 }
variable_sample_rate float discreteSet default ( 1 ) { 0.25 ; 0.75 ; 1 }
false_negative_weight float range default ( 0.5 ) [ 0.4 : 0.05 : 0.6 ]
xover_op string discreteSet default ( SPUCrossover ) { SPUCrossover ; KozaCrossover }
pop_size int discreteSet default ( 1000 ) { 1000 ; 1500 ; 2000 }

factoredParams {data_sample_rate, variable_sample_rate, xover_op, pop_size}
```

Listing 1: Example of parameters option file sent to all the FCUBE instances deployed within a FCUBE run. The path to the data and the number of threads are declared as fixed parameters. The built-in parameters for data management (*data_sample_rate* and *variable_sample_rate*) as well as the crossover operator and population size are all assigned a discrete set of choices. The learner-specific parameter indicating false negative weight is assigned a range of possible values. Finally, the instruction in the last line indicates the parameters that will be factored (stochastically selecting a value from the possible choices). Only the false negative weight will be set to its default value.

```
data_split=split_5.csv
data_sample_rate=0.1
variable_sample_rate=0.25
sampled_variables=X0 X4 X9 X11 X16
```

Listing 2: Example of a data configuration file generated with FCUBE factoring service. Note that due to the stochastic nature of the process, different FCUBE instances will generate different configuration files. This file will be used in the later process of filtering and fusion and allows to keep track of the split and variables used for learning in each FCUBE instance.

```
xover_op=SPUCrossover
threads=2
false_negative_weight=0.5
pop_size=1000
```

Listing 3: Example of a Java properties file generated with FCUBE factoring service. Note that due to the stochastic nature of the process, different FCUBE instances will generate different configuration files. This file will be an input argument for the learner deployed in the cloud

II. LEARNER TEMPLATE AND EXAMPLES

```
USAGE:

TRAIN:
java -jar gpfunction.jar -train path_to_data -minutes min -properties path_to_properties

OBTAIN PREDICTIONS:
java -jar gpfunction.jar -predict path_to_test_data -model path_to_model -o path_to_predictions


EXAMPLES:

TRAIN:
java -jar gpfunction.jar -train higgs_2.csv -minutes 60 -properties parameters.properties

OBTAIN PREDICTIONS:
java -jar gpfunction.jar -predict higgs_10.csv -model model.txt -o predictions.csv
```

Listing 4: GPFunction learner interface and examples of use. GPFunction is one of the learners integrated in FCUBE framework. In this case, the learner is packaged in a Java executable file. FCUBE supports also Python, C, and C++ binaries as long as the provided interface is implemented.