LEXenstein: A Framework for Lexical Simplification

Gustavo Henrique Paetzold and **Lucia Specia**
Department of Computer Science
University of Sheffield, UK
{ghpaetzold1,l.specia}@sheffield.ac.uk

Abstract

Lexical Simplification consists in replacing complex words in a text with simpler alternatives. We introduce LEXenstein, the first open source framework for Lexical Simplification. It covers all major stages of the process and allows for easy benchmarking of various approaches. We test the tool's performance and report comparisons on different datasets against the state of the art approaches. The results show that combining the novel Substitution Selection and Substitution Ranking approaches introduced in LEXenstein is the most effective approach to Lexical Simplification.

1 Introduction

The goal of a Lexical Simplification (LS) approach is to replace complex words and expressions in a given text, often a sentence, with simpler alternatives of equivalent meaning in context. Although very intuitive, this is a challenging task since the substitutions must preserve both the original meaning and the grammaticality of the sentence being simplified.

The LS task has been gaining significant attention since the late 1990's, thanks to the positive influence of the early work presented by (Devlin and Tait, 1998) and (Carroll et al., 1999). More recently, the LS task at SemEval-2012 (Specia et al., 2012) has given LS wider visibility. Participants had the opportunity to compare their approaches in the task of ranking candidate substitutions, all of which were already known to fit the context, according to their "simplicity".

Despite its growth in popularity, the inexistence of tools to support the process and help researchers to build upon has been hampering progress in the area. We were only able to find one tool for LS: a set of scripts designed for the training and testing of ranking models provided by (Jauhar and Specia, 2012)[1]. However, they cover only one step of the process. In an effort to tackle this issue, we present LEXenstein: a framework for Lexical Simplification development and benchmarking.

LEXenstein is an easy-to-use framework that provides simplified access to many approaches for several sub-tasks of the LS pipeline, which is illustrated in Figure 1. Its current version includes methods for the three main sub-tasks in the pipeline: Substitution Generation, Substitution Selection and Substitution Ranking.

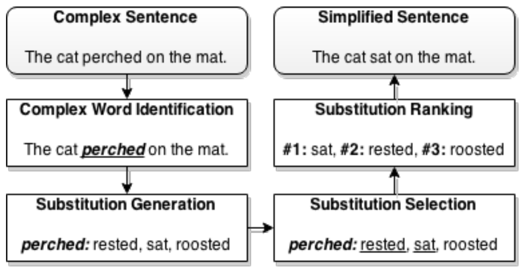
Complex Sentence
The cat perched on the mat.
Complex Word Identification
The cat *perched* on the mat.
Substitution Generation
perched: rested, sat, roosted
Substitution Selection
perched: rested, sat, roosted
Substitution Ranking
#1: sat, #2: rested, #3: roosted
Simplified Sentence
The cat sat on the mat.

Figure 1: Lexical Simplification Pipeline

LEXenstein was devised to facilitate performance comparisons among various LS approaches, as well as the creation of new strategies for LS. In the following Sections we present LEXenstein's components (Section 2) and discuss the results of several experiments conducted with the tool (Section 3).

2 System Overview

LEXenstein is a Python library that provides several approaches for sub-tasks in LS. To increase its flexibility, the library is structured in six modules: Substitution Generation, Substitution Selection, Substitution Ranking, Feature Estimation, Evaluation and Text Adorning. In the following Sections, we describe them in more detail.

[1]https://github.com/sjauhar/simplex

2.1 Substitution Generation

We define Substitution Generation (SG) as the task of producing candidate substitutions for complex words, which is normally done regardless of the context of the complex word. Previous work commonly addresses this task by querying general domain thesauri such as WordNet (Fellbaum, 1998), or domain specific ones such as UMLS (Bodenreider, 2004). Examples of work resorting to this strategy are (Devlin and Tait, 1998) and (Carroll et al., 1999). Recent work focuses on learning substitutions from sentence-aligned parallel corpora of complex-simple texts (Paetzold and Specia, 2013; Horn et al., 2014).

LEXenstein's SG module offers support for five approaches. All approaches use LEXenstein's Text Adorning module to create substitutions for all possible inflections of verbs and nouns. Each approach is represented by one of the following Python classes:

KauchakGenerator (Horn et al., 2014) Automatically extracts substitutions from parallel corpora. It requires a set of tagged parallel sentences and the word alignments between them in Pharaoh format (Och and Ney, 2000). It produces a dictionary of complex-to-simple substitutions filtered by the criteria described in (Horn et al., 2014).

BiranGenerator (Biran et al., 2011) Filters substitutions based on the Cartesian product between vocabularies of complex and simple words. It requires vocabularies of complex and simple words, as well as two language models trained over complex and simple corpora. It produces a dictionary linking words to a set of synonyms and hypernyms filtered by the criteria described in (Biran et al., 2011).

YamamotoGenerator (Kajiwara et al., 2013) Extracts substitutions from dictionary definitions of complex words. It requires an API key for the Merriam Dictionary[2], which can be obtained for free. It produces a dictionary linking words in the Merriam Dictionary and WordNet to words with the same Part-of-Speech (POS) tag in its entries' definitions and examples of usage.

MerriamGenerator Extracts a dictionary linking words to their synonyms, as listed in the Merriam Thesaurus. It requires an API key.

[2]http://www.dictionaryapi.com/

WordnetGenerator Extracts a dictionary linking words to their synonyms, as listed in WordNet.

2.2 Substitution Selection

Substitution Selection (SS) is the task of selecting which substitutions – from a given list – can replace a complex word in a given sentence without altering its meaning. Most work addresses this task referring to the context of the complex word by employing Word Sense Disambiguation (WSD) approaches (Sedding and Kazakov, 2004; Nunes et al., 2013), or by discarding substitutions which do not share the same POS tag of the target complex word (Kajiwara et al., 2013; Paetzold and Specia, 2013).

LEXenstein's SS module provides access to three approaches. All approaches require as input a dictionary of substitutions generated by a given approach and a dataset in the VICTOR format (as in Victor Frankenstein (Shelley, 2007)). As output, they produce a set of selected substitutions for each entry in the VICTOR dataset. The VICTOR format is structured as illustrated in Example 1, where S_i is the ith sentence in the dataset, w_i a target complex word in the h_ith position of S_i, c_i^j a substitution candidate and r_i^j its simplicity ranking. Each bracketed component is separated by a tabulation marker.

$$\begin{bmatrix} \langle \mathrm{S}_1 \rangle & \langle \mathrm{w}_1 \rangle & \langle \mathrm{h}_1 \rangle & \langle \mathrm{r}_1^1{:}\mathrm{c}_1^1 \rangle \cdots \langle \mathrm{r}_1^n{:}\mathrm{c}_1^n \rangle \\ & & \vdots & \\ \langle \mathrm{S}_m \rangle & \langle \mathrm{w}_m \rangle & \langle \mathrm{h}_m \rangle & \langle \mathrm{r}_m^1{:}\mathrm{c}_m^1 \rangle \cdots \langle \mathrm{r}_m^n{:}\mathrm{c}_m^n \rangle \end{bmatrix} \quad (1)$$

LEXenstein includes two resources for training/testing in the VICTOR format: the LexMTurk (Horn et al., 2014) and the SemEval corpus (Specia et al., 2012). Each approach in the SS module is represented by one of the following Python classes:

WSDSelector Allows for the user to use one among various classic WSD approaches in SS. It requires the PyWSD (Tan, 2014) module to be installed, which includes the approaches presented by (Lesk, 1986) and (Wu and Palmer, 1994), as well as baselines such as random and first senses.

BiranSelector (Biran et al., 2011) Employs a strategy in which a word co-occurrence model is used to determine which substitutions have meaning similar to that of a target complex word. It requires a plain text file with each line in the format specified in Example 2, where $\langle w_i \rangle$ is a word,

$\left\langle c_i^j \right\rangle$ a co-occurring word and $\left\langle f_i^j \right\rangle$ its frequency of occurrence.

$$\langle w_i \rangle \ \langle c_i^0 \rangle : \langle f_i^0 \rangle \cdots \langle c_i^n \rangle : \langle f_i^n \rangle \qquad (2)$$

Each component in the format in 2 must be separated by a tabulation marker. Given such a model, the approach filters all substitutions which are estimated to be more complex than the target word, and also those for which the distance between their co-occurrence vector and the target sentence's vector is higher than a threshold set by the user.

WordVectorSelector Employs a novel strategy, in which a word vector model is used to determine which substitutions have the closest meaning to that of the sentence being simplified. It requires a binary word vector model produced by Word2Vec[3], and can be configured in many ways. It retrieves a user-defined percentage of the substitutions, which are ranked with respect to the cosine distance between their word vector and the sum of some or all of the sentences' words, depending on the settings defined by the user.

2.3 Substitution Ranking

Substitution Ranking (SR) is the task of ranking a set of selected substitutions for a target complex word with respect to their simplicity. Approaches vary from simple word length and frequency-based measures (Devlin and Tait, 1998; Carroll et al., 1998; Carroll et al., 1999; Biran et al., 2011) to more sophisticated linear combinations of scoring functions (Jauhar and Specia, 2012), as well as machine learning-based approaches (Horn et al., 2014).

LEXenstein's SR module provides access to three approaches. All approaches receive as input datasets in the VICTOR format, which can be either training/testing datasets already containing only valid substitutions in context, or datasets generated with (potentially noisy) substitutions by a given SS approach. They also require as input a FeatureEstimator object to calculate feature values describing the candidate substitutes. More details on the FeatureEstimator class are provided in Section 2.4. Each approach in the SR module is represented by one of the following Python classes:

[3]https://code.google.com/p/word2vec/

MetricRanker Employs a simple ranking strategy based on the values of a single feature provided by the user. By configuring the input FeatureEstimator object, the user can calculate values of several features for the candidates in a given dataset and easily rank the candidates according to each of these features.

SVMRanker (Joachims, 2002) Use Support Vector Machines in a setup that minimises a loss function with respect to a ranking model. This strategy is the one employed in the LS experiments of (Horn et al., 2014), yielding promising results. The user needs to provide a path to their SVM-Rank installation, as well as SVM-related configurations, such as the kernel type and parameter values for C, $epsilon$, etc.

BoundaryRanker Employs a novel strategy, in which ranking is framed as a binary classification task. During training, this approach assigns the label 1 to all candidates of rank $1 \geq r \geq p$, where p is a range set by the user, and 0 to the remaining candidates. It then trains a stochastic descent linear classifier based on the features specified in the FeatureEstimator object. During testing, candidate substitutions are ranked based on how far from 0 they are. This ranker allows the user to provide several parameters during training, such as loss function and penalty type.

2.4 Feature Estimation

LEXenstein's Feature Estimation module allows the calculation of several features for LS-related tasks. Its class FeatureEstimator allows the user to select and configure many features commonly used by LS approaches.

The FeatureEstimator object can be used either for the creation of LEXenstein's rankers, or in stand-alone setups. For the latter, the class provides a function called *calculateFeatures*, which produces a matrix MxN containing M feature values for each of the N substitution candidates listed in the dataset. Each of the 11 features supported must be configured individually. They can be grouped in four categories:

Lexicon-oriented: Binary features which receive value 1 if a candidate appears in a given vocabulary, and 0 otherwise.

Morphological: Features that exploit morphological characteristics of substitutions, such as word length and number of syllables.

Collocational: N-gram probabilities of the form $P\left(S_{h-1}^{h-l}\, c\, S_{h+1}^{h+r}\right)$, where c is a candidate substitution in the hth position in sentence S, and S_{h-1}^{h-l} and S_{h+1}^{h+r} are n-grams of size l and r, respectively.

Sense-oriented: Several features which are related to the meaning of a candidate substitution such as number of senses, lemmas, synonyms, hypernyms, hyponyms and maximum and minimum distances among all of its senses.

2.5 Evaluation

Since one of the goals of LEXenstein is to facilitate the benchmarking LS approaches, it is crucial that it provides evaluation methods. This module includes functions for the evaluation of all subtasks, both individually and in combination. It contains four Python classes:

GeneratorEvaluator: Provides evaluation metrics for SG methods. It requires a gold-standard in the VICTOR format and a set of generated substitutions. It returns the Potential, Precision and F-measure, where Potential is the proportion of instances for which at least one of the substitutions generated is present in the gold-standard, Precision the proportion of generated instances which are present in the gold-standard, and F-measure their harmonic mean.

SelectorEvaluator: Provides evaluation metrics for SS methods. It requires a gold-standard in the VICTOR format and a set of selected substitutions. It returns the Potential, Precision and F-measure of the SS approach, as defined above.

RankerEvaluator: Provides evaluation metrics for SR methods. It requires a gold-standard in the VICTOR format and a set of ranked substitutions. It returns the TRank-at-1:3 and Recall-at-1:3 metrics (Specia et al., 2012), where Trank-at-i is the proportion of instances for which a candidate of gold-rank $r \leq i$ was ranked first, and Recall-at-i the proportion of candidates of gold-rank $r \leq i$ that are ranked in positions $p \leq i$.

PipelineEvaluator: Provides evaluation metrics for the entire LS pipeline. It requires as input a gold-standard in the VICTOR format and a set of ranked substitutions which have been generated and selected by a given set of approaches. It returns the approaches' Precision, Accuracy and Change Proportion, where Precision is the proportion of instances for which the highest ranking substitution is not the target complex word itself and is in the gold-standard, Accuracy is the proportion of instances for which the highest ranking substitution is in the gold-standard, and Change Proportion is the proportion of instances for which the highest ranking substitution is not the target complex word itself.

2.6 Text Adorning

This approach provides a Python interface to the Morph Adorner Toolkit (Paetzold, 2015), a set of Java tools that facilitates the access to Morph Adorner's functionalities. The class provides easy access to word lemmatisation, word stemming, syllable splitting, noun inflection, verb tensing and verb conjugation.

2.7 Resources

LEXenstein also provides a wide array of resources for the user to explore in benchmarking tasks. Among them are the aforementioned LexMturk and SemEval corpora in the VICTOR format, lists of stop and basic words, as well as language models and lexica built over Wikipedia and Simple Wikipedia.

3 Experiments

In this Section, we discuss the results obtained in four benchmarking experiments.

3.1 Substitution Generation

In this experiment we evaluate all SG approaches in LEXenstein. For the KauchakGenerator, we use the corpus provided by (Kauchak, 2013), composed of $150,569$ complex-to-simple parallel sentences, parsed by the Stanford Parser (Klein and Manning, 1965). From the the same corpus, we build the required vocabularies and language models for the BiranGenerator. We used the LexMturk dataset as the gold-standard (Horn et al., 2014), which is composed by 500 sentences, each with a single target complex word and 50 substitutions suggested by turkers. The results are presented in Table 1.

The results in Table 1 show that the method of (Horn et al., 2014) yields the best F-Measure results, although combining the output of all generation methods yields the highest Potential. This shows that using parallel corpora to generate substitution candidates for complex words can be a

Approach	Pot.	Prec.	F
Kauchak	0.830	**0.155**	**0.262**
Wordnet	0.608	0.109	0.184
Biran	0.630	0.102	0.175
Merriam	0.540	0.067	0.120
Yamamoto	0.504	0.054	0.098
All	**0.976**	0.066	0.124

Table 1: SG benchmarking results

more efficient strategy than querying dictionaries and databases. We must, however, keep in mind that the sentences that compose the LexMturk corpus were extracted from Wikipedia, which is the same corpus from which the KauchakGenerator learns substitutions.

3.2 Substitution Selection

Here we evaluate of all SS approaches in LEXenstein. For the BiranSelector, we trained a co-occurrence model over a corpus of 6+ billion words extracted from the various sources suggested in the Word2Vec documentation[4], the same sources over which the word vector model required by the WordVectorSelector was trained. In order to summarise the results, we present the scores obtained only with the best performing configurations of each approach. The LexMturk corpus is used as the gold-standard, and the initial set of substitutions is the one produced by all SG approaches combined. The results are presented in Table 2.

Approach	Pot.	Prec.	F	Size
Word Vec.	0.768	**0.219**	**0.341**	3,042
Biran	0.508	0.078	0.136	9,680
First	0.176	0.045	0.072	2,471
Lesk	0.246	0.041	0.070	4,716
Random	0.082	0.023	0.035	2,046
Wu-Pa	0.038	0.013	0.020	1,749
No Sel.	**0.976**	0.066	0.124	26,516

Table 2: SS benchmarking results

"Size" in Table 2 represents the total number of substitutions selected for all test instances. The results in Table 2 show that our novel word vector approach outperforms all others in F-Measure by a considerable margin, including the method of not performing selection at all. Note that not performing selection allows for Potential to be higher, but yields very poor Precision.

[4]https://code.google.com/p/word2vec/

3.3 Substitution Ranking

In Table 3 we present the results of the evaluation of several SR approaches. We trained the SVMRanker with features similar to the ones used in (Horn et al., 2014), and the BoundaryRanker with a set of 10 features selected through univariate feature selection. We compare these approaches to three baseline Metric Rankers, which use the word's frequency in Simple Wikipedia, its length or its number of senses. The SemEval corpus is used as the gold-standard so that we can compare our results with the best one obtained at SemEval-2012 (Jauhar and Specia, 2012) (SemEval, in Table 3).

Approach	TR-1	Rec-1	Rec-2	Rec-3
Boundary	**0.655**	**0.608**	0.602	0.663
SVM	0.486	0.451	0.502	0.592
Freq.	0.226	0.220	0.236	0.300
Length	0.180	0.175	0.200	0.261
Senses	0.130	0.126	0.161	0.223
SemEval	0.602	0.575	**0.689**	**0.769**

Table 3: SR benchmarking results

The novel Boundary ranking approach outperforms all other approaches in both TRank-at-1 and Recall-at-1 by a considerable margin, but it is worse than the best SemEval-2012 approach in terms of Recall-at-2 and 3. This however reveals not a limitation but a strength of our approach: since the Boundary ranker focuses on placing the best substitution in the highest rank, it becomes more effective at doing so as opposed to at producing a full ranking for all candidates.

3.4 Round-Trip Evaluation

In this experiment we evaluate the performance of different combinations of SS and SR approaches in selecting suitable substitutions for complex words from the ones produced by all generators combined. Rankers and selectors are configured in the same way as they were in the experiments in Sections 3.3 and 3.2. The gold-standard used is LexMturk, and the performance metric used is the combination's Precision: the proportion of times in which the candidate ranked highest is not the target complex word itself and belongs to the gold-standard list. Results are shown in Table 4.

The results show that combining the WordVectorSelector with the BoundaryRanker yields

	No Sel.	Word Vector	Biran
Boundary	0.342	**0.550**	0.197
SVM	0.108	0.219	0.003
Freq.	0.114	0.501	0.096
Length	0.120	0.408	0.092
Senses	0.214	0.448	0.122

Table 4: Round-trip benchmarking results

the highest performance in the pipeline evaluation. Interestingly, the SVMRanker, which performed very well in the individual evaluation of Section 3.3, was outperformed by all three baselines in this experiment.

4 Final Remarks

We have presented LEXenstein, a framework for Lexical Simplification distributed under the permissive BSD license. It provides a wide arrange of useful resources and tools for the task, such as feature estimators, text adorners, and various approaches for Substitution Generation, Selection and Ranking. These include methods from previous work, as well as novel approaches. LEXenstein's modular structure also allows for one to easily add new approaches to it.

We have conducted evaluation experiments including various LS approaches in the literature. Our results show that the novel approaches introduced in this paper outperform those from previous work. In the future, we intend to incorporate in LEXenstein approaches for Complex Word Identification, as well as more approaches for the remaining tasks of the usual LS pipeline.

The tool can be downloaded from: `http://ghpaetzold.github.io/LEXenstein/`.

References

O. Biran, S. Brody, and N. Elhadad. 2011. Putting it Simply: a Context-Aware Approach to Lexical Simplification. *The 49th Annual Meeting of the ACL.*

O. Bodenreider. 2004. The unified medical language system (umls): integrating biomedical terminology. *Nucleic acids research.*

J. Carroll, G. Minnen, Y. Canning, S. Devlin, and J. Tait. 1998. Practical simplification of english newspaper text to assist aphasic readers. In *The 15th AAAI.*

J. Carroll, G. Minnen, D. Pearce, Y. Canning, S. Devlin, and J. Tait. 1999. Simplifying Text for Language Impaired Readers. *The 9th EACL.*

S. Devlin and J. Tait. 1998. The use of a psycholinguistic database in the simplification of text for aphasic readers. *Linguistic Databases.*

C. Fellbaum. 1998. *WordNet: An Electronic Lexical Database.* Bradford Books.

C. Horn, C. Manduca, and D. Kauchak. 2014. Learning a Lexical Simplifier Using Wikipedia. *The 52nd Annual Meeting of the ACL.*

S.K. Jauhar and L. Specia. 2012. UOW-SHEF: SimpLex–lexical simplicity ranking based on contextual and psycholinguistic features. *The 1st *SEM.*

T. Joachims. 2002. Optimizing search engines using clickthrough data. In *The 8th ACM.*

T. Kajiwara, H. Matsumoto, and K. Yamamoto. 2013. Selecting Proper Lexical Paraphrase for Children.

D. Kauchak. 2013. Improving Text Simplification Language Modeling Using Unsimplified Text Data. *The 51st Annual Meeting of the ACL.*

D. Klein and C.D. Manning. 1965. Accurate Unlexicalized Parsing. In *The 41st Annual Meeting of ACL.*

M. Lesk. 1986. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. In *The 5th SIGDOC.*

B.P. Nunes, R. Kawase, P. Siehndel, M.A. Casanova, and S. Dietze. 2013. As Simple as It Gets - A Sentence Simplifier for Different Learning Levels and Contexts. *IEEE 13th ICALT.*

F.J. Och and H. Ney. 2000. Improved statistical alignment models. In *The 38th Annual Meeting of the ACL.*

G.H. Paetzold and L. Specia. 2013. Text simplification as tree transduction. In *The 9th STIL.*

G.H. Paetzold. 2015. Morph adorner toolkit: Morph adorner made simple. http://ghpaetzold.github.io/MorphAdornerToolkit/.

J. Sedding and D. Kazakov. 2004. Wordnet-based text document clustering. In *The 3rd ROMAND.*

M. Shelley. 2007. *Frankenstein.* Pearson Education.

L. Specia, S.K. Jauhar, and R. Mihalcea. 2012. Semeval-2012 task 1: English lexical simplification. In *The 1st *SEM.*

L. Tan. 2014. Pywsd: Python implementations of word sense disambiguation technologies. https://github.com/alvations/pywsd.

Z. Wu and M. Palmer. 1994. Verbs semantics and lexical selection. In *The 32nd Annual Meeting of ACL.*