

# Unsupervised Glossary Construction

Deep Inder Mohan (IMT2017013)

George Abraham (IMT2017019)

Kaushal Mittal (IMT2017024)

# Task Description

*“To extract a set of glossary terms and their definitions from a given body of text in an unsupervised manner”*

**Primary Idea:** Split into two serialized sub-tasks:-

1. Key term Extraction
2. Definition Extraction

# Key-term Extraction

**Formal Definition:** Given text  $T$  from a collection of documents  $D$ , whose vocabulary is  $V$ , output a set  $K \subseteq N(T)$  that describes the key-terms of  $T$ .

In this definition,  $N(T)$  is the set of n-grams from  $T$  for all  $n = 1, \dots, \eta$  ( $\eta$  is a maximum term length hyperparameter) where an n-gram is defined as a sequence of  $n$  consecutive terms that show up in the initial text.

# Approaches Researched

- Most work in this topic has been done specifically keeping in mind medical or scientific textbooks.
- Supervised, semi-supervised, and unsupervised approaches explored.

# Supervised Approaches

- Paper: *“Building A Scalable Database Driven Reverse Medical Dictionary Using NLP Techniques”*
- Target: To map input list of symptoms to possible diagnosis
- Described architecture pipeline includes Word Sense Disambiguation module followed by a Semantic Similarity module

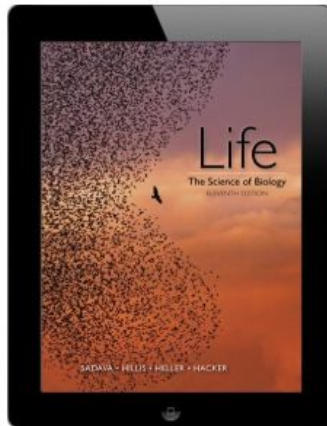
# WSD: *Lesk Algorithm*

- Algorithm works by comparing the dictionary definition of an ambiguous word with the terms contained in its neighborhood.
- Primary takeaway: words in a given "neighborhood" (section of text) will tend to share a common topic.
- **Semantic Similarity:** Glossary terms and definitions should summarise the document.

# INQUIRE

[basics](#) [features](#) [technology](#) [research](#) [learn more](#)

A prototype of an **intelligent textbook that answers students' questions**, engages their interest, and improves their understanding. [▶ Watch our introduction video](#)



What are the differences between **plant cells** and **animal cells**?

**define** Define **cellular respiration**

**structure** What is the structure of a **chloroplast**?

**function** What is the function of a **plasma membrane** in a **eukaryotic cell**?

**compare** What are the differences between **chloroplasts** and **mitochondria**?

**relate** If the **chloroplasts** were removed from a **plant**, what events would be affected?

**search** Search book for **photosynthesis**

# Automated Glossary Construction of a Biology Textbook.

-Anita Kulkarni et al. CS 229: Machine Learning, Fall 2018

- To automate the process of creating a knowledge base that includes key terms whose definitions have been selected for the textbook's glossary.
- The goal of this project is to predict which individual words, bi-grams, and tri-grams from the textbook should appear in the glossary.



# Previous Work

- Early work in this field focuses on sophisticated linguistic and statistical rules.  
[2]
- A more recent paper[3] used the Naïve Bayes algorithm along with decision trees to classify whether keyword or not.

---

[2] J. Foo and M. Merkel, “Using machine learning to perform automatic term recognition”

[3] T. Pardo, and S. Rezende, M. Conrado, A Machine Learning Approach to Automatic Term Extraction using a Rich Feature Set

# Dataset and Preprocessing

- Chapters 1-10 and 39-52 of Life
- List of the top 5000 words[4] in English.
- CountVectorizer methods in the Scikit-learn[5] Python package for machine learning.
- Removing stop words and lemmatizing.
- Pruning specific words.

---

[4] “Word frequency data.” <https://www.wordfrequency.info/free.asp>

[5] F. Pedregosa et al., “Scikit-learn: Machine Learning in Python,”

# Features

- Term frequency (TF), Document frequency (DF), TF-IDF
- Location of appearance (Sentence and Chapter/Document).
- Whether or not the term is in the top 5000 words in English.
- Variance of the term frequency per chapter across all chapters.
- Number of characters in the term.\*
- Other features such as Avg term frequency\*, occurrence in chapter title

# Model and Techniques

- Support Vector Machine (SVM)
- Gaussian Naive Bayes (GNB)
- Neural Net (Multi-Layer Perceptron)
- 10-fold cross-validation, Precision and Recall (to avoid false negatives)
- 8 fold oversampling

# Future Work

- Some changes to the learning algorithms or feature weights.
- To add crucial features such as part of speech tags and definition sentence tags

# Semi-Supervised approach

## **“Featureless Deep Learning Methods for Automated Key-Term Extraction”**

- Kush Khosla et al. 2019

- Co-training algorithm: CNN + fully connected NN
- Uses manually extracted knowledge base, i.e., pre-labelled glossary terms.

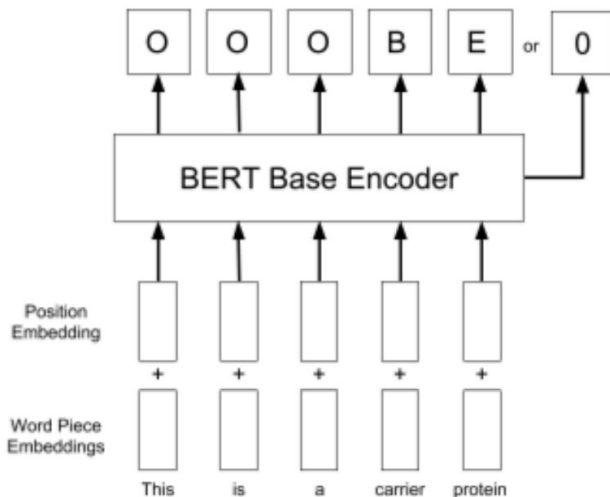
# Supervised approach

## **“Automatic Extraction of Textbook Glossaries Using Deep Learning”**

- Author et al. 2018

- BIOES tagging scheme:-
  - B - beginning
  - I - interior
  - E- end
  - S - singleton glossary
  - O - non-glossary labels
- Both term extraction and glossary sentence extraction done in an unsupervised manner.

# Glossary Term Extraction



Word representations built by concatenating GloVe word embeddings and character-level representations; input into a Bi-LSTM layer; fed into a fully connected softmax output layer; tag for each token



# Glossary Sentence Extraction

- Problem approached as a binary sequence classification task.
- One possible method: fine-tuned BERT
- Input data: Science textbooks from OpenStax

# Takeaways

- Cannot use this method as it assumes knowledge of the ontological content of the textbook whose glossary is to be built by fine-tuning BERT on relevant text.
- Fine tuning BERT requires labelled glossaries from textbooks. Our objective was to avoid the need for such data.
- BERT outperformed smaller, more specialized models on both term and definition extraction tasks, making it a prime candidate to be used in our version of the problem.

# Our Approach

The underlying task is automatically extracting textbook glossaries in an unsupervised manner. Taking inspiration from the previously discussed papers, the task was split into 2 subtasks:

1. Identifying the keywords
2. Extracting the definition for the keywords

Note: the glossary extraction is done keeping **textbooks** in mind.

# Glossary Keyword Extraction

It is done in the form of a pipeline consisting of 3 steps.

## **Step 1:**

For the unsupervised method of keyword extraction, we first use corpus statistics based models. Examples of such are: TF-IDF, RAKE, YAKE etc.

(Step 2) **Pruning based on POS tagging**

(Step 3) **Pruning Based on Named Entities**

# RAKE(Rapid Automatic Keyword Extraction)

1. Identify the candidate keywords in the document by removing 'stopwords', which are words like prepositions. The set of words between every pair of stopwords becomes one candidate keyword.
2. Create a graph of word co-occurrences for the text, and score every word in the graph using metrics of frequency and degree. A matrix is built (every row and column correspond to a word ), words are given a score. That score is the degree of the word divided by its frequency . **degree** - (the sum of the number of co-occurrences the word has with any other content word in the text) and **frequency** -(the number of times the word appears in the text)

# RAKE

	keyword	extraction	difficult	many	libraries	help	rapid	automatic
keyword	3	3	0	0	0	0	1	1
extraction	3	3	0	0	0	0	1	1
difficult	0	0	1	0	0	0	0	0
many	0	0	0	1	1	0	0	0
libraries	0	0	0	1	1	0	0	0
help	0	0	0	0	0	1	0	0
rapid	1	1	0	0	0	0	1	1
automatic	1	1	0	0	0	0	1	1

*(Example of matrix constructed for RAKE)*

There are multiple issues with this approach:-

- Mainly, it performs poorly on keywords that are unigrams.
- It is a very primitive approach that does not rely on features of words.

# YAKE ( from paper Yet Another Keyword Extractor)

Works on text statistical features extracted from the document/book to identify and rank the most important keywords. It has 4 main subtasks :

1. **Text pre-processing**
2. **Feature extraction:** A set of five features to capture the characteristics of each individual term.
3. **Individual terms score:** we heuristically combine all these features into a single measure such that each term is assigned a weight  $S(w)$ .
4. **Candidate keywords list generation:** After assigning scores wrt importance ,final step is accounting for n-gram words (uni,bi,tri).To collect the candidate keywords, we consider a sliding window of 3-gram generating a contiguous sequence of 1, 2 and 3-g candidate keywords.

# YAKE

**(1)Text Preprocessing:**apply a tokenization process which splits the text into individual terms whenever an empty space or a special character (e.g. brackets ,comma, period, etc.) delimiter is found.

**(2)Feature extraction:**Set of five features to capture the characteristics of each individual term. These are: (a) Casing (b) Word Positional (c) Word Frequency(d) Word Relatedness to Context and (e) Word DifSentence .

**(a)Casing:** Casing, reflects the casing aspect of a word.Particular attention to any word starting with a capital letter (excluding ones at the beginning of sentences) or to any acronym under the assumption that these words tend to be more relevant. Instead of counting them twice we only consider the maximum occurrence within the two of them. Equation 1 reflects this casing aspect:



$$W_{\text{Case}} = \frac{\max(\text{TF}(U(w)), \text{TF}(A(w)))}{\log_2(\text{TF}(w))} \quad (1)$$

where  $\text{TF}(U(w))$  is the number of times the candidate word  $w$  starts with an uppercase letter,  $\text{TF}(A(w))$  is the number of times the candidate word  $w$  is marked as an acronym and  $\text{TF}(w)$  is the frequency of  $w$ .

## **(b)Word Positional**

Word Positional values more those words occurring at the beginning of a document/chapter based on the assumption that relevant keywords often tend to concentrate more at the beginning of a document.

$$W_{\text{Position}} = \log_2(\log_2(2 + \text{Median}(\text{Sen}_w))) \quad (2)$$

where  $\text{Sen}_w$  indicates the positions of the set of sentences where the word  $w$  occurs, and  $\text{Median}$  is the median function. The result is an increasing function, where values tend to increase smoothly as words are positioned at the end of the document, meaning that the more often a word occurs at the beginning of a document the less its  $W_{\text{Position}}$  value. Conversely, words positioned more often at the end of the document (likely less relevant) will be given a higher  $W_{\text{Positional}}$  value. Note that a value of 2, is considered in the equation to guarantee that  $W_{\text{Positional}} > 0$ .

So keywords are extracted chapter wise from the textbook, since important words in a chapter tend towards the start of the chapter.

### (c)Word Frequency

Word Frequency indicates the frequency of the word, scoring more those words that occur more often.

This feature indicates the frequency of the word  $w$  within the document. It reflects the belief that the higher the frequency, the more important the word is. To prevent a bias towards high-frequency in long documents the TF value of a word  $w$  is divided by the mean of the frequencies (MeanTF) plus one time their standard deviation ( $\sigma$ ) as in Eq. 3. Our purpose is to score all those words that are above the mean of the terms (balanced by the degree of dispersion given by the standard deviation).

$$W_{\text{Freq}} = \frac{\text{TF}(w)}{\text{MeanTF} + 1 * \sigma} \quad (3)$$

**(d)Word Relatedness to Context( $W_{\text{rel}}$ ):** Word Relatedness to Context, computes the number of different terms that occur to the left and right side of the candidate word. The more the number of different terms that co-occur with the candidate word (on both sides), the more meaningless the candidate word is likely to be. In practical terms, the more insignificant the candidate word is, the higher the score of this feature will be. Thus, stopwords-like terms will easily obtain higher scores.

**(e) Word DifSentence:**Quantifies how often a candidate word appears within different sentences.It values more those words that often occur in different sentences.

$$W_{\text{DifSentence}} = \frac{SF(w)}{\#Sentences} \quad (5)$$

where  $SF(w)$  is the sentence frequency of the word  $(w)$ , i.e., the number of sentences where  $(w)$  appears, and  $\#Sentences$  is the total number of sentences in the text.

Word freq and word DifSentence,are combined with Word Relatedness to Context, meaning that the more they occur in different sentences the better, as long as they do not occur frequently with different words on the right or left side

### (3) Individual terms score :

we heuristically combine all these features into a single measure such that each term is assigned a weight  $S(w)$ . The smaller the value of  $S(w)$ , the more important the word ( $w$ ) would be.

$$S(w) = \frac{W_{Rel} * W_{Position}}{W_{Case} + \frac{W_{Freq}}{W_{Rel}} + \frac{W_{DifSentence}}{W_{Rel}}} \quad (6)$$

By looking at the equation, we can observe that both  $W_{Freq}$  and  $W_{DifSentence}$  are offset by  $W_{Rel}$ . The motivation behind this offset is to assign a high weight to words that appear frequently and appear in many sentences (likely indicative of their importance) as long as the word is relevant (i.e., for which  $W_{Rel}$  is low). Indeed, some words may occur plenty of times and in many sentences and yet be useless (e.g., stopwords or similar). These terms should be penalized. Likewise, the position of a word in sentences occurring at the top of a document is an important feature that is taken into account, when multiplying  $W_{Rel} * W_{Position}$ .

Now we have successfully assigned scores to each word. Now we just need to handle n-ary keywords based on these scores.

#### (4)Candidate keywords list generation

$$S(kw) = \frac{\prod_{w \in kw} S(w)}{TF(kw) * (1 + \sum_{w \in kw} S(w))} \quad (7)$$

where  $S(kw)$  is the score of a candidate keyword with a maximum size of 3 terms, determined by multiplying (in the numerator) the score of the first term of the candidate keyword by the subsequent scores of the remaining terms, such that the smaller this multiplication the more meaningful the keyword will be. This is divided by the sum of the  $S(w)$  scores to average out with respect to the length of the keyword, such that longer n-grams do not benefit just because they have a higher n. The result is further divided by  $TF(kw)$ - term frequency of the keyword - to penalize less frequent candidates.

Now that we have score of all keywords, we rank them and select the ones with the least scores.This completes step 1.

YAKE performs very well compared to other approaches and these are results on different datasets including medical research documents, computer science papers and long news documents

**Table 1.** SemEval2010, Schutz2008, 500N-KPCrowd and WICC results

Method	SemEval2010			Schutz2008			500N-KPCrowd			WICC		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
YAKE!	0.153	0.103	0.123	0.217	0.058	0.091	0.251	0.063	0.101	0.050	0.141	0.073
TextRank	0.101▼	0.067▼	0.081▼	0.198▼	0.052▼	0.082▼	0.265	0.063	0.103	0.018▼	0.058▼	0.027▼
TF.IDF	0.036▼	0.023▼	0.028▼	0.100▼	0.028▼	0.043▼	0.223▼	0.060▼	0.095▼	0.026▼	0.067▼	0.037▼
SingleRank	0.035▼	0.022▼	0.027▼	0.082▼	0.024▼	0.037▼	0.190▼	0.054▼	0.084▼	0.017▼	0.045▼	0.024▼
RAKE	0.007▼	0.004▼	0.005▼	0.013▼	0.004▼	0.006▼	0.120▼	0.038▼	0.058▼	0.004▼	0.012▼	0.006▼

# Glossary term vs Keyword

But there are few explicit properties of Glossary terms that we have not made use of yet, seen in Step2 and Step 3. Before we move forward with Keyword Definition extraction task, we are required to do a pruning of the keywords from the above YAKE algorithm. This is done in 2 steps:

- Pruning based on POS tagging
- Based on Named Entity Recognition

# Pruning

(Step 2)**Pruning based on POS tagging:**Glossary words generally are not verbs ,adverbs or prepositions.So we only extract the keywords that correspond to nouns/adjectives. More specifically keywords with POS tags:

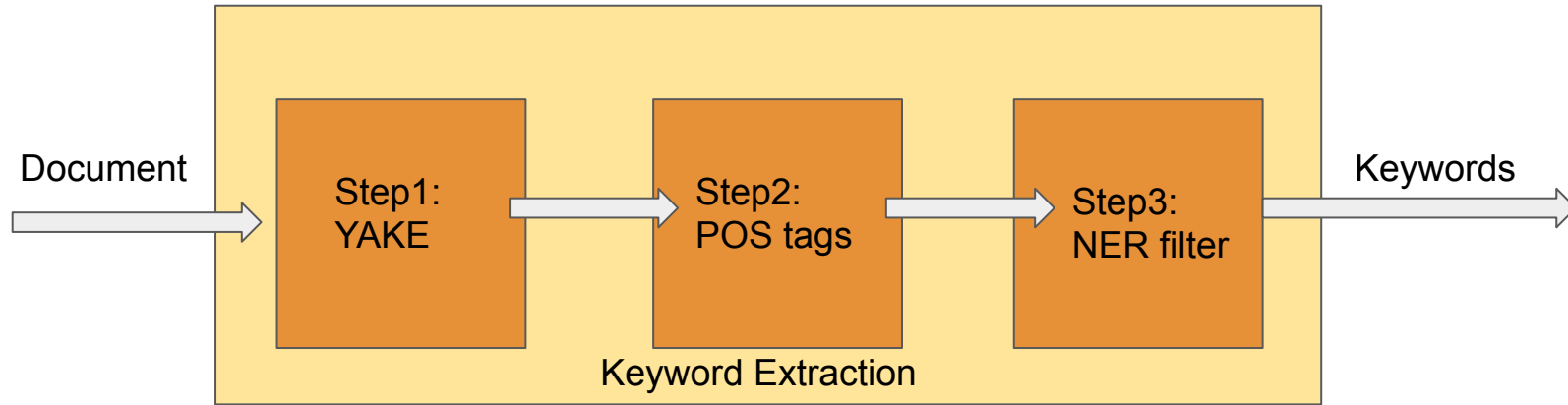
[ ‘NN’, ‘NNP’,etc.] are retained. We have used the pos tagging algorithm from **nlTK library** to achieve the same.

(Step 3)**Pruning Based on Named Entities:**We do not want define the following categories of keywords in our glossary: [People,Dates,Time,Money,Places ]

Reason being , for example in science textbooks Darwin or London may be mentioned many times but we are not interested in defining such entities.

We have used the **spaCy package** and the **Named Entity Recognition Algorithm** it provides for this task.





## **PART 2 : Glossary Definition Extraction**

Assumptions:

- Extract the definition of a keyword from the given text itself.
- Done keeping in mind the structure of a well-ordered textbook. In such an instance, definition of an introduced keyword in the given text and in vicinity of the keyword itself.

# Approach

We can expect the **definition of a keyword in the vicinity/neighbourhood** of the keyword itself. We need to note that we do not know the location of the definition across instances of the keyword. The intuition is that we need to find a sentence/sub-sentence across all vicinities

So we use BERT (Bidirectional Encoder Representations from Transformers).

Here the representation of sentence/subphrase is mean of word representations.

The approach is we take the word representation of the keyword using the BERT model represented by  $X$ . Then we run a sliding window approach over the vicinity at every instance of the keyword. We take all possible subregions in that vicinity and get the vector representation for that subregion, represented by  $R$ . Then the instance of  $R$  which is closest to  $X$  in the vector space is chosen as its definition.

# Approach

The closeness of the 2 vectors is defined by the metric cosine similarity (Similar vectors have higher cosine similarity). Finally, we choose the subregion with the highest cosine similarity (meaning the phrase and the keyword imply very similar things). This subregion is then chosen as the definition of the keyword.





















Let A be keyword and B,C be candidate definitions.

```
if cosine(A, B) > cosine(A, C), then A is more similar to B than C.
```

Points on Implementation:

- We use the base BERT architecture which has 12 transformer encoder layers stacked on top of each other and outputs 768 dimensional vectors. So we can pull out vector representation for words from any layer and still get decent outputs.

# Which layer or combination of layers provides the best representation?

For named-entity recognition task CoNLL-2003 NER			Dev F1 Score
12 	First Layer	Embedding 	91.0
...	Last Hidden Layer	12 	94.9
7 	Sum All 12 Layers	12 	95.5
6 		+	
5 		...	
4 		2 	
3 	Second-to-Last Hidden Layer	+	95.6
2 		1 	
1 	Sum Last Four Hidden	=	95.9
		11 	
		12 	
		+	
		11 	
		+	
		10 	
		+	
		9 	
		=	
			

The BERT authors tested this by feeding different vector combinations as input features for a named entity recognition task and observing the resulting F1 scores.

## Experimentation:

- The mean of the last four layers gave longer definitions, it was not so great in capturing the exact meaning.
- Outputs from the last layer, it prefers shorter definitions but more semantically similar.

# Results and observations

This approach gives decent outputs as shown in the below picture, where we see the extracted definition for some of the keywords :

```
[34] meaning_of_keyword=best_meaning_from_sentence(hopes[0])
```

```
[35] print(meaning_of_keyword)
      print (hopes[0][1])
```

```
↳ ['platform']
   ['Google']
```

```
▶ meaning_of_keyword=best_meaning_from_sentence(hopes[17])
```

```
[38] print(meaning_of_keyword)
      print(hopes[17][1])
```

```
↳ ['deny']
   ['declined']
```

**Note:** For composed words, words that are broken down into smaller more meaningful words during tokenization it performs OK but for words out of vocabulary performs badly.Ex: `[['deny'], ['declined']]`  
`[['happening', 'Google', 'itself'], ['Kaggle']]`

While tokenizing Kaggle , it got split into `[Kag,##gle]` none of which are meaningful subwords and therefore similarity measures fail here.

# Time Complexity:

Initially is  $O(k.n.c^2 .\lambda)$  where  $k$  is size of keyword table ,  $n$  is size of book/doc ,  $c$  is size of vicinity and  $\lambda$  is time complexity for getting BERT representation for a subsection.

## Improvements/Further Approaches:

- Can be improved by having a hash table for the keyword so that in constant time we can check if given word is a keyword.  
 $O(H(k).n.c^2 .\lambda)$  where  $H(k)$  is complexity to look up a keyword in the hashtable.
- Better and additional pruning methods like words not so common (not in top frequent words) in corpus be given higher scores.
- Instead of mean, try to account for no. of words that are closer to the original keyword.

# References

- ❑ **Building A Scalable Database Driven Reverse Medical Dictionary Using NLP Techniques**, Pragma Tripathi, Manuja Deshmukh
- ❑ **Life: The Science of Biology**, D. Sadava, D. Hillis, H. C. Heller, and S. Hacker
- ❑ **Automated Glossary Construction of a Biology Textbook**, Anita Kulkarni, Rachel Smith
- ❑ **Featureless Deep Learning Methods for Automated Key-Term Extraction**, Kush Khosla et al.
- ❑ **Automatic Extraction of Textbook Glossaries Using Deep Learning**, Manish Singh, Matthew Boggess
- ❑ **Automatic keyword extraction from individual documents**, Stuart Rose, Dave Engel, Nick Cramer and Wendy Cowley
- ❑ **A Text Feature Based Automatic Keyword Extraction Method for Single Documents**, Ricardo Campos, Vítor Mangaravite, Arian Pasquali
- ❑ **BERT- Pre-training of Deep Bidirectional Transformers for Language Understanding**, Jacob Devlin, Ming-Wei Chang, Kenton Lee, Kristina Toutanova