Latent Structured Perceptron 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) with the emphasis on fast training speed. This toolkit includes Latent Structured Perceptron (LSP) model (Sun et al., 2009, 2013). It also includes traditional Structured Perceptron (SP) model and with the averaged version (Collins, 2002).

Main features:

- Developed with C#
- Automatic modeling of hidden information (latent structures) in the data (Sun et al., 2009, 2013)
- High accuracy (Sun et al., 2009, 2013)
- Fast training (much faster than probabilistic models like CRFs)
- General purpose (it is task-independent & trainable using your own tagged corpus)
- Support rich edge features (Sun et al., 2012)
- 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` $\longrightarrow$ feature file for training
- `gtrain.txt` $\longrightarrow$ gold-standard tagging file for training
- `ftest.txt` $\longrightarrow$ 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 the toolkit 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 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);
  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:
  lsp.paf (train LSP model with PAF training (Sun et al., 2013). This is recommended for LSP model)
  lsp.avg (train LSP model with averaged training (Sun et al., 2013))
  lsp.naive (train LSP model with naive training (Sun et al., 2013))
  sp.paf (train SP model with PAF training (Sun et al., 2013))

sp.avg (train SP model with averaged training (Collins, 2002). This is recommended for SP model)
sp.naive (train SP model with naive training (Collins, 2002))
Default: *lsp.paf*

- 'd' $\longrightarrow$ setting *Global.random*
  Optional values:
  0 (all weights are initialized with 0)
  1 (random initialization of weights. This is recommended!)
  Default: *1*

- 'e' $\longrightarrow$ 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' $\longrightarrow$ 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' $\longrightarrow$ 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. For perceptron style models, typically it needs a held-out data to decide the best iteration number to end the training (Collins, 2002).)
  Default: *50*

- '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*

4.1 How to Train the Model

Command examples:

- `./run.exe m:train mo:lsp.paf d:1 e:str.acc i:50`
  It means: use the normal training mode; model is LSP and training algorithm is *PAF*; random initialization of model weights; evaluation metric is string-accuracy; total number of training iteration is 50.

- `./run.exe m:train.rich mo:sp.avg e:f1 t:np.chunk i:50 of:out.sgd.f1`
  It means: training with rich edge features for higher accuracy (yet with slower speed); train SP model with averaged training *AVG*; evaluation metric is F-score; the task-information (chunk structures) for computing F-score is *np.chunk*; total number of training iteration is 50; 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*.

5. About Output Files

- *./out/trainLog.txt* $\longrightarrow$ recording detailed training information of each iteration.

- *./out/rawResult.txt* $\longrightarrow$ 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* $\longrightarrow$ 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* $\longrightarrow$ the tags predicted from the test data.

- *./model/model.txt* $\longrightarrow$ 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)
- `Dataset.cs` ⟶ For storing and processing data (feature files and tag files)
- `FeatureGenerator.cs` ⟶ For generating features
- `Inference.cs` ⟶ For inference & decoding
- `Model.cs` ⟶ For reading & writing model
- `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
- `ToolboxTrainTest.cs` ⟶ For high level functions of training & testing
- `OptimPerceptron.cs` ⟶ For detailed implementation of training methods

References

Michael Collins. Discriminative training methods for hidden markov models: Theory and experiments with perceptron algorithms. In *Proceedings of EMNLP'02*, pages 1–8, 2002.

Xu Sun, Takuya Matsuzaki, Daisuke Okanohara, and Jun'ichi Tsujii. Latent variable perceptron algorithm for structured classification. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1236–1242, 2009.

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 ACL'12*, pages 253–262, 2012.

Xu Sun, Takuya Matsuzaki, and Wenjie Li. Latent structured perceptrons for large-scale learning with hidden information. *IEEE Trans. Knowl. Data Eng.*, 25(9):2063–2075, 2013.