# DebugSL: An Interactive Tool for Debugging Sentiment Lexicons

**Andrew T. Schneider, John N. Male, Saroja Bhogadhi, Eduard C. Dragut**
Temple University, Philadelphia, PA
Computer and Information Sciences Department
atschneider, john.male, tug63697, edragut@temple.edu

## Abstract

We introduce DebugSL, a visual (Web) debugging tool for sentiment lexicons (SLs). Its core component implements our algorithms for the automatic detection of polarity inconsistencies in SLs. An inconsistency is a set of words and/or word-senses whose polarity assignments cannot all be simultaneously satisfied. DebugSL finds inconsistencies of small sizes in SLs and has a rich user interface which helps users in the correction process.

## 1 Introduction

The problem of Sentiment Analysis (SA) is that of identifying the polarity (*positive*, *negative*, *neutral*) of the speaker towards the topic of a given piece of text. SA techniques are facilitated by Sentiment Lexicons (SLs), which are lists of words or word-senses tagged with their *a priori* polarity probability values. A tag may be a single value, e.g., *positive*, or it may be a distribution.The large number of SLs and methods to generate them renders errors and disagreements inevitable (Liu, 2015; Feldman, 2013). Numerous works raise the issue of polarity disagreements between SLs and its negative impact on SA tasks (Potts, 2011; Emerson and Declerck, 2014; Liu, 2015). Schneider and Dragut (2015) gives examples of SLs that disagree on up to 78% of their annotations and shows that the accuracy of an SA task can improve by 8.5% by correcting a modest number of inconsistencies in an SL.

Dragut et al. (2012) introduces the Polarity Consistency Problem (PCP) which provides a framework for identifying inconsistent polarity annotations in SLs based on the interaction between words and their underlying shared senses (synsets). Dragut et al. (2015); Dragut and Fellbaum (2014); Schneider and Dragut (2015) further developed the theoretical basis of the PCP.

In this work we present DebugSL, an SL consistency checker and debugger system that implements the methods developed in those works for solving the PCP, in a user friendly environment. Given an SL as input, DebugSL automatically detects entries with potentially incorrect polarity tags and displays these entries in a bipartite graph to facilitate correction efforts. DebugSL interfaces with external sources, such as Dictionary.com, to assist in the debugging process.

## 2 Background

We give a brief overview of the PCP and our methods to solve it in this section. The interested reader is directed to Dragut et al. (2012, 2015); Dragut and Fellbaum (2014); Schneider and Dragut (2015) for full details.

### 2.1 Polarity Probability

Every word and synset is taken to have an underlying discrete probability distribution, called a **polarity distribution**, carrying the *a priori* probability it is used with a positive, negative, or neutral sense, $P_+$, $P_-$ and $P_0$, respectively, with the requirements that $P_+, P_-, P_0 \geq 0$ and $\sum_p P_p = 1$. For instance, the synset *"worthy of reliance or trust"* of the adjective reliable has $P_+ = .375, P_- = .0$ and $P_0 = .625$ in the SL SentiWordNet (Baccianella et al., 2010). The polarity distribution of a word is a weighted sum over the polarity distributions of its senses. For word $w$: $P_p(w) = \sum_{s \in S(w)} f(w,s) \cdot P_p(s)$, where $P_p(s)$ is the polarity value of synset $s$ with polarity $p \in \{+,-,0\}$, $S(w)$ is the set of all synsets of $w$, and $f(w,s)$ denotes the weight between word $w$ and sense $s$. The weights may be the word–synset usage frequencies; they may also be drawn from uniform, Zipfian, or geometric distributions. In all cases the weights are normalized so that $\sum_{s \in S(w)} f(w,s) = 1$ for all $w$.

### 2.2 Word Polarity Value

Word SLs often only give a discrete annotation tag for a word, one of the values: *positive*(+), *negative*(-), or *neutral*(0). We call this a **polarity value**. We determine polarity value for a word as follows:

$$\text{polarity}(w) = \begin{cases} + & \text{if } P_+ > P_- + P_0, \\ - & \text{if } P_- > P_+ + P_0, \\ 0 & \text{otherwise} \end{cases}$$

Schneider and Dragut (2015) discusses alternatives for these equations.

### 2.3 Polarity Consistency

In this context, the PCP amounts to the following question:

*Given a set of words and polarity tags from an SL, does there exist an assignment of polarities to the word senses such that all of the word polarity values agree with the SL tags*?

If the answer is **yes** we say the SL in question is *consistent*. If the answer is **no** we say the SL is ***in**consistent*.

Figure 1 shows a network of 4 words with their annotation tags from (Opinion Finder) OF (Wilson et al., 2005) and their related synsets which comprise a connected component. $w_3$ : *pertinacity* and $w_4$ : *tenacity* are tagged $-$ and $+$, respectively. Since both words share only one synset, $s_3$, there is no polarity distribution for $s_3$ which can simultaneously satisfy the polarity value demands of both $w_3$ and $w_4$. Hence this component is inconsistent.

### 2.4 Solving the PCP via Linear Programming

Using the above definitions, the conversion to an LP problem follows a direct procedure. We refer the interested reader to Schneider and Dragut (2015) for the details.

`DebugSL` supports both discrete and continuous polarity distributions of the synsets. In the discrete case, the synset polarity distributions are restricted to the set $\{0,1\}$, i.e., exactly one of $P_+$, $P_-$, or $P_0$ is 1 and the other two are 0, for each synset; this corresponds to an integer LP problem. For the continuous case, each of $P_+$, $P_-$, or $P_0$ is in the range $[0, 1]$, which corresponds to a general LP problem over real numbers. In the discrete case PCP is NP-complete, while for the continuous case the problem is solvable in polynomial time (Dragut et al., 2012; Schneider and Dragut, 2015).

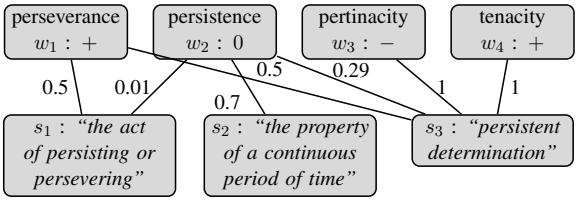



Figure 1: A network of 4 words and 3 synsets. Edges represent word–synset relations, weighted by frequency.

## 3 Supporting SL Debugging

A key capability of `DebugSL` is that of isolating a (small) subset of words/synsets that is polarity inconsistent, but becomes consistent if one element is removed; we call this an Irreducible Polarity Inconsistent Subset (IPIS). Fixing an SL via IPIS isolation proceeds iteratively: (1) isolate an IPIS, (2) determine a repair for this IPIS, (3) if the model is still infeasible, return to step 1. `DebugSL` can deterministically identify an IPIS, but it cannot deterministically decide which inconsistent words and/or senses to adjust as this is simply not an objective decision. Much like a software debugger, which can identify a known programming error, say the use of an undefined variable,but cannot assign a value to the variable, our debugger can identify inconsistent components, but it does not decide which elements to adjust. In Figure 1, minimally one of `pertinacity`(-) and `tenacity`(+) must be adjusted, but the decision as to which requires user feedback.

The example of Figure 1 belies the complexity of the PCP. `DebugSL` employs a divide and conquer approach, dividing an instance of PCP into the connected components of the bipartite word–synset graph, then solving for each component separately. Running `DebugSL` on OF generates 1178 such components for adjectives alone; the largest component has 914 unique words and 1880 unique synsets. Manually checking such SLs is unrealistic.

## 4 System Overview

`DebugSL` follows a four-step procedure to identify and reduce lexicon inconsistency. (1) The user uploads a formatted SL to the `DebugSL` system. (2) A server-side program uses the lexical database WordNet(WN) (Fellbaum, 1998) to form the underlying word–synset graph of the SL and checks for inconsistencies. (3) Inconsistent components are returned to the client and they are displayed

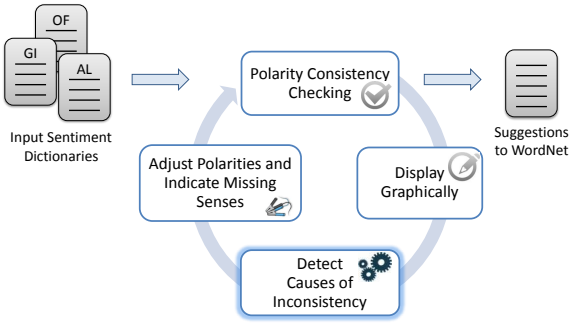



Figure 2: DebugSL process flow

graphically for inspection. (4) The user dynamically interacts with the components, such as looking up information on the words/synsets and adjusting their polarities. The user can repeat the process as desired.

**1. Input.** The user begins by uploading an SL to DebugSL and specifying various interface options, such as the part of speech and the version of WN to use. After the lexicon has been successfully uploaded, the user can download the (modified) lexicon, or save it in the system.

**2. Lexicon Inconsistency Checking.** A server-side Java program builds the connected components of word–synset linkage. DebugSL checks the consistency of each connected component. Inconsistent components are returned to the client.

**3. Component Display.** A JavaScript process receives the inconsistent components from the server and formats them to be understood by the graphing library SigmaJS which displays the inconsistent components to the user.

**4. Interactive Viewing and User Analysis.** The interactive viewing is essential to our tool because it allows a user to focus on a small set of inconsistent words or synsets, make adjustments, and re-run the program to see the effects.

## 5 Software

DebugSL is developed using web technologies. Client side work is completed using HTML 5, CSS, and JavaScript. The graphs are structured using the open source JavaScript library SigmaJS. The client-side program is programmed in Java and is called through a Java Servlet. For linear programming DebugSL employs the GUROBI LP solver (www.gurobi.com). Apache Tomcat is used as the local server.

### 5.1 Interactive Features

**Customizable Polarity Display.** DebugSL displays the word polarities from the uploaded SL on the word–synset graph. By default, the polarities are shown by node color: green, red, and grey for *positive*, *negative*, and *neutral*, respectively. These color choices are customizable. The user can switch to display polarities by symbols: $+$, $-$, and 0, instead, or use both symbols and colors.

**Component and Word Selection.** Inconsistent components are listed on the left represented by the first word of each component. An ellipsis indicates the presence of additional words. When a component is selected, its words are expanded into a sublist (Figure 3a on the left).

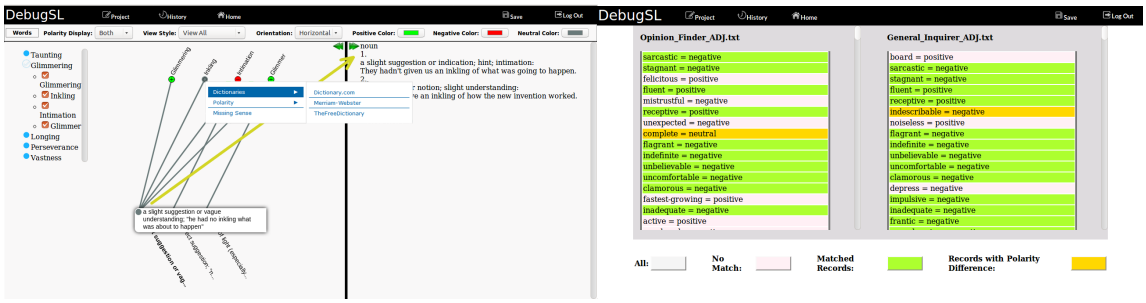
**All Viewing and Progressive Viewing.** Two modes of viewing the graph are available. In *All Viewing* mode all checked words and synsets are visible. Clicking on a word or synset hides any edges not associated with that word or synset.

*Progressive Viewing* mode allows the user to build the desired connected graph by progressively adding words. Clicking a word node hides other words and reveals its associated synsets. Clicking a revealed synset hides all but the associated words and edges. The user may use any selected word or synset as a base for building the graph. Holding shift allows words to be added progressively.

**Dictionary Query.** The user may look up the senses of any word from an online dictionary by right-clicking. This functionality is currently implemented to interface with three online dictionaries: Merriam-Webster (www.merriam-webster.com), Dictionary.com (www.dictionary.com), and the Free Dictionary (www.thefreedictionary.com). The dictionary appears to the right of the graph window with the results of the word lookup. Right-clicking a new word, automatically updates the dictionary area.

### 5.2 Identifying Missing Senses in WordNet

In some cases, when an inconsistency is identified, the user may decide that the polarity assignments are correct and the error is in fact due to a missing sense in WN. DebugSL allows the user to check for potentially missing senses in WN. As an example, the verbs “tantalize” and “taunt,” have positive and negative polarities, respectively, in Opinion Finder (Wilson et al., 2005). They also have a shared, unique sense in WN. By our formula-




(a) DebugSL debugging (b) DebugSL comparison

Figure 3: DebugSL screenshots

tion, this leads to a contradiction. In this case the Free Dictionary gives a second sense of tantalize that is missing from WN: *"to be strongly attractive to..."*. This sense conveys a positive polarity. Hence, tantalize conveys a positive sentiment when used with this sense, and the inconsistency is not due to a mislabeling in the SL.

When the user utilizes the online dictionary lookup feature, the dictionary senses (definitions) of the word are automatically matched to their corresponding best matching synsets in the WN graph, using the Levenshtein edit distance and the Gale-Shapley stable marriage matching algorithm (Gale and Shapley, 1962). If a relevant sense appears to be missing from WN, the user can log a note of the missing synset with us and we can, in turn, provide this information to the WN team (Dragut and Fellbaum, 2014).

## 6 Demonstration

We present two main scenarios to demonstrate the practical usefulness of DebugSL, screenshots of which are shown in Figure 3. The source code is available at https://github.com/atschneid/DebugSL.

**1. Iterative SL Debugging:** This commences with a user uploading an SL, $\mathcal{L}$. The user must turn a few knobs, e.g., select the part of speech, weighting scheme, and WN version. DebugSL displays the discovered sets of polarity inconsistencies on the left side. Let $\mathcal{I}$ be an IPIS the user has selected. This demo scenario has two paths:

**Debugging.** The user attempts to correct polarity values assigned to the entries in $\mathcal{I}$. Most IPISs consist of up to 4 words; very few have more than 8. This aids in holding the user's focus. The user can use the bipartite view in DebugSL to analyze the words and their sense and determine if some entries in $\mathcal{I}$ have wrong polarity tags. Upon identifying a mislabeled entry in $\mathcal{I}$, the user can edit its polarity value (see Figure 3a), then repeat the consistency check again on the revised SL, and analyze any new or remaining IPISs. At any time, the user can save, revisit, and eventually download the revised version of $\mathcal{L}$.

**Missing Senses.** If all the entries in $\mathcal{I}$ have the correct polarity tags, there remains the possibility that WN has incomplete information. DebugSL allows the user to compare the synsets of a word $w$ in $\mathcal{I}$ with those of $w$ in several online dictionaries (Figure 3a, see the popup menu in the center). The user selects a dictionary to reference and the senses of $w$ from this dictionary will appear beside the graph. DebugSL matches the WN synsets with the dictionary senses (see Figure 3). The user can suggest via DebugSL a sense present in Dictionary.com for $w$ that is missing in WN. We store all the suggestions.

**2. Comparing Two SLs:** This second demo scenario uncovers the disagreement between two SLs: typically one word annotating, $S_w$, (e.g., OF) and one synset annotating, $S_s$, (e.g., SentiWordNet). The formal procedure for this functionality is described at length in Schneider and Dragut (2015). The output is the collection of graphs of components with inconsistencies. DebugSL can also compare two word annotating SLs or two sense annotating SLs wherein DebugSL checks for agreement.

## 7 Conclusion

In this paper we have presented the system DebugSL and described its usage. The project source code is available at https://github.com/atschneid/DebugSL and a screencast can be viewed at https://cis.temple.edu/~edragut/DebugSL.webm. The system will be deployed online for use by the public.

## References


Stefano Baccianella, Andrea Esuli, and Fabrizio Sebastiani. 2010. SentiWordNet 3.0: An Enhanced Lexical Resource for Sentiment Analysis and Opinion Mining. In *LREC*.

Eduard Dragut and Christiane Fellbaum. 2014. The role of adverbs in sentiment analysis. In *Proceedings of Frame Semantics in NLP: A Workshop in Honor of Chuck Fillmore (1929-2014)*.

Eduard Dragut, Hong Wang, Prasad Sistla, Clement Yu, and Weiyi Meng. 2015. Polarity consistency checking for domain independent sentiment dictionaries. *IEEE TKDE*, 27(3):838–851.

Eduard C. Dragut, Hong Wang, Clement Yu, Prasad Sistla, and Weiyi Meng. 2012. Polarity consistency checking for sentiment dictionaries. In *ACL*.

Guy Emerson and Thierry Declerck. 2014. Sentimerge: Combining sentiment lexicons in a bayesian framework. In *workshop at COLING*.

Ronen Feldman. 2013. Techniques and applications for sentiment analysis. *Commun. ACM*, 56(4).

Christiane Fellbaum. 1998. *WordNet: An On-Line Lexical Database and Some of its Applications.* MIT Press.

David Gale and Lloyd S. Shapley. 1962. College admissions and the stability of marriage. *AMM*, 69.

Bing Liu. 2015. *Sentiment Analysis: Mining Opinions, Sentiments, and Emotions*. Cambridge University Press.

Christopher Potts. 2011. Sentiment symposium tutorial: Lexicons.

Andrew T. Schneider and Eduard C. Dragut. 2015. Towards debugging sentiment lexicons. In *ACL*.

T. Wilson, J. Wiebe, and P. Hoffmann. 2005. Recognizing contextual polarity in phrase-level sentiment analysis. In *HLT/EMNLP*.