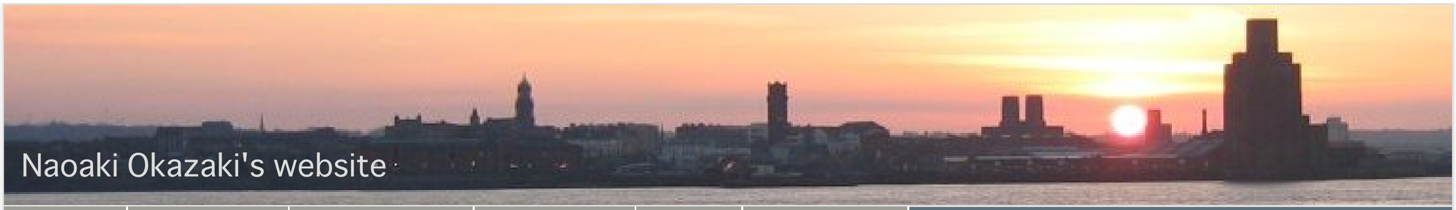


CRFsuite - Documentation

Table of Contents

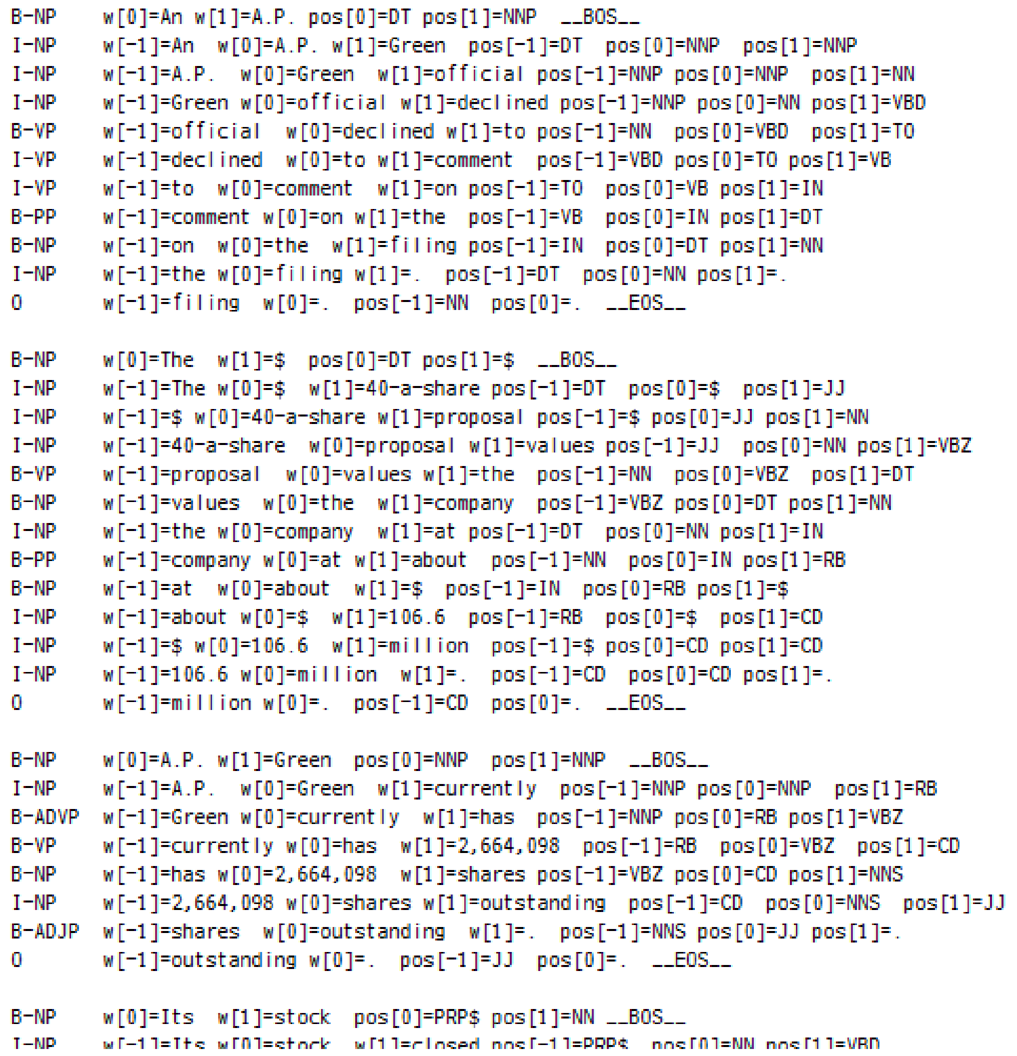
[Format of training/tagging data]
[Installation]
- [Using the binary distribution]
- [Building binaries from the source distribution]

[Usage]
- [Training]
  - [Graphical models]
  - [Training algorithms]
- [Tagging]
- [Model dump]

Format of training/tagging data

This section explains the data format used by CRFsuite for training and tagging. A *data* consists of a set of *item sequences* each of which is represented by consecutive lines and terminated by an empty line. An item sequence consists of a series of *items* whose characteristics (labels and attributes) are described in lines. An item line begins with its *label*, followed by its *attributes* separated by TAB ('\t') characters.

This is an example of training data (obtained from the CoNLL 2000 chunking shared task).



```
B-NP    w[0]=An w[1]=A.P. pos[0]=DT pos[1]=NNP  __BOS__
I-NP    w[-1]=An  w[0]=A.P. w[1]=Green  pos[-1]=DT  pos[0]=NNP  pos[1]=NNP
I-NP    w[-1]=A.P.  w[0]=Green  w[1]=official pos[-1]=NNP pos[0]=NNP  pos[1]=NN
I-NP    w[-1]=Green w[0]=official w[1]=declined pos[-1]=NNP pos[0]=NN pos[1]=VBD
B-VP    w[-1]=official  w[0]=declined w[1]=to pos[-1]=NN  pos[0]=VBD  pos[1]=TO
I-VP    w[-1]=declined  w[0]=to w[1]=comment  pos[-1]=VBD pos[0]=TO pos[1]=VB
I-VP    w[-1]=to  w[0]=comment  w[1]=on pos[-1]=TO  pos[0]=VB pos[1]=IN
B-PP    w[-1]=comment w[0]=on w[1]=the  pos[-1]=VB  pos[0]=IN pos[1]=DT
B-NP    w[-1]=on  w[0]=the  w[1]=filing pos[-1]=IN  pos[0]=DT pos[1]=NN
I-NP    w[-1]=the w[0]=filing w[1]=.  pos[-1]=DT  pos[0]=NN pos[1]=.
O       w[-1]=filing  w[0]=.  pos[-1]=NN  pos[0]=.  __EOS__

B-NP    w[0]=The  w[1]=$  pos[0]=DT pos[1]=$  __BOS__
I-NP    w[-1]=The w[0]=$  w[1]=40-a-share pos[-1]=DT  pos[0]=$  pos[1]=JJ
I-NP    w[-1]=$ w[0]=40-a-share w[1]=proposal pos[-1]=$ pos[0]=JJ pos[1]=NN
I-NP    w[-1]=40-a-share  w[0]=proposal w[1]=values pos[-1]=JJ  pos[0]=NN pos[1]=VBZ
B-VP    w[-1]=proposal  w[0]=values w[1]=the  pos[-1]=NN  pos[0]=VBZ  pos[1]=DT
B-NP    w[-1]=values  w[0]=the  w[1]=company  pos[-1]=VBZ pos[0]=DT pos[1]=NN
I-NP    w[-1]=the w[0]=company  w[1]=at pos[-1]=DT  pos[0]=NN pos[1]=IN
B-PP    w[-1]=company w[0]=at w[1]=about  pos[-1]=NN  pos[0]=IN pos[1]=RB
B-NP    w[-1]=at  w[0]=about  w[1]=$  pos[-1]=IN  pos[0]=RB pos[1]=$
I-NP    w[-1]=about w[0]=$  w[1]=106.6  pos[-1]=RB  pos[0]=$  pos[1]=CD
I-NP    w[-1]=$ w[0]=106.6  w[1]=million  pos[-1]=$ pos[0]=CD pos[1]=CD
I-NP    w[-1]=106.6 w[0]=million  w[1]=.  pos[-1]=CD  pos[0]=CD pos[1]=.
O       w[-1]=million w[0]=.  pos[-1]=CD  pos[0]=.  __EOS__

B-NP    w[0]=A.P. w[1]=Green  pos[0]=NNP  pos[1]=NNP  __BOS__
I-NP    w[-1]=A.P.  w[0]=Green  w[1]=currently  pos[-1]=NNP pos[0]=NNP  pos[1]=RB
B-ADVP  w[-1]=Green w[0]=currently  w[1]=has  pos[-1]=NNP pos[0]=RB pos[1]=VBZ
B-VP    w[-1]=currently w[0]=has  w[1]=2,664,098  pos[-1]=RB  pos[0]=VBZ  pos[1]=CD
B-NP    w[-1]=has w[0]=2,664,098  w[1]=shares pos[-1]=VBZ pos[0]=CD pos[1]=NNS
I-NP    w[-1]=2,664,098 w[0]=shares w[1]=outstanding  pos[-1]=CD  pos[0]=NNS  pos[1]=JJ
B-ADJP  w[-1]=shares  w[0]=outstanding  w[1]=.  pos[-1]=NNS pos[0]=JJ pos[1]=.
O       w[-1]=outstanding w[0]=.  pos[-1]=JJ  pos[0]=.  __EOS__

B-NP    w[0]=Its  w[1]=stock  pos[0]=PRP$ pos[1]=NN __BOS__
I-NP    w[-1]=Its w[0]=stock  w[1]=closed pos[-1]=PRP$  pos[0]=NN pos[1]=VBD
```

Figure 1. A sample data for CRFsuite

This example contains four item sequences (the last one is partially shown). The first item of the first sequence is annotated with the label "B-NP" and characterized by five attributes, "w[0]=An", "w[1]=A.P.", "pos[0]=DT", "pos[1]=NNP", and "__BOS__". The labels and attributes in this example follow a certain naming convension (feature design); for example, "B-NP" presents that the current token is a beginning of a noun phrase, "w[0]=An" presents that the surface form of the current item is "An", "pos[1]=NNP" presents that the next token is a proper noun, and "__BOS__" presents that the current item is the first item in a sequence. However, *CRFsuite does not care for the naming convension nor feature design of labels and attributes, but treats them as mere strings.* CRFsuite learns weights of associations (feature weights) between attributes and labels (e.g., if the current item has an attribute "pos[0]=DT", it is likely to have the label "B-NP"), without knowing the meaning of labels and attributes. In other words, one can design and use arbiterary features just by writing label and attribute names in data sets.

An attribute can have a scaling value separated by a colon character (':'). Formally, the amount of the influence of a feature is determined by a scaling value of the corresponding attribute multiplied by the feature weight. Roughly speaking, a scaling value of an attribute has the similar effect to the frequency of occurrences of the attribute, but can be decimal or negative. Note that a large scaling value may cause an overflow (range error) in training. Because a colon character has the special role in data sets, CRFsuite employs escape sequences; "\:" and "\\" represent ':' and '\', respectively, for attribute names. If an attribute value is omitted (without colon character), CRFsuite assumes the scaling value to be one. For example, these three items are identical in terms of attributes and their scaling values:

```
B-NP    w[1..4]=a:2 w[1..4]=man w[1..4]=eats

B-NP    w[1..4]=a w[1..4]=a w[1..4]=man w[1..4]=eats

B-NP    w[1..4]=a:2.0 w[1..4]=man:1.0 w[1..4]=eats:1.0
```

The data format for tagging is exactly the same as that for training, except that labels in the tagging data can be empty (but cannot be omitted). When tagging, CRFsuite ignores labels in the input data or uses them for measuring the performance of predictions.

This is the BNF notation representing the data format.

```
<line>              ::= <item> | <eos>
<item>              ::= <label> ('\t' <attribute>)+ <br>
<eos>               ::= <br>
<label>             ::= <string>
<attribute>         ::= <name> | <name> ':' <scaling>
<name>              ::= (<letter> | "\:" | "\\")+
<scaling>           ::= <numeric>
<br>                ::= '\n'
```

Installation

Using the binary distribution

The easiest way for installing CRFsuite is to use a binary distribution. Currently, binaries for Win32 and Linux (Intel 32bit and 64bit architectures) are distributed.

Building binaries from the source distribution

As of CRFsuite 0.5, the source package no longer include the portion of libLBFGS. In order to build CRFsuite, you need to download and build libLBFGS first.

In Windows environments, open the Visual Studio solution file (`lbfgs.sln`) of libLBFGS, and build it. The solution file builds a static-link library, `lbfgs.lib` (release build) or `lbfgs_debug.lib` (debug build), at `Release` or `Debug` directory. Because the solution file (`crfsuite.sln`) of CRFsuite assumes that the header and library files of libLBFGS exist in `win32/lbfgs` directory, create this directory, and copy `lbfgs.h`, `lbfgs.lib` and/or `lbfgs_debug.lib` into the directory. Then open the solution file (`crfsuite.sln`) and build it.

In Linux environments, download the source package of libLBFGS, and build it. If you do not want to install libLBFGS into your operating system, specify "--prefix" option to the configure script. This example installs libLBFGS to the directory `local` under the home directory (`$HOME`).

```
$ ./configure --prefix=$HOME/local
$ make
$ make install
```

Now you are ready to build CRFsuite. If you have libLFGS installed to a different directory, please specify the directory in the argument of "--with-liblbfgs" option.

```
$ ./configure --prefix=$HOME/local --with-liblbfgs=$HOME/local
$ make
$ make install
```

Usage

CRFsuite utility expects the first command-line argument to be a command name:

learn
: Train a CRF model from a training set.

tag
: Tag sequences using a CRF model.

dump
: Dump a CRF model in plain-text format.

To see the command-line syntax, use -h (--help) option.

```
$ crfsuite -h
CRFSuite 0.12  Copyright (c) 2007-2011 Naoaki Okazaki

USAGE: crfsuite <COMMAND> [OPTIONS]
    COMMAND     Command name to specify the processing
    OPTIONS     Arguments for the command (optional; command-specific)

COMMAND:
    learn       Obtain a model from a training set of instances
    tag         Assign suitable labels to given instances by using a model
    dump        Output a model in a plain-text format

For the usage of each command, specify -h option in the command argument.
```

Training

To train a CRF model from a training set, enter the following command,

```
$ crfsuite learn [OPTIONS] [DATA]
```

If the argument DATA is omitted or '-', this utility reads a training data from STDIN. To see the usage of learn command, specify -h (--help) option.

```
$ crfsuite learn -h
CRFSuite 0.12  Copyright (c) 2007-2011 Naoaki Okazaki

USAGE: crfsuite learn [OPTIONS] [DATA1] [DATA2] ...
Trains a model using training data set(s).

  DATA    file(s) corresponding to data set(s) for training; if multiple N files
          are specified, this utility assigns a group number (1...N) to the
          instances in each file; if a file name is '-', the utility reads a
          data set from STDIN

OPTIONS:
  -t, --type=TYPE       specify a graphical model (DEFAULT='1d'):
                        (this option is reserved for the future use)
      1d                    1st-order Markov CRF with state and transition
                            features; transition features are not conditioned
                            on observations
  -a, --algorithm=NAME  specify a training algorithm (DEFAULT='lbfgs')
      lbfgs                 L-BFGS with L1/L2 regularization
      l2sgd                 SGD with L2-regularization
      ap                    Averaged Perceptron
      pa                    Passive Aggressive
      arow                  Adaptive Regularization of Weights (AROW)
  -p, --set=NAME=VALUE  set the algorithm-specific parameter NAME to VALUE;
                        use '-H' or '--help-parameters' with the algorithm name
                        specified by '-a' or '--algorithm' and the graphical
                        model specified by '-t' or '--type' to see the list of
                        algorithm-specific parameters
  -m, --model=FILE      store the model to FILE (DEFAULT=''); if the value is
                        empty, this utility does not store the model
  -g, --split=N         split the instances into N groups; this option is
                        useful for holdout evaluation and cross validation
  -e, --holdout=M       use the M-th data for holdout evaluation and the rest
                        for training
  -x, --cross-validate  repeat holdout evaluations for #i in {1, ..., N} groups
                        (N-fold cross validation)
  -l, --log-to-file     write the training log to a file instead of to STDOUT;
                        The filename is determined automatically by the training
                        algorithm, parameters, and source files
  -L, --logbase=BASE    set the base name for a log file (used with -l option)
  -h, --help            show the usage of this command and exit
  -H, --help-parameters show the help message of algorithm-specific parameters;
```

```
                         specify an algorithm with '-a' or '--algorithm' option,
                         and specify a graphical model with '-t' or '--type' option
```

The following options are available for training.

-t, --type=TYPE
: Specify a graphical model used for feature generation. The default value is "1d".

    1d
    : The 1st-order Markov CRF with state and transition features (dyad features). State features are conditioned on combinations of attributes and labels, and transition features are conditioned on label bigrams.

-a, --algorithm=NAME
: Specify a training algorithm. The default value is "lbfgs".

    lbfgs
    : Gradient descent using the L-BFGS method

    l2sgd
    : Stochastic Gradient Descent with L2 regularization term

    ap
    : Averaged Perceptron

    pa
    : Passive Aggressive (PA)

    arow
    : Adaptive Regularization Of Weight Vector (AROW)

-p, --param=NAME=VALUE
: Configure a parameter for the training. CRFsuite sets the parameter (NAME) to VALUE. Available parameters depend on the graphical model and training algorithm selected. To see the help message of available parameters, use '-H' or '--help-parameters' with the algorithm name specified by '-a' or '--algorithm' and the graphical model specified by '-t' or '--type'.

-m, --model=MODEL
: Store the trained model to a file MODEL. The default value is "" (empty). CRFsuite does not store the model to a file when MODEL is empty.

-g, --split=N
: Split the instances into N groups, and assign a number in {1, ..., N} to each group. This option is mostly used to perform N-fold cross validation (with -x option). By default, CRFsuite does not split input data into groups.

-e, --holdout=M
: Use the instances of group number M for holdout evaluation. CRFsuite does not use the instances of group number M for training. By default, CRFsuite does not perform holdout evaluation.

-x, --cross-validate
: Perform N-fold cross validation. Specify the number of splits with -g option. By default, CRFsuite does not perform cross validation.

-l, --log-to-file
: Write out the log message of training to a file. The file name is determined automatically from the command-line arguments (e.g., training algorithm, graphical model, parameters, source files). By default, CRFsuite writes out the log message to STDOUT.

-L, --logbase=BASE
: Specify the base name for the log file (used with -l option). By default, the base name is "log.crfsuite".

-h, --help

Show the usage of this command and exit.

-H, --help-parameters
: Show the list of parameters and their descriptions. Specify the graphical model and training algorithm with -t and -a options, respectively.

-p, --param=NAME=VALUE
: Configure a parameter for the training. CRFsuite sets the parameter (NAME) to VALUE. To see the list of parameters and their descriptions, use -H (--help-parameters) option.

Here are some examples of CRFsuite command-lines for training.

Train a CRF model from `train.txt` with the default parameters, and store the model to `CRF.model`.

```
$ crfsuite learn -m CRF.model train.txt
```

Train a CRF model from `STDIN` with the default parameters.

```
$ cat train.txt | crfsuite learn -
```

Train a CRF model from `train.txt` (group #1). During the trainig, test the model with a holdout data `test.txt` (group #2).

```
$ crfsuite learn -e2 train.txt test.txt
```

Perform 10-fold cross-validation on the training data `train.txt`. The log output will be stored in `log.crfsuite_lbfgs` (the name of the log file may vary depending on the training parameters).

```
$ crfsuite learn -g10 -x -l train.txt
```

Graphical models

1d: 1st-order Markov CRF with dyad features

The 1st-order Markov CRF with state and transition features (dyad features). State features are conditioned on combinations of attributes and labels, and transition features are conditioned on label bigrams.

feature.minfreq=VALUE
: Cut-off threshold for occurrence frequency of a feature. CRFsuite will ignore features whose frequencies of occurrences in the training data are no greater than VALUE. The default value is 0 (i.e., no cut-off).

feature.possible_states=BOOL
: Specify whether CRFsuite generates state features that do not even occur in the training data (i.e., negative state features). Setting BOOL to 1, CRFsuite generates state features that associate all of possible combinations between attributes and labels. Suppose that the numbers of attributes and labels are A and L respectively, this function will generate (A * L) features. Enabling this function may improve the labeling accuracy because the CRF model can learn the condition where an item is not predicted to its reference label. However, this function may also increase the number of features and slow down the training process drastically. This function is disabled by default.

feature.possible_transitions=BOOL
: Specify whether CRFsuite generates transition features that do not even occur in the training data (i.e., negative transition features). Setting BOOL to 1, CRFsuite generates transition features that associate all of possible label pairs. Suppose that the number of labels in the training data is L, this function will generate (L * L) transition features. This function is disabled by default.

Here are some examples of CRFsuite command-lines.

Features occurring less than twice will be unused for training.

```
$ crfsuite learn -m CRF.model -p feature.minfreq=2 train.txt
```

Generate negative state and transition features (aka, dense feature set).

```
$ crfsuite learn -m CRF.model -p feature.possible_states=1 -p feature.possible_transitions=1 trai
```

Training algorithms

lbfgs: Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method

Maximize the logarithm of the likelihood of the training data with L1 and/or L2 regularization term(s) using the Limited-memory Broyden-Fletcher-Goldfarb-Shanno (L-BFGS) method. When a non-zero coefficient for L1 regularization term is specified, the algorithm switches to the Orthant-Wise Limited-memory Quasi-Newton (OWL-QN) method. In practice, this algorithm improves feature weights very slowly at the beginning of a training process, but converges to the optimal feature weights quickly in the end.

c1=VALUE
: The coefficient for L1 regularization. If a non-zero value is specified, CRFsuite switches to the Orthant-Wise Limited-memory Quasi-Newton (OWL-QN) method. The default value is zero (no L1 regularization).

c2=VALUE
: The coefficient for L2 regularization. The default value is 1.

max_iterations=NUM
: The maximum number of iterations for L-BFGS optimization. The L-BFGS routine terminates if the iteration count exceeds this value. The default value is set to the maximum value of integer on the machine (INT_MAX).

num_memories=NUM
: The number of limited memories that L-BFGS uses for approximating the inverse hessian matrix. The default value is 6.

epsilon=VALUE
: The epsilon parameter that determines the condition of convergence. The default value is 1e-5.

stop=NUM
: The duration of iterations to test the stopping criterion. The default value is 10.

delta=VALUE
: The threshold for the stopping criterion; an L-BFGS iteration stops when the improvement of the log likelihood over the last ${stop} iterations is no greater than this threshold. The default value is 1e-5.

linesearch=STRING
: The line search method used in the L-BFGS algorithm. Available methods are: "MoreThuente" (MoreThuente method proposd by More and Thuente), "Backtracking" (Backtracking method with regular Wolfe condition), and "StrongBacktracking" (Backtracking method with strong Wolfe condition). The default method is "MoreThuente".

max_linesearch=NUM
: The maximum number of trials for the line search algorithm. The default value is 20.

Here are some examples of command-lines for L-BFGS training.

Training a model with L2 regularization (c1=0, c2=1.0).

```
$ crfsuite learn -m CRF.model -a lbfgs -p c2=1 train.txt
```

Training a model with L1 regularization (c1=1.0, c2=0).

```
$ crfsuite learn -m CRF.model -a lbfgs -p c1=1 -p c2=0 train.txt
```

Training a model with L1 and L2 regularization (c1=1.0, c2=1.0).

```
$ crfsuite learn -m CRF.model -a lbfgs -p c1=1 -p c2=1 train.txt
```

l2sgd: Stochastic Gradient Descent (SGD) with L2 regularization

Maximize the logarithm of the likelihood of the training data with L2 regularization term(s) using Stochastic Gradient Descent (SGD) with batch size 1. This algorithm usually approaches to the optimal feature weights quite rapidly, but shows slow convergences at the end.

c2=VALUE
: The coefficient for L2 regularization. The default value is 1.

max_iterations=NUM
: The maximum number of iterations (epochs) for SGD optimization. The optimization routine terminates if the iteration count exceeds this value. The default value is 1000.

period=NUM
: The duration of iterations to test the stopping criterion. The default value is 10.

delta=VALUE
: The threshold for the stopping criterion; an optimization process stops when the improvement of the log likelihood over the last ${period} iterations is no greater than this threshold. The default value is 1e-5.

calibration.eta=VALUE
: The initial value of learning rate (eta) used for calibration. The default value is 0.1.

calibration.rate=VALUE
: The rate of increase/decrease of learning rate for calibration. The default value is 2.

calibration.samples=NUM
: The number of instances used for calibration. The calibration routine randomly chooses instances no larger than VALUE. The default value is 1000.

calibration.candidates=NUM
: The number of candidates of learning rate. The calibration routine terminates after finding NUM candidates of learning rates that can increase log-likelihood. The default value is 10.

calibration.max_trials=NUM
: The maximum number of trials of learning rates for calibration. The calibration routine terminates after trying NUM candidate values of learning rates. The default value is 20.

Here is an example of command-line for SGD training.

Training a model with L2 regularization (c2=1.0).

```
$ crfsuite learn -m CRF.model -a l2sgd -p c2=1 train.txt
```

ap: Averaged Perceptron

When the current model parameter cannot predict the item sequence correctly, this algorithm applies perceptron updates to the model. The algorithm takes the average of feature weights at all updates in the training process. The algorithm is fastest in terms of training speed. Even though the algorithm is very simple, it exhibits high prediction performance. In practice, it is necessary to stop a training process by specifying the maximum number of iterations, which should be tuned on a development set.

max_iterations=NUM
: The maximum number of iterations (epochs). The optimization routine terminates if the iteration count exceeds this value. The default value is 100.

epsilon=VALUE
: The epsilon parameter that determines the condition of convergence. The optimization routine terminates if the ratio of incorrect labels predicted by the model is no greater than VALUE. The default value is 1e-5.

Here is an example of command-line for Averaged Perceptron.

Training a model with 10 iterations at most.

```
$ crfsuite learn -m CRF.model -a ap -p max_iterations=10 train.txt
```

pa: Passive Aggressive

Given an item sequence (x, y) in the training data, the algorithm computes the loss: s(x, y') - s(x, y) + sqrt(d(y', y)), where s(x, y') is the score of the Viterbi label sequence, s(x, y) is the score of the label sequence of the training data, and d(y', y) measures the distance between the Viterbi label sequence (y') and the reference label sequence (y). If the item suffers from a non-negative loss, the algorithm updates the model based on the loss.

type=NUM
: The strategy for updating feature weights: PA without slack variables (0), PA type I (1), or PA type II (2). The default value is 1.

c=VALUE
: Aggressiveness parameter (used only for PA-I and PA-II). This parameter controls the influence of the slack term on the objective function. The default value is 1.

error_sensitive=BOOL
: If this parameter is true (non-zero), the optimization routine includes into the objective function the square root of the number of incorrect labels predicted by the model. The default value is 1 (true).

averaging=BOOL
: If this parameter is true (non-zero), the optimization routine computes the average of feature weights at all updates in the training process (similarly to Averaged Perceptron). The default value is 1 (true).

max_iterations=NUM
: The maximum number of iterations (epochs). The optimization routine terminates if the iteration count exceeds this value. The default value is 100.

epsilon=VALUE
: The epsilon parameter that determines the condition of convergence. The optimization routine terminates if the mean loss is no greater than VALUE. The default value is 1e-5.

arow: Adaptive Regularization Of Weight Vector (AROW)

Given an item sequence (x, y) in the training data, the algorithm computes the loss: s(x, y') - s(x, y), where s(x, y') is the score of the Viterbi label sequence, and s(x, y) is the score of the label sequence of the training data.

variance=VALUE
: The initial variance of every feature weight. The algorithm initialize a vector of feature weights as a multivariate Gaussian distribution with mean 0 and variance VALUE. The default value is 1.

gamma=VALUE
: The tradeoff between loss function and changes of feature weights. The default value is 1.

max_iterations=NUM
: The maximum number of iterations (epochs). The optimization routine terminates if the iteration count exceeds this value. The default value is 100.

epsilon=VALUE
: The epsilon parameter that determines the condition of convergence. The optimization routine terminates if the mean loss is no greater than VALUE. The default value is 1e-5.

Tagging

To tag a data using a CRF model, enter the following command,

```
$ crfsuite tag [OPTIONS] [DATA]
```

If the argument DATA is omitted or '-', CRFsuite reads a data from STDIN.To see the usage of tag command, specify -h (--help) option.

```
$ crfsuite tag -h
CRFSuite 0.12  Copyright (c) 2007-2011 Naoaki Okazaki

USAGE: crfsuite tag [OPTIONS] [DATA]
Assign suitable labels to the instances in the data set given by a file (DATA).
If the argument DATA is omitted or '-', this utility reads a data from STDIN.
Evaluate the performance of the model on labeled instances (with -t option).

OPTIONS:
    -m, --model=MODEL   Read a model from a file (MODEL)
    -t, --test          Report the performance of the model on the data
    -r, --reference     Output the reference labels in the input data
    -p, --probability   Output the probability of the label sequences
    -i, --marginal      Output the marginal probabilities of items
    -q, --quiet         Suppress tagging results (useful for test mode)
    -h, --help          Show the usage of this command and exit
```

The following options are available for tagging.

-m, --model=MODEL
: A filename from which CRFsuite reads a CRF model.

-t, --test
: Evaluate the performance (accuracy, precision, recall, f1 measure) of the CRF model, assuming that the input data is labeled. This function is disabled by default.

-r, --reference
: Output the reference labels in parallel with predicted labels, assuming that the input data is labeled. This function is disabled by default.

-p, --probability
: Output the probabilities of label sequences predicted by the model. When this function is enabled, a label sequence begins with a line "@probability\tx.xxxx", where "x.xxxx" presents the probability of the sequence and "\t" denotes a TAB character. This function is disabled by default.

-i, --marginal
: Output the marginal probabilities of labels. When this function is enabled, each predicted label is followed by ":x.xxxx", where "x.xxxx" presents the probability of the label. This function is disabled by default.

-q, --quiet
: Suppress the output of tagged labels. This function is useful for evaluating a CRF model with -t option.

-h, --help
: Show the usage of this command and exit.

Here are some examples of CRFsuite command-lines for tagging.

Tag a data `test.txt` using a CRF model `CRF.model`

```
$ crfsuite tag -m CRF.model test.txt
```

Evaluate a CRF model `CRF.model` on the labeled data `test.txt`.

```
$ crfsuite tag -m CRF.model -qt test.txt
```

Model dump

To dump a CRF model in plain-text format, enter the following command,

```
$ crfsuite dump <MODEL>
```