NLP Final Project Report
Unsupervised Lexical Simplification

By Ayala Raanan

1. Background

Lexical simplification (LS) is the task of replacing complex words in a text by simple ones, while maintaining the original meaning of the text. This type of simplification is potentially useful for non-native readers of the language, for children, and for people suffering from various language impairments, such as aphasia [1].

The typical LS process consists of the following steps: identifying the complex words in a sentence; finding candidates for replacement; filtering the candidates according to various criteria; and finally ranking the candidates and returning the best options for substitution.

For example, when considering the sentence "The cat perched on the mat", we expect the LS method to identify the word 'perched' as complex, find 'sat' as the best candidate for a simple replacement and return the sentence "The cat sat on the mat".

Each of the steps mentioned above has its own challenges and set of tools.
Identifying complex words usually relies on word-frequency statistics and even the length of the word is sometimes used as an indicator.
The simplest solution that comes to mind when trying to find candidates for substitution is using a sort of a thesaurus. While such resources are expensive and not always comprehensive enough, several other problems arise from this approach. The main problem is that a word may have many substitutions in different contexts. A good substitution in one sentence may be semantically or even grammatically wrong in another sentence. A radical example is polysemy, where a synonym may belong to an entirely different world of concepts.
As a result, a bank of candidates is often collected automatically using some corpora, but many of them would not be fit replacements. A set of tools is then used to superficially filter unfit candidates. One popular tool is POS tagging, where we only accept replacements classified as the same part-of-speech. Another elementary filter is removing candidates sharing the same stem with the original word.
All of this still does not solve the problem of lacking context for the replacement. Several approaches will be mentioned in the related work section for producing context features.

Some powerful tools have become available in recent years, which enable us to tackle the problem of LS using a totally unsupervised approach. Pre-trained word embeddings can be used to suggest candidates for replacement according to similarities in the embedding space. Pre-trained transformer-based language models like BERT enable us to inspect word representations in context. Such representations should help choose the candidates with the correct connotation.

In this work we investigate the use of pre-trained tools and their combination for unsupervised simplification of words in context. We show that our simple approach surpasses the state of the art results on three lexical simplification datasets.

Due to the nature of the datasets, we focus our work on finding the best replacements and skip identification of complex words.

2. Related Work

Early work used professional databases such as thesauri, or WordNet for generating substitution candidates. A set of rules was learnt and applied in the form of word - substitution pairs. Word frequencies and other basic features were used to decide if and when to apply substitution rules.

Then came additional useful resources of simplified corpora, where a source and its simplified version are available. Popular examples are English Wikipedia (EW) together with Simplified English Wikipedia (SEW) and Newsela. These databases can be processed to automatically produce aligned pairs of a complex and a simple version of a sentence. One can learn both substitution rules and context rules from the aligned pairs. Horn et al. (2014) used aligned sentences from EW to SEW to suggest candidates. They then applied machine learning techniques (SVM) to learn a ranking function from extracted features of the aligned sentences.

The advancement in word embedding models, which require only a large corpora to be trained, has introduced a new option for substitution suggestion. Candidate words could be retrieved according to similarities in the embedding space. Glavaš and Štajner(2015) used the GloVe model to produce a vector representation for all words in the corpus. An embedding space representation however, still lacks context information. Glavaš and Štajner constructed a context indicator by averaging similarities of the candidate with other words in the original sentence within a window of the original word. This indicator was then used as a dominant feature for replacement ranking.

Paetzold and Specia (2016) also use word embedding, but train a special word embedding, also incorporating information on the POS tagging of the data. This constructs a partial context aware embedding. They introduce a "Boundary Ranking" method for learning a ranking decision boundary over a space of given features.

The breakthroughs in transformer models, specifically with the release of pre-trained BERT (Devlin et al. 2018) were a game changer regarding tasks involving embedding in context. BERT is trained to predict masked words and can therefore produce a suggestion distribution to a masked word in context. Qiang, Jipeng, et al. (2020) exploited this ability and had the model itself suggest candidates to a masked complex word. In order to retain the meaning of the full sentence, they also used the 'next sentence prediction' ability of the model. The original unmasked sentence was placed as the first sentence, and the masked sentence second. The great advantage of this method is that the suggested words are already in context. In all other methods, the set of suggested words is first taken from a space without context. This usually means that a large number of spurious candidates are included. While these can be filtered out, the process can consume time and resources.

Our work examines the bridge between the dominant methods. We suggest candidates using an embedding space, filter them, and finally evaluate their validity in context using BERT.

3. Database

Our ability to evaluate the performance of our algorithm relies on an annotated dataset. An instance of such a dataset includes a sentence, a complex word chosen in this sentence and its position, and a set of replacement candidates with their ranking (the gold standards). The objective of the model is to cover the set of gold candidates as faithfully as possible. Table 1 shows an example of such an instance.

There are 3 widely used LS datasets:

- **LexMTurk** (Horn et al. 2014), which is composed of 500 English sentences taken from Wikipedia. Each sentence has a target complex word and 50 substitutions annotated by 50 Amazon Mechanical "turkers".

Table 1: Instance taken from BenchLS

Sentence	'Kowal suggested the name and the IAU endorsed it in 1975.'
Target word & position	'endorsed', 7
Gold candidates	1:supported, 2:approved, 3:accepted, 4:backed, 4:okayed, 4:authorized, 5:favored, 5:recommended, 5:passed, 5:adopted, 5:ratified

- **BenchLS** (Paetzold and Specia 2016) is the largest dataset with 929 entries combining instances from LexMTurk and LSeval (De Belder and Moens 2012). The LSeval dataset contains 429 instances, annotated by both turkers and 9 Ph.D. students.
- **NNSeval** (Paetzold and Specia 2017b), with 239 instances, is a filtered version of BenchLS. This was the result of a study of the needs of non-native English speakers, which led to a refined version of BenchLS. Instances were removed either where the marked word was not considered complex, or where candidate words were too complex, according to the 400 non-native speakers in the study.

There are some obvious anomalies in the datasets, such as: suggestions marked simply as **pn** (probably for proper noun); suggesting two replacements for 'struck' as 'hit' and **'nit'**; suggesting two replacements for 'deceased' as 'dead' or **'building'**, etc. There are cases where it is obvious that the model eventually does a better job than the annotators. However, these datasets remain our point of reference to other papers.

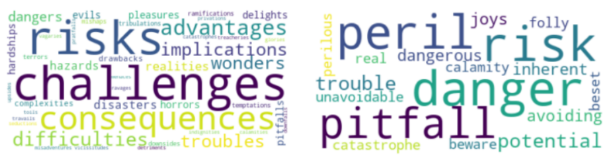
4. Algorithm

Our simplified pseudocode is given in Alg.1. It consists of the following stages:

4.1 **Finding close words** to the marked complex word by similarities in the embedding space. First, an embedding space needs to be chosen. Section 6.1 describes our experiment comparing GloVe and Fasttext in order to choose the best suited embedding space. The embedding tokenizer is used to obtain the vector representation of the target word in the embedding space. We make sure all target words are indeed present in the embedding vocabulary. We can then calculate similarities to all other words in the embedding space. The most common similarity used is cosine similarity. We explored the advantages of combining L2 similarity and found that it was very useful for better coverage of the gold standards. The comparison can be found in 6.2. Stage 1 returns a list of candidates combining closest candidates from each similarity measure.

4.2 The next stage is **filtering according to POS tagging and stem**. We use nltk POS tagging to annotate each full sentence with all substitutions and keep only words that are tagged with the same POS as the target word. This is followed by filtering for stem, using nltk's PorterStemmer. We keep only words with a different stem from that of the target word. We can look at an example to see the output of this stage. Fig.1 shows the collection of candidates chosen from the embedding space for the target word 'perils' in the sentence 'escapologists escape from handcuffs, straitjackets, cages, coffins, steel boxes, barrels, bags, burning buildings, fish-tanks and other perils, often in combination'. Fig.1.a shows the words which passed the filter. Fig.1.b shows the words removed by the filter (word size in the image is meaningless). We can see how all singular nouns were removed along with some related adjectives and verbs.

4.3 Finally we **evaluate the candidates in context** using the pre-trained BERT-base-uncased transformer model.



1.a Candidates which passed filtering

1.b Candidates removed by filtering

Figure 1: POS and stem filtering for target word 'perils'

For each entry, we pass the original sentence through BERT, along with all sentences with our target word replaced by candidates from stage 2. Each sentence is being tokenized by BERT tokenizer and we retrieve all the hidden representations of all tokens in all layers. We can now test different functions on all hidden states to obtain the best representation for a similarity measure. The highest score will be given to candidates for which this representation had the highest cosine similarity to that of the original sentence. In section 6.4 we show our comparison between three different functions: using the final hidden state for [CLS]; using the concatenation of the 4 last layers' hidden states of our target word; using the sum of the 4 last layers' hidden states of our target word. Special care needs to be taken when tracking the target/candidate hidden state. Since Bert tokenizer may split words to several tokens, we track all tokens in that case and sum (or average) their respective representations.

4.4 After choosing a representation function we sort our candidates from highest to lowest score by BERT similarity. Next, we must make sure the replacement is **simplifying the target word**. For this task, we use word-frequency statistics of 333,333 single words analyzed based on the Google Web Trillion Word Corpus (by Rachael Tatman [11]). We choose a threshold to define what is considered a simple word. All words with frequencies below this threshold will be rejected. After some consideration, we chose a frequency threshold according to the top 4% of all words. In an independent application, this threshold can be a parameter by which users can tune the level of simplification they desire.

Our list of final suggestions is now ready. The last thing we need to determine is how many suggestions we want to output to try and cover the dataset's gold labels. The answer depends somewhat arbitrarily on the number of gold labels given, as will be discussed in section 5.

5. Evaluation

In order to quantify the quality of the model suggestions, we conventionally look at three measures: precision, recall and f1-score.

Precision is the percentage of model suggestions that appear in the gold labels. Recall is the percentage of gold labels covered by the model suggestions. F1-score is the harmonic mean of precision and recall, which aims at balancing between the two.

If we favor the model to output a long list, our recall would increase, but precision would decrease. This decision is also biased by the datasets. LexMTurk includes a longer list than the rest and so it would pull the optimized retrieval number up. Fig.5 shows dependency of the different measures on the

```
Algorithm 1: emb2BERT_simplify
Input: (Sentence s, Target word w)

 1: find embedding of w in embedding space.
 2: return C closest candidates in embedding space for both
    cosine similarity and L2 similarity and merge lists.
 3: for $c \in C$ do
 4:     replace c into s to get s'.
 5:     retrieve POS(c) from POS(s').
 6:     if POS(c) != POS(w) OR stem(c) == stem(w) then
 7:         remove c.
 8:     end if
 9: end for
10: find rep(w) = representation of w in BERT(s) (4 last hidden states)
11: BERT_scores $\leftarrow \emptyset$
12: for $c \in C$ do
13:     replace c into s to get s'
14:     find rep(c) = representation of c in BERT(s') (4 last
        hidden states)
15:     BERT_scores[c] $\leftarrow$ cos_similarity(rep(w), rep(c))
16: end for
17: sort BERT_scores from high to low
18: remove all c where freq(c) $<$ freq_threshold
19: return n highest score candidates (n=7)
```

retrieved number of suggestions.

6. Experiments and results

Several experiments were performed to achieve best results:

6.1 For our first experiment, it was important to determine which pre-trained embedding space is better suited for the task. We chose to compare between GloVe [8] and Fasttext [9] applied to the NNSeval dataset. To make the comparison fair and calculations lighter, only the most frequent 400,000 words were taken from Fasttext to match the length of GloVe vocabulary. It makes sense to use recall in order to compare performance in this case. For each sentence, we return the same number of candidates from each embedding space and check how many of the gold standards for this sentence are covered by the returned list. Fig. 2 plots the average recall percentage for all sentences vs. the number of retrieved candidates n. We also count how many of the sentences are left with zero coverage as a function of increasing n. We can easily see that Fasttext is superior to GloVe. A gap of over 15% in recall is surprisingly maintained over the entire curve. We used Fasttext for all other experiments.

6.2 An additional experiment for stage 1 compared similarity measures in the embedding space. The most commonly used is cosine similarity, but L2 distance is also natural to explore. We wanted to see what L2 can add to the recall results and what are the differences in the retrieved words. We find that in general, there is good correlation between the two lists. However L2 tends to return more generic and spurious words. While these can be filtered in later stages, the filter has a cost and they can potentially slip through. However, the recall results show that L2 contribution is sub-

.3

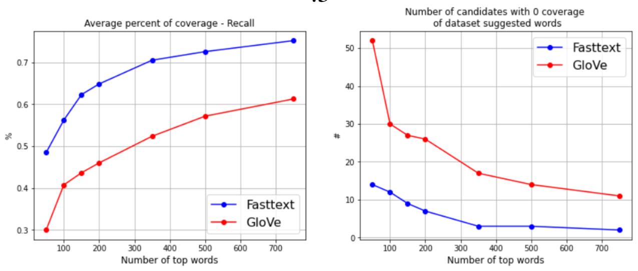


Figure 2: Comparing embedding spaces

.3

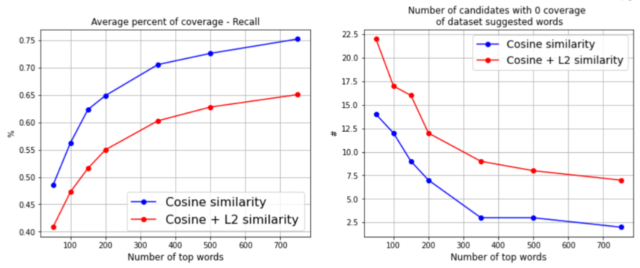


Figure 3: Comparing similarity measures

stantial. It is important to mention that the way L2 results were added was merging top n results from each similarity measure. Even though correlation is high, to be useful, the merged list must be longer than n, potentially twice as long. Nevertheless it is obvious from the results in Fig.3 that the benefit is greater than simply taking a longer list of cosine similarity alone. Merging of L2 results was therefore made default.

6.3 We kept a data structure containing candidates lists for all sentences, given the number of top words returned. All of this structure was then filtered for POS tagging and stem and we collected statistics on what percentage of words was filtered for every number of retrievals. Fig. 4 shows the increase in the filtered words percentage. Adding words is slowly becoming less effective, although our starting point was already low with about 45% of the words eliminated. These would include some spurious words, but also inflections of the original word.

6.4 Finally, we evaluate the final result after filtering by BERT. Before running candidates from very long lists, we compare similarity functions on the hidden states of BERT for 100 retrieved words, to see which method is most effective. We compare precision, recall and F1-score, using the 3 functions detailed in 4.3. Results are given in Table 2 for retrieval of 7 candidates. We conclude that determining similarity according to the target word hidden state is much stronger than similarity by [CLS] and that there is not much difference between summing or concatenating last hidden states for the target word.

6.5 We keep 20 candidates sorted according to their BERT score. This enables us to calculate precision, recall and F1-score with every cutoff of the list. We can see the results on all datasets in figure 5. We find that the best results to fit all datasets are obtained with retrieval of 7 candidates.

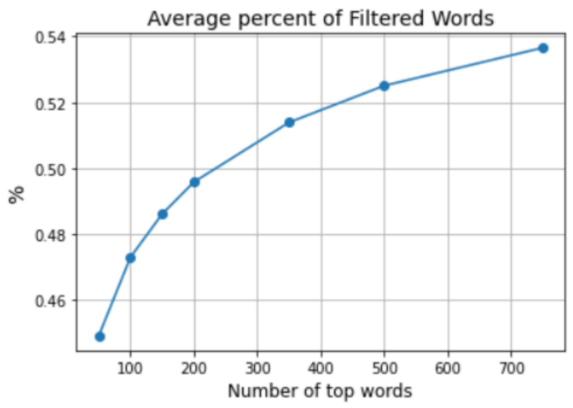


Figure 4: Filtering factor vs. retrieval from embedding

Comparison to previous work is found in Table 3, where we name our method emb2BERT. For stage 1 we use 500 words retrieved from the embedding space, although the difference is very small compared to retrieving 100 results for example. Our results are significantly better than the state of the art BERT-LS.

7. Conclusions

Our work showed that a combination of powerful available tools can be used to simplify words in context in an unsupervised way. Results show better performance than the state of art on all three traditional datasets. The same method may be employed for other languages, providing the equivalent pre-trained tools exist. When comparing our model to BERT-LS, it is possible that suggesting words from an embedding space is better suited to the task in comparison to BERT's next sentence prediction capabilities. On the other hand, our processing has a heavier filtering overhead of unfit words. Additional optimizations and modifications can be made to the model, especially regarding the word-frequency cutoff to determine the level of simplification. Replacement of multi-words can also be attempted in the future.

References. [1] Carroll, J. A., Minnen, G., Pearce, D., Canning, Y., Devlin, S., Tait, J. (1999, June). Simplifying text for language-impaired readers. In Ninth Conference of the European Chapter of the Association for Computational Linguistics (pp. 269-270).

Table 2: Comparing BERT scoring functions

Bert scoring function	Precision	Recall	F1-score
[CLS] last hidden state	0.177	0.239	0.203
Concatenate 4 last target hidden states	0.2062	0.2733	0.2350
sum 4 last target hidden states	0.2056	0.2733	0.2346

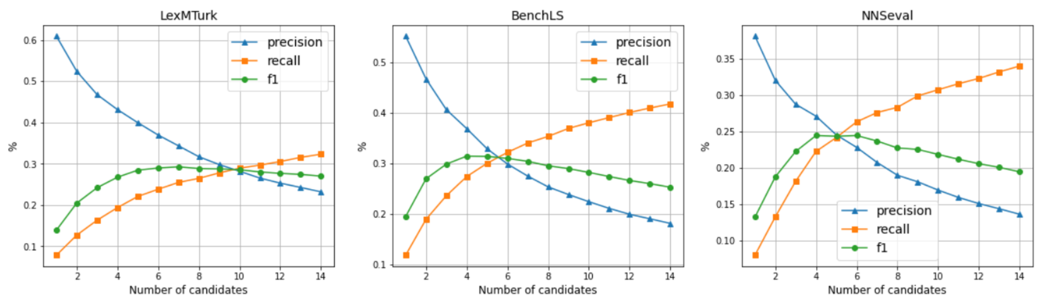


Figure 5: Evaluation dependence on final number of retrieved candidates

	LexMTurk			BenchLS			NNSeval		
	Precision	Recall	F1	Precision	Recall	F1	Precision	Recall	F1
Horn	0.153	0.134	0.143	0.235	0.131	0.168	0.134	0.088	0.106
Glavaš	0.151	0.122	0.135	0.142	0.191	0.163	0.105	0.141	0.121
Paetzold - CA	0.177	0.140	0.156	0.180	0.252	0.210	0.118	0.161	0.136
Paetzold - NE	0.310	0.142	0.195	0.270	0.209	0.236	0.186	0.136	0.157
BERT-LS	0.296	0.230	0.259	0.236	0.320	0.272	0.190	0.254	0.218
emb2BERT-7	**0.344**	**0.255**	**0.293**	**0.275**	**0.341**	**0.304**	**0.207**	**0.276**	**0.237**

Table 3. Evaluation results for LS on three datasets (different methods)

[2] Horn, Colby, Cathryn Manduca, and David Kauchak. ”Learning a lexical simplifier using wikipedia.” Proceedings of the 52nd Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers). 2014.

[3] Glavaš, Goran, and Sanja Štajner. ”Simplifying lexical simplification: Do we need simplified corpora?.” Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers). 2015.

[4] Paetzold, Gustavo, and Lucia Specia. ”Unsupervised lexical simplification for non-native speakers.” Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 30. No. 1. 2016. https://zenodo.org/record/2552381

[5] Belder, Jan De, and Marie-Francine Moens. ”A dataset for the evaluation of lexical simplification.” International Conference on Intelligent Text Processing and Computational Linguistics. Springer, Berlin, Heidelberg, 2012.

[6] Qiang, Jipeng, et al. ”Lexical simplification with pretrained encoders.” Proceedings of the AAAI Conference on Artificial Intelligence. Vol. 34. No. 05. 2020.

[7] Paetzold, Specia. (2016). BenchLS: A Reliable Dataset for Lexical Simplification [Data set]. Tenth International Conference on Language Resources and Evaluation (LREC 2016), Portorož, Slovenia. Zenodo. https://zenodo.org/record/2552393

[8] Jeffrey Pennington, Richard Socher, and Christopher Manning. 2014. GloVe: Global Vectors for Word Representation. In Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

[9] Mikolov, T., Grave, E., Bojanowski, P., Puhrsch, C., Joulin, A. (2017). Advances in pre-training distributed word representations. arXiv preprint arXiv:1712.09405.

[10] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In NAACL, 2018.

[11] Word-frequency by Rachael Tatman at: https://www.kaggle.com/datasets/rtatman/english-word-frequency