CRF-ADF Sequential Tagging Toolkit 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 sequential tagging (or called sequential labelling, linear-chain structured classification). The CRF (Conditional Random Fields) model is described in (Lafferty et al., 2001) and the ADF (Adaptive stochastic gradient Decent based on Feature-frequency information) fast training algorithm is described in (Sun et al., 2012).

Main features:

- Developed with C#
- High accuracy (72.3% on *Bio-Entity Recognition Task at BioNLP/NLPBA 2004*, and 97.5% on *Chinese Word Segmentation MSR Task*)
- Fast training (faster convergence rate than traditional batch/online training methods, including LBFGS & SGD)
- General purpose (it is task-independent & trainable using your own tagged corpus)
- Support rich edge features (Sun et al., 2012)
- Support various training methods, including ADF training, SGD training, & Limited-memory BFGS training
- Support automatic n-fold cross-validation for tuning hyper-parameters
- 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 noun-phrase chunking task.

- `ftrain.txt` ⟶ feature file for training
- `gtrain.txt` ⟶ gold-standard tagging file for training
- `ftest.txt` ⟶ feature file for testing
- `gtest.txt` ⟶ 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 CRF-ADF to work properly.

ftrain.txt includes the total-#feature-information and the detailed features of each training instance. Take the sample file *ftrain.txt* for example, the 1st line "40636" of the file is the total-#feature-information, and it means 40636 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 *gtrain.txt*, the 1st line "3" is the total-#tag-information, and it means this task has 3 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 CRF 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:
    train (normal training without rich edge features);
    train.rich (training with rich edge features);
    test (test mode);
    tune (automatic tuning for stochastic training);
    tune.rich (automatic tuning for stochastic training with rich edge features);
    cv (automatic n-fold cross validation, default is 4-fold CV);
    cv.rich (automatic n-fold cross validation with rich edge features, $n = 4$ by default)
  Default: *train*

- 'mo' ⟶ setting *Global.modelOptimizer*
  Optional values:
  crf.adf (train CRF model with ADF training algorithm)
  crf.sgd (train CRF model with SGD of fast/lazy regularization (Sun et al., 2013))
  crf.sgder (train CRF with SGD of exact regularization)
  crf.bfgs (train CRF with Limited-Memory BFGS batch training)
  Default: *crf.adf*

- 'a' ⟶ setting *Global.rate0*
  Optional values:
  a real-value (set the initial value of the learning rate γ in (Sun et al., 2012))
  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’ $\longrightarrow$ setting *Global.ttlIter*
  Optional values:
  an integral value (the total number of training iterations)
  Default: *50*

- ‘n’ $\longrightarrow$ setting *Global.nUpdate*
  Optional values:
  an integral value (The default value of 10 means that the feature-frequency-information is updated 10 times per iteration. This value is for computing q in (Sun et al., 2012). Using this default value works well in most cases)
  Default: *10*

- ‘s’ $\longrightarrow$ 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’ $\longrightarrow$ setting *Global.outFolder*
  Optional values:
  a string (setting the folder name for storing the output files)
  Default: *out*

- ‘mb’ $\longrightarrow$ setting *Global.miniBatch*
  Optional values:
  an integral value (set the size of mini-batches in ADF and SGD stochastic training. Typically set it as 1 for online training)
  Default: *1*

- ‘up’ $\longrightarrow$ setting *Global.upper*
  Optional values:
  a real value (set ADF decay rate upper-bound, the α, in (Sun et al., 2012). The default value works well for most cases.)
  Default: *0.995*

- ‘lw’ $\longrightarrow$ setting *Global.lower*
  Optional values:
  a real value (set ADF decay rate lower-bound, the β, in (Sun et al., 2012). The default value works well for most cases.)
  Default: *0.6*

4.1 How to Train the Model

Command examples:

- `./run.exe m:train mo:crf.adf a:0.05 r:5 e:str.acc i:200`
  It means: use the normal training mode; training algorithm is *ADF*; 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 200.

- `./run.exe m:train.rich mo:crf.sgd a:0.05 r:1,5,10 e:f1 t:np.chunk i:200 of:out.sgd.f1`
  It means: training with rich edge features for higher accuracy (yet with slower speed); training algorithm is *SGD*; initial value of the learning rate γ is 0.05; 3 rounds of training will be automatically conducted with the regularizer values of 1.0, 5.0, and 10.0 respectively; evaluation metric is F-score; the task-information (chunk structures) is *np.chunk*; total number of training iteration is 200; the output folder is *out.adf.f1*.

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:test`
  It means: use the test mode (with default settings of hyper-parameters).

- `./run.exe m:test e:str.acc of:out.test`
  It means: use the test mode; evaluation metric is string-accuracy; output folder is *out.test*.

4.3 How to Perform Cross-Validation & Tuning

Cross-validation (CV) is typically used for tuning hyper-parameters on a new task. Cross-validation is automatic and is based solely on the training data, without the need to observe the test data. The tool can conduct automatic n-fold CV with user defined value of n. By default, $n = 4$. The CV command is similar to the training command, and the only difference is to set the option 'm' as CV mode (i.e., `m:cv` or `m:cv.rich`). Command examples:

- `./run.exe m:cv mo:crf.adf a:0.05 r:5 e:str.acc i:200`
  It means: cross-validation on the training data with the specific settings on hyper-parameters.

- `./run.exe m:cv.rich mo:crf.sgd a:0.05 r:1,5,10 e:f1 t:np.chunk i:200 of:out.cv`
  It means: cross-validation with rich edge features.

The `m:tune` and `m:tune.rich` options can be seen as simplified versions of the CV. Compared with the CV option, the `m:tune` and `m:tune.rich` options randomly sample a held-out dataset from the training set for tuning hyper-parameters. The `m:tune` and `m:tune.rich` options are less reliable on tuning hyper-parameters compared with the CV option, but the time cost is lower. Moreover, the `m:tune` and `m:tune.rich` options only tune major hyper-parameters of stochastic training methods, while the CV option can tune all hyper-parameters of all methods.

5. About Output Files

- *./out/trainLog.txt* $\longrightarrow$ 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/outputTag.txt* ⟶ the tags predicted from the test data.

- *./model/model.txt* ⟶ the model file derived from training.

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)

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

- `CRF.FeatureGenerator.cs` ⟶ For generating features

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

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

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

- `CRF.RichEdge.cs` ⟶ For using rich edge features described in (Sun et al., 2012), using rich-edge features typically brings higher accuracy yet with slower training speed

- `CRF.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 ADF and SGD online/stochastic training methods

7. Code Update History

Dec. 17 2013 ⟶ version 1.0

References

John Lafferty, Andrew McCallum, and Fernando Pereira. Conditional random fields: Probabilistic models for segmenting and labeling sequence data. In *ICML'01*, pages 282–289, 2001.

Xu Sun, Houfeng Wang, and Wenjie Li. Fast online training with frequency-adaptive learning rates for chinese word segmentation and new word detection. In *Proceedings of A-CL'12*, pages 253–262, 2012.

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, 2013.