**ClearSRL User Guide**

**Setup**

1. Set up the environment variable CLEARSRL\_HOME to the directory of this README file
2. Install WordNet 3.0 (not included, and optionally the WordNet 3.1 data files)
   1. Download WordNet from <http://wordnet.princeton.edu/wordnet/download/current-version/>
3. Configure the properties file (clearsrl.properties)
   1. Set english.wordnet\_dic=${location of WordNet dict directory}

Note: the system will parse environment variables (enclose with ${ENV\_VAR}) in the properties file values.

Ex: if WordNet is installed in /home/username/WordNet-3.0, then we can set:

srl.wordnet\_dic=${HOME}/WordNet-3.0/dict

* 1. Set the constituent parser models (files end with .gr) and SRL models (files end with .model) (parser.grammar and srl.model\_file).

2 parser and SRL model sets are included, one is trained on the official OntoNotes 5.0 training data (models/en.ontonotes5.[gr|model]), the other is trained on OntoNotes 5.0 w/ additional nominal predicate annotations and BOLT data (both verbal and nominal predicates) (models/en.ontonotes5-bolt.[gr|model])

**Running ClearSRL**

1. Set up any additional (optional) property values
   1. parser.threads: indicate the number of threads the Berkeley parser will use. More threads will speed up phrase parsing on multi-core systems (recommend no more than 1 thread per physical CPU cores).
   2. srl.run.regex: acts as a file filter (uses java regular expression convention) when performing SRL on a directory of files. Only filters the actual file name, not the sub-directory path
2. Adjust the java heap space in runsrl.sh (OPTS="-Xmx??g ...”)
   1. The model trained on the official OntoNotes 5.0 corpora requires -Xmx7g or more to run, the model trained OntoNotes 5.0 w/ additional nominal predicate annotations and BOLT data requires –Xmx11g or more to run
3. Invoke java edu.colorado.clear.srl.RunSRL –h to familiarize with the options:

-compressOutput : Compress the SRL output

-depIn FILE : dependency input file/directory

-format [TEXT | PROPBANK | : srl output format: TEXT/PROPBANK

PROPBANK\_PROB | CONLL | CONLL\_DEP |

BINARY]

-h : help message

-in FILE : input file/directory

-inCorpus VAL : input corpus directory (overwrites

everyrthing else about the input)

-inList FILE : list of files in the input directory

to process (overwrites regex)

-model VAL : model file to use

-out FILE : output file/directory

-outputParse : output the intermediary parse

-parsed : input is parse trees

-prop FILE : properties file

-usePBCorpus VAL : use specified PropBank corpus to

find predicates

* 1. -compressOutput compresses the SRL output in gzip format
  2. -format outputs SRL in TEXT , PropBank or CoNLL-2005 format

The text format for “John loves Mary” looks like "[ARG0 John] [rel loves] [ARG1 Mary]"

* 1. -inList a text file containing all the filenames (including subdirectory path, one name per line) within a directory to process. This overrides any regular expression based file name filter
  2. -outputParse outputs the intermediary phrase structure parse trees if the input are text files
  3. -parsed indicates the input files are parse trees (instead of sentence segmented and tokenized text files)
  4. -out the name of the output file/directory. If omitted, the system will output to the terminal STDOUT
  5. -prop the name of the property file

1. Perform SRL on the included sample file (run in the CLEARSRL\_HOME directory) to verify everything is working

java –Xmx8g -XX:+UseConcMarkSweepGC edu.colorado.clear.srl.RunSRL -prop config/clearsrl.properties -in $HOME/data/English/sample.txt –out $HOME/data/english/sample.prop –outputParse

This should produce sample.prop in text format and sample.parse (the phrase structure parses of the tokenized input)

**Training ClearSRL**

Running:

java -Xmx12g -XX:+UseConcMarkSweepGC edu.colorado.clear.srl.TrainSRL clearsrl.properties

The amount of memory required greatly depends on the corpus size, feature cutoff, number of concurrent training threads, whether sequence or stage-2 classification is used. The -XX:+UseConcMarkSweepGC VM option greatly reduces peak memory usage. English OntoNotes 5 can be trained with 20G-30G of memory (not all needs to be in memory, some can be swapped out w/o performance hit).

The trainer does not accept any command line options besides what’s in the properties file (ones start with srl.train prefix)

**General parameters:**

model: output model file (can also be defined one with srl.model so training & running will use the same model file)

saveFrameMap(default false): save the frame files with the model. Useful if not using standard frame files (Pre-release unified frame files for English & OntoNotes 5 frame files for Chinese are provided)

corpus: comma separated list of corpora used to train the model, each corpus is separately defined

corpus.source: comma separated list of source type for the corpus. This usually includes at least PARSE (automatic parse trees), TREEBANK, PROPBANK. But may also include dependency annotation, named entity, etc. See edu.colorado.clear.srl.Sentence.Source for list of possible values

filterArguments (default true): whether to filter unlike argument candidate for argument identification/classification. This greatly reduces the number of argument candidates considered, at a very slightly lower f1 score.

trainNominal (default true): train a model w/ verb & non-verb predicates (use false when using the old WSJ corpus)

partialNominalAnnotation (default true): whether to assumes files with no nominal annotation to NOT have nominal predicates annotated. This is so the model won’t treat nouns/adjectives in those files as negative examples for the purpose of predicate identification modeling.

separateNominalClassifier (default false): use a separate classifier for arguments of nominal predicates

goldPredicateSeparation (default false): another one of those options for getting numbers when publishing papers

tmpdir (default /tmp): Temp directory during training. The model makes multiple passes of the training samples and stores them in serialized format (file prefix: clearsrl). These files are usually cleaned up unless the java VM crashes.

feature: comma separated list of argument labeling features. Features can be combined with “-” (ex: PREDICATE-VOICE means a combined predicate lemma+voice feature). (see edu.colorado.clear.srl.SRLModel.Feature for full list of individual feature)

predicateFeature: same as feature, but for predicate identification

**Specifying a corpus:**

Each parameter should start w/ the name of the corpus, then followed by it type. Ex:

on5.parse (automatic parse tree), on5.tb (treebank), on5.pb (propbank)

See edu.colorado.clear.srl.Sentence.Source for list of possible source types (like dependency parse, named entity, etc)

For each source type one can specify 3 things:

dir: directory of the source type (ex: on5.parse.dir = ${HOME}/ont5/parse)

regex: regex for filtering the files in the directory (see java Pattern class)

filelist: a list of files relative to the directory (overwrites the regex)

Usually, the corpus is keyed on the automatic parse trees (or treebank trees for training on gold trees). So it’s OK to over specify files for the other source types; they’ll be ignored.

For each corpus, you can also specify a positive integer

weight (default: 1): this is the weight of the corpus. If you have a large & small corpus, you can give more weight to the smaller corpus. Note: the current model simply duplicate samples weight number of times, so this can greatly increase overall training samples. Use caution.

**Classifier parameters:**

threads (default: 1): number of threads that may be used during training. The LinearClassifier is currently single threaded, so this would only affect cross-validation, but other parts of the system (especially feature extraction) may be parallelized in the future. Note this increases memory requirements during training.

dictionary.cutoff (default: 5): feature frequency cutoff. Any feature that occurs less than the cutoff will be discarded. 5 is a good number for a large (OntoNotes) corpus, 2 gives the best results for WSJ.

Classifier (default: LinearClassifier): class name of the classifier (only LinearClassifier & PairwiseClassifier in edu.colorado.clear.common.alg are provided, the latter wraps around the LinearClassifier & is rarely better other than old WSJ).

crossvalidation.folds (default: 5): number of folds for cross-validation. Note each fold contain all samples extracted from a single parse tree file, so make sure the input corpora do not have all the trees lumped into a single file. Cross-validation is only active when there are “support” and 2-stage classification features, or if the next parameter is enabled

crossvalidation.final (default: false): whether to perform cross-validation in the final step. This gives an idea on performance of the model (note, the numbers will be slightly optimistic compared to CoNLL-2005 metric since it only checks classification accuracy of each tree node (closer to CoNLL-2008).

liblinear.bias (default -1): whether to have a bias featured (see LibLINEAR). Not really needed for NLP as we typically have a lot of features. Currently the value must be -1 or 1

liblinear.C: C penalty value for LibLINEAR (solver type sensitive)

liblinear.solverType: solver type for LibLINEAR. L2R\_L1LOSS\_SVC\_DUAL (faster of the 2 SVC solver for NLP features) and L2R\_LR (logistic regression). The regression model trains slower, generates non-sparse feature weights (much larger compressed model size), but provides probabilities and not necessarily less accurate.

**sequence classification (**classifies arguments of root level predicates and use those as features for classifying arguments of lower level predicates, mostly useful for arguments of nominal predicates**):**

sequence.iterations (default 3): maximum number of iterative refinement for sequence classification

sequence.threshold (default 0.001): classification difference threshold before stopping iterative refinement (note, there are a lot of none argument nodes, so difference of argument nodes could be much larger than the threshold would suggest).

**stage 2 classification (**uses well-formed-ness of SRL output of the first classifier as additional features**):**

stage2.threshold (default 0.98): filter out candidates unlikely to be arguments to reduce training time/model size of the second classifier. 0.98 means candidates that are 0.98 likely to be non-arguments judged by the first classifier

stage2.liblinear.C, stage2.liblinear.solverType: if not specified, inherits value from above (options for first classifier). One way to reduce training time/model size & still have argument probability is use L2R\_L1LOSS\_SVC\_DUAL for stage 1 & L2R\_LR for stage 2

**Scoring SRL**

Run:

java –Xmx4g -XX:+UseConcMarkSweepGC edu.colorado.clear.srl. ScoreSRL -prop clearsrl.properties

This performs the CoNLL-2005 style of SRL scoring (not guaranteed to match the scores of CoNLL-2005). To get the output to match CoNLL-2005 (or in general, only score argument labeling), make sure to use the “-usePBCorpus” option when running SRL. This will ensure the automatic SRL & the PropBank have the same number of predicates. The evaluator also ignores bad formed SRLs (PropBank format), as well as predicates with multiple sets of arguments (rare WSJ annotation, mostly when there is an “empty” predicate. Ex: “The market was up today, the Dow closed at 18,000, and NASDAQ (closed) at 5000”).

Although ScoreSRL has a few command line parameters (for other purposes), most options (srl.score prefix) are specified in the properties file:

labels: the set (comma separated) of scoring labels. The default properties file contains a list of CoNLL-2005 labels. New labels may be introduced and difference languages use different labels

corpus: comma separated list of corpora used to train the model, each corpus is separately defined

systems: comma separated list of systems. Each system needs to have property for Treebank directory, PropBank directory, and a regex that operates on the PropBank files in the directory (same name & format as the corpus specification of TrainSRL). These properties should start with the srl.score.system-name namespace. gold (does not need to be included in the system list) is the reserved name for the gold PropBank to score against