

SentiStrength Java User Manual

This document describes the main tasks and options for the Java version of SentiStrength. Java must be installed on your computer. SentiStrength can then run via the command prompt using a command like:

```
java -jar SentiStrength.jar sentidata C:/SentiStrength_Data/ text
i+don't+hate+you.
```

Contents

SentiStrength Java User Manual.....	1
Quick start.....	2
Windows, Linux.....	2
Mac.....	2
Sentiment classification tasks.....	2
Classify a single text.....	3
Classify all lines of text in a file for sentiment [includes accuracy evaluations].....	3
Classify texts in a column within a file or folder.....	3
Listen at a port for texts to classify.....	3
Run interactively from the command line.....	4
Process stdin and send to stdout.....	4
Import the JAR file to run within your Java program.....	4
Improving the accuracy of SentiStrength.....	6
Basic manual improvements.....	6
Optimise sentiment strengths of existing sentiment terms.....	7
Suggest new sentiment terms (from terms in misclassified texts).....	7
Options:.....	8
Explain the classification.....	8
Only classify text near specified keywords.....	8
Classify positive (1 to 5) and negative (-1 to -5) sentiment strength separately.....	8
Use trinary classification (positive-negative-neutral).....	8
Use binary classification (positive-negative).....	8
Use a single positive-negative scale classification.....	8
Location of linguistic data folder.....	8
Location of sentiment term weights.....	9
Location of output folder.....	9
File name extension for output.....	9
Classification algorithm parameters.....	9
Additional considerations.....	10
Language issues.....	10
Long texts.....	10
Machine learning evaluations.....	10
Evaluation options.....	11

Quick start

Windows, Linux

1. Save SentiStrength.jar to your main computer Desktop and Unzip SentiStrength_Data.zip to a folder on your main Desktop called SentiStrength_Data. So if you open SentiStrength_Data you should see all the input files (can also run from USB or elsewhere).
2. Unzip the downloaded SentiStrength text files from the zip file into a new folder – a subfolder of the Desktop folder is easiest.
3. Click the Windows start button, type **cmd** and then select cmd.exe to start a command prompt. Use **Terminal** for Linux (Ctrl-Alt-T).
4. (The tricky bit) At the command prompt, navigate to the folder containing SentiStrength.jar by (Windows) entering the drive letter, followed by a colon to change the default directory to the USB drive. Then type `cd [name]` with the name of the folder containing SentiStrength. More information here (Windows) if you get stuck: <http://www.digitalcitizen.life/command-prompt-how-use-basic-commands>
5. Test SentiStrength with the following command, where the path of the SentiStrength data folder name will need to be changed to the name on your computer (Windows tip: commands can be pasted to the command prompt with the right click menu).
java -jar SentiStrength.jar sentidata D:/senti/SentiStrength_Data/ text i+like+you. Explain

Mac

1. Save SentiStrength.jar to your main computer Desktop and Unzip SentiStrength_Data.zip to a folder on your main Desktop called SentiStrength_Data. So if you open SentiStrength_Data you should see all the input files (can also run from USB or elsewhere).
2. Unzip the downloaded SentiStrength text files from the zip file into a new folder – a subfolder of the Desktop folder is easiest.
3. Start **Terminal** for Macs (Applications|Utilities).
4. In the terminal window, type the following (case sensitive) command and press return to navigate to the Desktop (i.e., where SentiStrengthCom.jar is).
cd Desktop
5. Test SentiStrength with the following command. **java -jar SentiStrength.jar sentidata SentiStrength_Data/ text i+like+you. Explain**

Sentiment classification tasks

SentiStrength can classify individual texts or multiple texts and can be invoked in many different ways. This section covers these methods although most users only need one of them.

Classify a single text

text [text to process]

The submitted text will be classified and the result returned in the form +ve -space- -ve.

If the classification method is trinary, binary or scale then the result will have the form

+ve -space- -ve -space- overall. E.g.,

```
java -jar SentiStrength.jar sentidata C:/SentiStrength_Data/ text
i+love+your+dog.
```

The result will be: 3 -1

Classify all lines of text in a file for sentiment [includes accuracy evaluations]

input [filename]

Each line of [filename] will be classified for sentiment. Here is an example.

```
java -jar SentiStrength.jar sentidata C:/SentiStrength_Data/ input
myfile.txt
```

A new file will be created with the sentiment classifications added to the end of each line.

If the task is to test the accuracy of SentiStrength, then the file may have +ve codes in the 1st column, then negative codes in the 2nd column and text in the last column. If using binary/trinary/scale classification then the first column can contain the human coded values. Columns must be tab-separated. If human coded sentiment scores are included in the file then the accuracy of SentiStrength will be compared against them.

Classify texts in a column within a file or folder

For each line, the text in the specified column will be extracted and classified, with the result added to an extra column at the end of the file (all three parameters are compulsory).

```
annotateCol [col # 1..] (classify text in col, result at line end)
inputFolder [foldername] (all files in folder will be *annotated*)
fileSubstring [text] (string must be present in files to annotate)
Ok to overwrite files [overwrite]
```

If a folder is specified instead of a filename (i.e., an input parameter) then all files in the folder are processed as above. If a fileSubstring value is specified, then only files matching the substring will be classified. The parameter overwrite must be specified to explicitly allow the input files to be modified. This is a purely safety feature. E.g.,

```
java -jar SentiStrength.jar sentidata C:/SentiStrength_Data/
annotateCol 1 inputFolder C:/textfiles/ fileSubstring txt
```

Listen at a port for texts to classify

listen [port number to listen at - call OR

This sets the program to listen at a port number for texts to classify, e.g., to listen at port 81 for texts for trinary classification:

```
java -jar SentiStrength.jar sentidata C:/SentiStrength_Data/
listen 81 trinary
```

The texts must be URLEncoded and submitted as part of the URL. E.g., if the listening was set up on port 81 then requesting the following URL would trigger classification of the text "love you": `http://127.0.0.1:81/love%20you`

The result for this would be 3 -1 1. This is: (+ve classification) (-ve classification) (trinary classification)

Run interactively from the command line

`cmd` (can also set options and sentidata folder). E.g.,

```
java -jar c:\SentiStrength.jar cmd sentidata C:/SentiStrength_Data/
```

This allows the program to classify texts from the command prompt. After running this every line you enter will be classified for sentiment. To finish enter @end

Process stdin and send to stdout

`stdin` (can also set options and sentidata folder). E.g.,

```
java -jar c:\SentiStrength.jar stdin sentidata C:/SentiStrength_Data/
```

SentiStrength will classify all texts sent to it from stdin and then will close. This probably the most efficient way of integrating SentiStrength efficiently with non-Java programs.

The alternatives are the Listen at a port option or dumping the texts to be classified into a file and then running SentiStrength on the file.

The parameter `textCol` can be set [default 0 for the first column] if the data is sent in multiple tab-separated columns and one column contains the text to be classified.

The results will be appended to the end of the input data and send to STD out.

The Java loop code for this is essentially:

```
while((textToParse = stdin.readLine()) != null) {
    //code to analyse sentiment and return results
}
```

So for greatest efficiency, null should not be sent to stdin as this will close the program.

Import the JAR file to run within your Java program

Import the Jar and > initialise it by sending commands to `public static void main(String[] args)` in `public class SentiStrength` and then call `public String computeSentimentScores(String sentence)` also from `public class SentiStrength` to get each text processed. Here is some sample code for after importing the Jar and creating a class:

```
package uk.ac.wlv.sentistrengthapp; //Whatever package name you choose
import uk.ac.wlv.sentistrength.*;
public class SentiStrengthApp {
    public static void main(String[] args) {
        //Method 1: one-off classification (inefficient for multiple classifications)
        //Create an array of command line parameters, including text or file to process
        String ssthInitialisationAndText[] = {"sentidata", "f:/SentiStrength_Data/",
            "text", "I+hate+frogs+but+love+dogs.", "explain"};
        SentiStrength.main(ssthInitialisationAndText);

        //Method 2: One initialisation and repeated classifications
        SentiStrength sentiStrength = new SentiStrength();
```

```

//Create an array of command line parameters to send (not text or file to
    process)
String ssthInitialisation[] = {"sentidata", "f:/SentStrength_Data/", "explain"};
SentiStrength.initialise(ssthInitialisation); //Initialise
//can now calculate sentiment scores quickly without having to initialise again
System.out.println(sentiStrength.computeSentimentScores("I hate frogs."));
System.out.println(sentiStrength.computeSentimentScores("I love dogs."));
    }
}

```

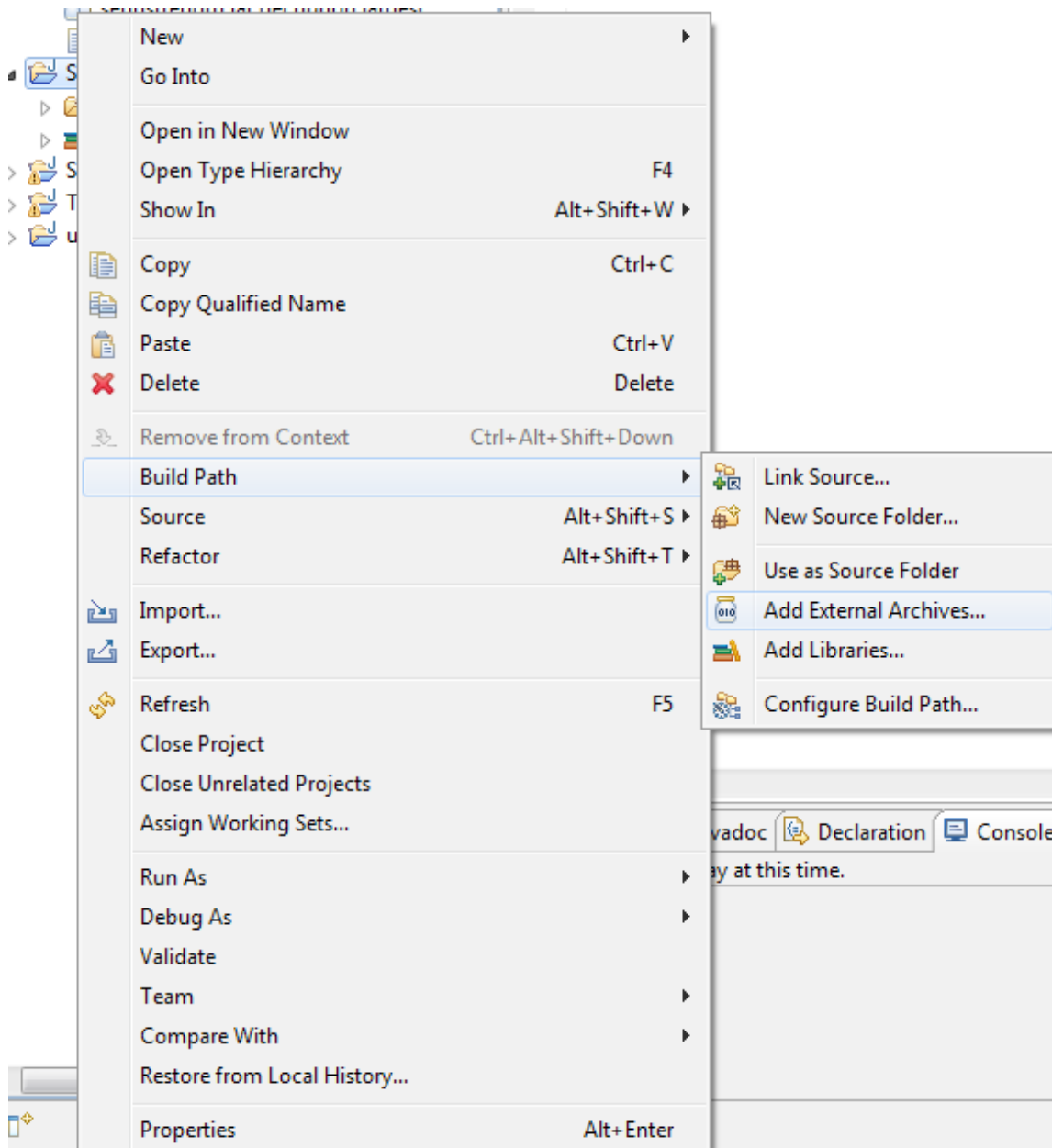
To instantiate multiple classifiers you can start and initialise each one separately.

```

SentiStrength classifier1 = new SentiStrength();
SentiStrength classifier2 = new SentiStrength();
//Also need to initialise both, as above
String ssthInitialisation1[] = {"sentidata", "f:/SentStrength_Data/", "explain"};
classifier1.initialise(ssthInitialisation1); //Initialise
String ssthInitialisation2[] = {"sentidata", "f:/SentStrength_Spanish_Data/"};
Classifier2.initialise(ssthInitialisation2); //Initialise
// after initialisation, can call both whenever needed:
String result_from_classifier1 = classifier1.computeSentimentScores(input);
String result_from_classifier2 = classifier2.computeSentimentScores(input);

```

Note: if using Eclipse then the following imports SentiStrength into your project (there are also other ways).



Improving the accuracy of SentiStrength

Basic manual improvements

If you see a systematic pattern in the results, such as the term “disgusting” typically having a stronger or weaker sentiment strength in your texts than given by SentiStrength then you can edit the text files with SentiStrength to change this. Please edit SentiStrength’s input files using a plain text editor because if it is edited with a word processor then SentiStrength may not be able to read the file afterwards.

Optimise sentiment strengths of existing sentiment terms

SentiStrength can suggest revised sentiment strengths for the **EmotionLookupTable.txt** in order to give more accurate classifications for a given set of texts. This option needs a large (>500) set of texts in a plain text file with a human sentiment classification for each text. SentiStrength will then try to adjust the **EmotionLookupTable.txt** term weights to be more accurate when classifying these texts. It should then also be more accurate when classifying similar texts.

optimise [Filename for optimal term strengths (e.g. EmotionLookupTable2.txt)]

This creates a new emotion lookup table with improved sentiment weights based upon an input file with human coded sentiment values for the texts. This feature allows

SentiStrength term weights to be customised for new domains. E.g.,

```
java -jar c:/SentiStrength.jar minImprovement 3 input
C:/twitter4242.txt optimise
C:/twitter4242OptimalSentimentLookupTable.txt
```

This is very slow (hours or days) if the input file is large (hundreds of thousands or millions, respectively). The main optional parameter is minImprovement (default value 2). Set this to specify the minimum overall number of additional correct classifications to change the sentiment term weighting. For example, if increasing the sentiment strength of *love* from 3 to 4 improves the number of correctly classified texts from 500 to 502 then this change would be kept if minImprovement was 1 or 2 but rejected if minImprovement was >2. Set this higher to have more robust changes to the dictionary. Higher settings are possible with larger input files.

To check the performance on the new dictionary, the file could be reclassified using it instead of the original SentimentLookupTable.txt as follows:

```
java -jar c:/SentiStrength.jar input C:/twitter4242.txt
EmotionLookupTable C:/twitter4242OptimalSentimentLookupTable.txt
```

Suggest new sentiment terms (from terms in misclassified texts)

SentiStrength can suggest a new set of terms to add to the **EmotionLookupTable.txt** in order to give more accurate classifications for a given set of texts. This option needs a large (>500) set of texts in a plain text file with a human sentiment classification for each text. SentiStrength will then list words not found in the **EmotionLookupTable.txt** that may indicate sentiment. Adding some of these terms should make SentiStrength more accurate when classifying similar texts.

termWeights

This lists all terms in the data set and the proportion of times they are in incorrectly classified positive or negative texts. Load this into a spreadsheet and sort on the PosClassAvDiff and NegClassAvDiff to get an idea about terms that either should be added to the sentiment dictionary because one of these two values is high. This option also lists words that are already in the sentiment dictionary. Must be used with a text file containing correct classifications. E.g.,

```
java -jar c:/SentiStrength.jar input C:/twitter4242.txt
termWeights
```

This is very slow (hours or days) if the input file is large (tens of thousands or millions, respectively).

Interpretation: In the output file, the column PosClassAvDiff means the average difference between the predicted sentiment score and the human classified sentiment score for texts containing the word. For example, if the word “nasty” was in two texts and SentiStrength had classified them both as +1,-3 but the human classifiers had classified the texts as (+2,-3) and (+3,-5) then PosClassAvDiff would be the average of 2-1 (first text) and 3-1 (second text) which is 1.5. All the negative scores are ignored for PosClassAvDiff

NegClassAvDiff is the same as for PosClassAvDiff except for the negative scores.

Options:

Explain the classification

explain

Adding this parameter to most of the options results in an approximate explanation being given for the classification. E.g.,

```
java -jar SentiStrength.jar text i+don't+hate+you. explain
```

Only classify text near specified keywords

keywords [comma-separated list - sentiment only classified close to these]

wordsBeforeKeywords [words to classify before keyword (default 4)]

wordsAfterKeywords [words to classify after keyword (default 4)]

Classify positive (1 to 5) and negative (-1 to -5) sentiment strength separately

This is the default and is used unless binary, trinary or scale is selected. Note that 1 indicates no positive sentiment and -1 indicates no negative sentiment. There is no output of 0.

Use trinary classification (positive-negative-neutral)

trinary (report positive-negative-neutral classification instead)

The result for this would be like 3 -1 1. This is: (+ve classification) (-ve classification) (trinary classification)

Use binary classification (positive-negative)

binary (report positive-negative classification instead)

The result for this would be like 3 -1 1. This is: (+ve classification) (-ve classification) (binary classification)

Use a single positive-negative scale classification

scale (report single -4 to +4 classification instead)

The result for this would be like 3 -4 -1. This is: (+ve classification) (-ve classification) (scale classification)

Location of linguistic data folder

sentiata [folder for SentiStrength data (end in slash, no spaces)]

Location of sentiment term weights

EmotionLookupTable [filename (default: EmotionLookupTable.txt or SentimentLookupTable.txt)].

Location of output folder

outputFolder [foldername where to put the output (default: folder of input)]

File name extension for output

resultsextension [file-extension for output (default _out.txt)]

Classification algorithm parameters

These options change how the sentiment analysis algorithm works.

- alwaysSplitWordsAtApostrophes (split words when an apostrophe is met – important for languages that merge words with ', like French (e.g., t'aime -> t ' aime with this option t'aime without))
- noBoosters (ignore sentiment booster words (e.g., very))
- noNegatingPositiveFlipsEmotion (don't use negating words to flip +ve words)
- noNegatingNegativeNeutralisesEmotion (don't use negating words to neuter -ve words)
- negatedWordStrengthMultiplier (strength multiplier when negated (default=0.5))
- maxWordsBeforeSentimentToNegate (max words between negator & sentiment word (default 0))
- noIdioms (ignore idiom list)
- questionsReduceNeg (-ve sentiment reduced in questions)
- noEmoticons (ignore emoticon list)
- exclamations2 (exclamation marks count them as +2 if not -ve sentence)
- mood [-1,0,1](interpretation of neutral emphasis (e.g., miiike; hello!!). -1 means neutral emphasis interpreted as -ve; 1 means interpreted as +ve; 0 means emphasis ignored)
- noMultiplePosWords (don't allow multiple +ve words to increase +ve sentiment)
- noMultipleNegWords (don't allow multiple -ve words to increase -ve sentiment)
- noIgnoreBoosterWordsAfterNegatives (don't ignore boosters after negating words)
- noDictionary (don't try to correct spellings using the dictionary by deleting duplicate letters from unknown words to make known words)
- noDeleteExtraDuplicateLetters (don't delete extra duplicate letters in words even when they are impossible, e.g., heyyyy) [this option does not check if the new word is legal, in contrast to the above option]
- illegalDoubleLettersInWordMiddle [letters never duplicate in word middles] this is a list of characters that never occur twice in succession. For English the following list is used (default): ahijkquvxyz Never include w in this list as it often occurs in www

- `illegalDoubleLettersAtWordEnd` [letters never duplicate at word ends] this is a list of characters that never occur twice in succession at the end of a word. For English the following list is used (default): `achijkmnpqruvwxyz`
- `noMultipleLetters` (don't use the presence of additional letters in a word to boost sentiment)

Additional considerations

Language issues

If using a language with a character set that is not the standard ASCII collection then please save in UTF8 format and use the `utf8` option to get SentiStrength to read the input files as UTF8. If using European language like Spanish with diacritics, please try both with and without the `utf8` option – depending on your system, one or the other might work (Possibly due to a weird ANSI/ASCII coding issue with Windows).

Long texts

SentiStrength is designed for short texts but can be used for polarity detection on longer texts with the following options (see binary or trinary below). This works similarly to Maite Taboada's SOCAL program. In this mode, the total positive sentiment is calculated and compared to the total negative sentiment. If the total positive is bigger than 1.5* the total negative sentiment then the classification is positive, otherwise it is negative. Why 1.5? Because negativity is rarer than positivity, so stands out more (see the work of Maite Taboada).

```
java -jar SentiStrength.jar sentidata C:/SentiStrength_Data/ text
I+hate+frogs+but+love+dogs.+Do+You+like. sentenceCombineTot
paragraphCombineTot trinary
```

If you prefer a multiplier other than 1.5 then set it with the `negativeMultiplier` option.

E.g., for a multiplier of 1 (equally weighting positive and negative test) try:

```
java -jar SentiStrength.jar sentidata C:/SentiStrength_Data/ text
I+hate+frogs+but+love+dogs.+Do+You+like. sentenceCombineTot
paragraphCombineTot trinary negativeMultiplier 1
```

Machine learning evaluations

These are machine learning options to evaluate SentiStrength for academic research. The basic command is *train*.

train (evaluate SentiStrength by training term strengths on results in file). An input file of 500+ human classified texts is also needed - e.g.,

```
java -jar SentiStrength.jar train input C:\1041MySpace.txt
```

This attempts to optimise the sentiment dictionary using a machine learning approach and 10-fold cross validation. This is equivalent to using the command *optimise* on a random 90% of the data, then evaluating the results on the remaining 10% and repeating this 9 more times with the remaining 9 sections of 10% of the data. The accuracy results reported are the average of the 10 attempts. This estimates the improved accuracy gained from using the *optimise* command to improve the sentiment dictionary.

The output of this is two files. The file ending in **_out.txt** reports various accuracy statistics (e.g., number and proportion correct, number and proportion within 1 of the correct value; correlation between SentiStrength and human coded values. The file ending in **_out_termStrVars.txt** reports the changes to the sentiment dictionary in each of the folds. Both files also report the parameters used for the sentiment algorithm and machine learning. See the *What do the results mean?* section at the end for more information.

Evaluation options

- *all* Test all option variations listed in Classification Algorithm Parameters above rather than use the default options
- *tot* Optimise by the number of correct classifications rather than the sum of the classification differences
- *iterations* [number of 10-fold iterations (default 1)] This sets the number of times that the training and evaluation is conducted. A value of 30 is recommended to help average out differences between runs.
- *minImprovement* [min extra correct class. to change sentiment weights (default 2)] This sets the minimum number of extra correct classifications necessary to adjust a term weight during the training phase.
- *multi* [# duplicate term strength optimisations to change sentiment weights (default 1)] This is a kind of super-optimisation. Instead of being optimised once, term weights are optimised multiple times from the starting values and then the average of these weights is taken and optimised and used as the final optimised term strengths. This should in theory give better values than optimisation once. e.g.,

```
java -jar SentiStrength.jar multi 8 input C:\1041MySpace.txt iterations 2
```

Example: Using SentiStrength for 10-fold cross-validation

What is this? This estimates the accuracy of SentiStrength *after* it has optimised the term weights for the sentiment words (i.e., the values in the file EmotionLookupTable.txt).

What do I need for this test? You need an input file that is a list of texts with human classified values for positive (1-5) and negative (1-5) sentiment. Each line of the file should be in the format:

Positive <tab> Negative <tab> text

How do I run the test? Type the following command, replacing the filename with your own file name.

```
java -jar SentiStrength.jar input C:\1041MySpace.txt iterations 30
```

This should take up to one hour – much longer for longer files. The output will be a list of accuracy statistics. Each 10-fold cross-validation

What does 10-fold cross-validation mean? See the k-fold section in [http://en.wikipedia.org/wiki/Cross-validation_\(statistics\)](http://en.wikipedia.org/wiki/Cross-validation_(statistics)). Essentially, it means that the same data is used to identify the best sentiment strength values for the terms in EmotionLookupTable.txt as is used to evaluate the accuracy of the revised (trained) algorithm – but this isn't cheating when it is done this way. The first line in the results file gives the accuracy of SentiStrength with the original term weights in EmotionLookupTable.txt.

What do the results mean? The easiest way to read the results is to copy and paste them into a spreadsheet like Excel. The table created lists the options used to classify the texts and well as the results. Here is an extract from the first two rows of the key results. It gives the total number correct for positive sentiment (Pos Correct) and the proportion correct (Pos Correct/Total). It also reports the number of predictions that are correct or within 1 of being correct (Pos Within1). The same information is given for negative sentiment.

Pos Correct	Pos Correct/ Total	Neg Correct	Neg Correct/ Total	Pos Within1	Pos Within1/ Total	Neg Within1	Neg Within1/ Total
653	0.627281	754	0.724304	1008	0.9683	991	0.951969

Here is another extract of the first two rows of the key results. It gives the correlation between the positive sentiment predictions and the human coded values for positive sentiment (Pos Corr) and the Mean Percentage Error (PosMPEnoDiv). The same information is given for negative sentiment.

Pos Corr	NegCorr	PosMPE	NegMPE	PosMPEnoDiv	NegMPEnoDiv
0.638382	0.61354	Ignore this	Ignore this	0.405379	0.32853

If you specified 30 iterations then there will be 31 rows, one for the header and 1 for each iteration. Take the average of the rows as the value to use.

Thanks to Hannes Pirker at OFAI for some of the above code.