Online Multi-Task Learning Toolkit (OMT) v1.0

Xu Sun (xusun@pku.edu.cn)
School of EECS, Peking University
http://klcl.pku.edu.cn/member/sunxu/index.htm

1. Overview

This is a general purpose software for online multi-task learning. The online multi-task learning is mainly based on Conditional Random Fields (CRF) model and Stochastic Gradient Descent (SGD) training. The work is described in (Sun et al., 2013a).

Main features:

- Developed with C#
- High accuracy on the human activity recognition tasks (Sun et al., 2013a)
- General purpose (it is task-independent & trainable using your own tagged corpus)
- Support SGD training & Limited-memory BFGS training
- Support various evaluation metrics, including token-accuracy, string-accuracy, & F-score

2. Installation

Need C# compiler, e.g., *VisualStudio.net* or *Mono*

3. Format of Data Files

The sample train/test files are given for illustrating the format of data files. The sample train/test files are extracted from a sensor-based human activity recognition task.

- `(n)ftrain.txt` ⟶ task-n's feature file for training (note: n should start from 0)
- `(n)gtrain.txt` ⟶ task-n's gold-standard tagging file for training
- `(n)ftest.txt` ⟶ task-n's feature file for testing
- `(n)gtest.txt` ⟶ task-n's gold-standard tagging file for testing (essentially it is not required, it is only for evaluation of the model accuracy)

The training files (*ftrain.txt* and *gtrain.txt*) should follow a specially defined format for the tool to work properly.

ftrain.txt includes the total-#feature-information and the detailed features of each training instance. Take the sample file *0ftrain.txt* for example, the 1st line "11016" of the file is the total-#feature-information, and it means 11016 features in total. There should be boundaries between the total-#feature-info and training instances, and among different training instances. A boundary is expressed by a blank-line. A training instance has multiple lines of features. A feature (e.g., "the current word is *cat*") is expressed by an index (e.g., "532") with the index started from 0. The 1st line of features corresponds to the 1st token (e.g., a word in a sentence or a signal in a signal sequence), the 2nd line of features corresponds to the 2nd token in the sequence, and so on. For each line, the features/indices are sorted incrementally.

gtrain.txt includes the total-#tag-information and the detailed gold-standard tags. In *0gtrain.txt*, the 1st line "5" is the total-#tag-information, and it means this task has 5 tags in total. Also, a boundary is expressed by a blank-line. A tag sequence (expressed by a line) has multiple tags. A tag (e.g., "Beginning of a chunk") is expressed by an index (e.g., "0") with the index started from 0. In a line, the 1st tag corresponds to the 1st token, and 2nd tag corresponds to the 2nd token, and so on.

The test files, *ftest.txt* and *gtest.txt*, have the same format like the training files.

4. How to Use

You can build a model based on your own tagged data of a task. The only thing need to do is to provide the properly formatted tagged files. Use the command "`option1:value1 option2:value2 ...`" for setting values of options (hyper-parameters). The command "`help`" shows help information on command format. Below is the options and values:

- 'm' ⟶ setting *Global.runMode*
  Optional values:
  mt.train (OMT training);
  mt.train.fast (fast OMT training with probabilistic sampling (Sun et al., 2013a));
  mt.test1 (standard OMT testing);
  mt.test2 (OMT testing for a new task by choosing the most similar model);
  mt.test3 (OMT testing for a new task by using a voted model, i.e., the OMT-SBD method described in Section 4.4 of (Sun et al., 2013a))
  Default: *mt.train.fast*

- 'o' ⟶ setting *Global.optim*
  Optional values:
  sgd (SGD training with fast regularization (Sun et al., 2013b))
  sgder (SGD training with exact regularization)
  bfgs (Limited-Memory BFGS training. Note: this is only for *Single* and *Merge* baselines for experimental comparisons)
  Default: *sgd*

- 'a' ⟶ setting *Global.rate0*
  Optional values:
  a real-value (set the initial value of the learning rate γ)
  Default: *0.1*

- ‘r’ ⟶ setting *Global.regList*
  Optional values:
  one or multiple real-values (e.g., “`r:1`” for setting regularizer as 1.0; “`r:1,5,10`” for multiple rounds of training with the regularizer of 1.0, 5.0, and 10.0, respectively)
  Default: *1*

- ‘d’ ⟶ setting *Global.random*
  Optional values:
  0 (all weights are initialized with 0)
  1 (random initialization of weights)
  Default: *0*

- ‘e’ ⟶ setting *Global.evalMetric*
  Optional values:
  tok.acc (evaluation metric is token-accuracy)
  str.acc (evaluation metric string-accuracy)
  f1 (evaluation metric F1-score, i.e., balanced F-score)
  Default: *tok.acc*

- ‘t’ ⟶ setting *Global.taskBasedChunkInfo*
  Optional values:
  np.chunk (set task-based-chunk-information as *NP-chunking-task*. This option is for calculating F-score because F-score is task-dependent. This option is useless when the evaluation metric is not F-score. You can also set other specific task-based-chunk-information. In this case you should re-define the *getChunkTagMap()* function.)
  bio.ner (for Bio-NER-task)
  Default: *np.chunk*

- ‘ss’ ⟶ setting *Global.trainSizeScale*
  Optional values:
  a real value (this is for scaling training data. For example, can set as 0.1 to use 10% training data for experiments. The default value 1 means 100% of training data)
  Default: *1*

- ‘i’ ⟶ setting *Global.ttlIter*
  Optional values:
  an integral value (the total number of training iterations)
  Default: *100*

- ‘s’ ⟶ setting *Global.save*
  Optional values:
  1 (model weights will be saved as model.txt file when training ends)
  0 (no save of model)
  Default: *1*

- ‘of’ ⟶ setting *Global.outFolder*
  Optional values:

a string (setting the folder name for storing the output files)
Default: *out*

Other global variables can be set directly via revising the *A.Global.cs* file, if necessary:

- *Global.mt_singleTrain* ⟶ for turning on/off "*Single*" baseline in (Sun et al., 2013a)
- *Global.mt_mergeTrain* ⟶ for turning on/off the "*Merge*" baseline in (Sun et al., 2013a)
- *Global.mt_mtTrain* ⟶ for turning on/off the "*OMT*" method in (Sun et al., 2013a)
- *Global.nTask* ⟶ set the number of tasks
- *Global.cFactors* ⟶ set the C value of OMT in (Sun et al., 2013a)
- *Global.simiMode* ⟶ set the similarity kernel of OMT in (Sun et al., 2013a)
- *Global.sampleFactor* ⟶ set the probabilistic sampling factor of OMT in (Sun et al., 2013a)

4.1 How to Train the Model

Command examples:

- `./run.exe m:mt.train.fast o:sgd a:0.05 r:5 e:str.acc i:50`
  It means: use the fast OMT training; training algorithm is *SGD*; initial value of the learning rate γ is 0.05; the regularizer value is 5.0; evaluation metric is string-accuracy; total number of training iteration is 50.

4.2 How to Evaluate on Test Data

Evaluation on the test data is simpler than training, because there are less hyper-parameters to set. Command examples:

- `./run.exe m:mt.test1`
  It means: use the test1 mode (with default settings of hyper-parameters).
- `./run.exe m:mt.test3 e:str.acc of:out.test`
  It means: use the test3 mode; evaluation metric is string-accuracy; output folder is *out.test*.

5. About Output Files

- *./out/trainLog.txt* ⟶ recording detailed training information of each iteration.
- *./out/rawResult.txt* ⟶ recording the evaluation-score-on-test-data, time-cost, objective-function-value, etc. of each training iteration. This file has 2 formats available, including a matrix format.

- *./out/summarizeResult.txt* ⟶ to automatically summarize the results in *rawResult.txt*, e.g., computing averaged evaluation scores and standard deviations of multiple runs of training or n-fold CV, etc.

- *./out/(n)taskOutput.txt* ⟶ the tags predicted from the test data for task-n.

- *./model/(n)model.txt* ⟶ the model file derived from training for task-n.

6. About Code Files

- `A.Global.cs` ⟶ This file has the definitions and values of global variables. Most hyper-parameters are stored here.

- `A.Main.cs` ⟶ Main() function

- `Base.**.cs` ⟶ These files defines the basic data structures (e.g., hashmap, matrix) and general algorithms (e.g., Viterbi decoding)

- `Dataset.cs` ⟶ For storing and processing data (feature files and tag files) in CRF-ADF

- `FeatureGenerator.cs` ⟶ For generating features

- `Gradient.cs` ⟶ For computing CRF gradient, which is useful in training

- `Inference.cs` ⟶ For CRF inference & decoding

- `Model.cs` ⟶ For reading & writing CRF model

- `ToolboxTrainTest.cs` ⟶ For high level functions of training & testing

- `Optim.BatchLBFGS.cs` ⟶ For detailed implementation of the LBFGS batch training method

- `Optim.Stochastic.cs` ⟶ For detailed implementation of the SGD online training methods

7. Code Update History

Dec. 17 2013 ⟶ version 1.0

References

Xu Sun, Hisashi Kashima, and Naonori Ueda. Large-scale personalized human activity recognition using online multitask learning. *IEEE Trans. Knowl. Data Eng.*, 25(11): 2551–2563, 2013a.

Xu Sun, Yao zhong Zhang, Takuya Matsuzaki, Yoshimasa Tsuruoka, and Jun'ichi Tsujii. Probabilistic chinese word segmentation with non-local information and stochastic training. *Inf. Process. Manage.*, 49(3):626–636, 2013b.