Featureless Deep Learning Methods for Automated Key-Term Extraction

Kush Khosla
Department of Mathematics
Stanford University
Stanford, CA 94305
kkhosla@stanford.edu

Robbie Jones
Department of Computer Science
Stanford University
Stanford, CA 94305
rmjones@stanford.edu

Nicholas Bowman
Department of Computer Science
Stanford University
Stanford, CA 94305
nbowman@stanford.edu

Abstract

Domain-specific term extraction is an important task in a number of areas, and particularly in the realm of knowledge base construction from unstructured text data. The motivation behind this project is to be able to build an "intelligent" textbook, where all significant words are hot buttons that take the student first to a definition and then gives them opportunities to explore connected terms and concepts. This sort of knowledge base construction is currently done by hand, and is therefore a costly, time-intensive process. Our goal in this paper is to present a method for automating the first step in the knowledge base construction process, by building a classifier to identify from raw textbook text those words that are important enough to be placed in a post-chapter glossary. We aim to achieve high precision in glossary term extraction, as generation of incorrect glossary terms is thought to be disruptive to student learning. To tackle this problem, we try multiple different architectures of featureless deep learning approaches, including both supervised and semi-supervised models, and present their results. We utilize both a convolutional neural network and a standard fully connected neural network, as well as a semi-supervised co-training pipeline that unifies both of these models. While both supervised approaches substantially outperform a standard multi-layer perceptron baseline with hand-engineered features, we find the co-training approach to be unsuccessful in achieving competitive performance. Evaluations on three biology textbooks demonstrate that the supervised CNN with contextual word vectors and dimensionality reduction performs the best on the data-constrained task of term extraction.

1 Introduction

Term extraction is a broad task within NLP that exists as a subtask of information extraction, which is the task of automatically extracting structured information from unstructured documents. Term extraction has a variety of applications to common NLP problems, including document summarization, machine translation, document identification and semantic similarity evaluation. For example, term extraction is useful for document summarization, because a concise summary should make sure to include all of the important terms of the original article. In another example, knowing the key-terms present within a document allows for easier document identification and efficient

document search and information retrieval. In our case, we are interested in key-term extraction as applied to the task of automatic glossary creation, inspired by the *Inquire* intelligent textbook [3]. *Inquire* is an online biology textbook that aims to supplement student learning by providing features such as in-line term definitions, question generation, question answering and more. In order to complete these tasks, the textbook must have a knowledge base of the information contained in the book. The first step to the construction of this knowledge base is gathering important technical terms and their definitions in order to relate them to each other. Once these terms are filtered, their low ambiguity and high specificity make them particularly useful for supporting the creation of the domain ontology that underlies all of the interesting features of an intelligent textbook. Right now, the task of term extraction is a slow, labor-intensive process done entirely by hand by domain experts. This work aims to automate the process of key-term extraction from textbooks in order to reduce the amount of human labor needed for ontology building and provide an extensible, generalizable framework that goes beyond the specific field of biology.

Formally, the task of key-term extraction can be described as follows: 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 $n = 1, ..., \eta$ (η 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. An important note is that the phrase 'key-term' is a general way of stating 'a term of importance.' The actual definition of key-term will vary from application to application. In our case, a key-term is considered to be one that appears in the glossary of the textbook, which serves as a brief dictionary of significant domain-specific terms.

In order to accomplish this task, we use labelled data to learn a function $f : N(T) \rightarrow \{0, 1\}$ with the goal that $f(x) = 1$ when x should be in the glossary, and $f(x) = 0$ otherwise. In order to learn this function, the labelled text T should have examples of both key and non key-terms so that a model can learn what each looks like. Let $G \subseteq N(T)$ denote the 'gold terms' for the text T: the glossary terms. Here lies a roadblock when attempting to solve this problem. In the case of glossary term extraction, there are many more non key-terms than key-terms ($|G| \ll |N(T)|$), which can be seen in Table 1. This class imbalance makes it difficult for a neural network to learn what a 'positive' example looks like because the prior is so heavily weighted in the negative direction. We are faced with the challenging task of having a relatively minuscule set of labelled positive training examples, from which the model must learn to generalize.

Book ID	$\lvert N(T)\rvert$	$\lvert G\rvert$	% Positive
1	1,569,238	2307	0.15
2	1,612,753	1,744	0.10
3	3,089,173	1,902	0.06

Table 1: Dataset Class Distribution for $\eta = 4$

In order to accomplish this task without relying on hand-engineered feature sets, we choose to represent our candidate terms using word vectors, which act as the inputs into our deep neural network classifier. We explore the use of both traditional skip-gram generated word vectors (FastText)[4] and contextual word embeddings (BERT) [1] for this task and find that contextual word embeddings provide for better performance in a data-constrained space.

2 Related Work

Traditional approaches to this problem often rely on the training of binary classifiers to identify whether a candidate term is relevant or not. In such approaches, the models operate on hand-selected features of the input text, including statistical and linguistic information, like document frequency and part-of-speech patterns [6]. This exact problem setting was previously attempted by a group of Stanford students in the machine learning class for their final project [5]. For each of their candidate terms (which were unigrams, bigrams, and trigrams extracted from the original text), they constructed a feature set that includes such things as TF-IDF, location in the sentence of first appearance, and number of occurrences in chapter titles. We use their results as a baseline and motivation, but aim to move beyond the constraints of hand-engineered features. Specifically, we choose to avoid these approaches as finding an optimal feature set is a time-consumptive process that also creates the need for labor-intensive input from domain experts. In addition, we try to develop methods that work well

in data-constrained environments, as labelled training data is often unavailable or difficult to obtain in this domain, due to the inherently difficult nature of the process of key-term extraction for glossary construction.

In this paper we build off of a co-training approach similar to that presented by Wang et al. [9]. A noted benefit of using a co-training approach is that it is a semi-supervised method that relies only on the existence of a small seed set of terms, which allows it to be much less dependent on large amounts of labelled training data. Co-training involves training two networks at once on the data, each with a different 'view' of the input. This is done with the hope that one network can learn valuable information that the other might overlook, and then share this insight through labeling examples and handing them off to the other model. However, results show that this co-training approach is not needed in this glossary term extraction, and can actually be detrimental to learning.

3 Approach

In this section we will describe our approach to creating models that can automatically determine if a term is a glossary term. First, we describe the baseline performance achieved on this task by previous student groups. Second, we explain the neural architectures that were used. Finally, we discuss the training algorithms implemented to train the different neural architectures explained in the previous section.

3.1 Baseline

As described in the Related Work section, we will be referring to the work done by last quarter's CS229 project group as our baseline model [5]. Specifically we are comparing with their best model, which was feature-based multi-layer perceptron (MLP). In order to ensure consistency in reported results and to make sure we are making a fair comparison, we took their provided code base and made slight modifications to their input processing to make it compatible with the format of data we were using for our own model. Thus, for all numbers presented later in the Experiments section, we have run the group's MLP training and evaluation process while mirroring the same training/evaluation data splits and feeding in the exact same textbook data into both our pipeline and theirs. For these reasons, we believe this provides a fair baseline.

3.2 Word Vectors

In our problem, we are working with domain-specific literature, which can pose a problem for pre-trained word vectors because there is a high likelihood of out of vocabulary (OOV) words. We took two approaches in generating domain-specific word vectors to use for our task. The first approach stems from the belief that a model that takes into account subword information would perform well by extracting prefixes and suffixes that are common in biological literature. For these reasons, we opted for character-level n-gram embeddings from Facebook's FastText library[4] which were originally trained using a skip-gram model on a Wikipedia corpus. Whenever we generate new candidate sets and seed sets for our experiments, we first run each unigram through the FastText model in order to obtain their word embeddings and then save them to disk. Our second word vector approach was motivated by thinking about how we could incorporate contextual information about candidate terms into our classification process. For this reason, we also chose to generate contextual word embeddings using BERT [1], which involved running the `bert-as-service` tool [10] on our whole corpus of textbook data and generating unique contextual vectors for each unigram and saving them to disk for later use in experiments.

3.3 Convolutional Neural Network

In order to learn which terms should be included in the glossary, we use a convolutional neural network as a classifier. A candidate $c \in C$ is converted to a 768×4 representation, where each row is a word vector for a word in the term. This initial input is first fed through a linear transformation that acts to reduce dimensionality, reducing the size to 300×4. This reduced matrix is then fed through three different one-dimensional convolution layers. These three layers correspond to looking at bigrams, trigrams and quadgrams within the term. These outputs are then concatenated together. After passing through a max-pooling and ReLU layer, the network uses a two dimensional convolution

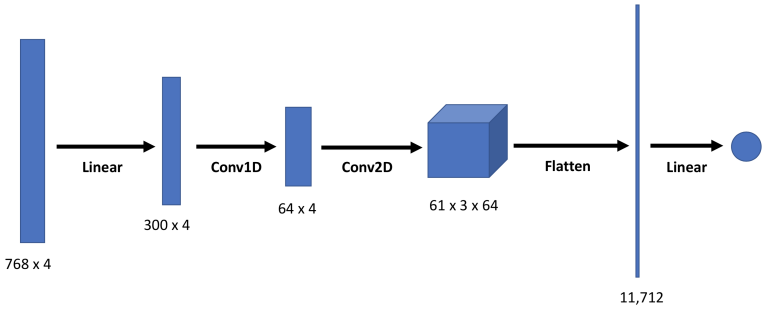

Figure 1: Design for the convolutional neural network. Note: max-pool layers are omitted from the diagram for simplicity.

layer with 64, 2×2 filters. After one more max-pool layer, the input is flattened and passed through a dense layer. From the previous layers, the input is transformed to a $61 \times 3 \times 64$ tensor, so the fully connected layer has 11712 nodes, and outputs a single number that represents the probability the input is a glossary term.

3.4 Fully Connected Neural Network

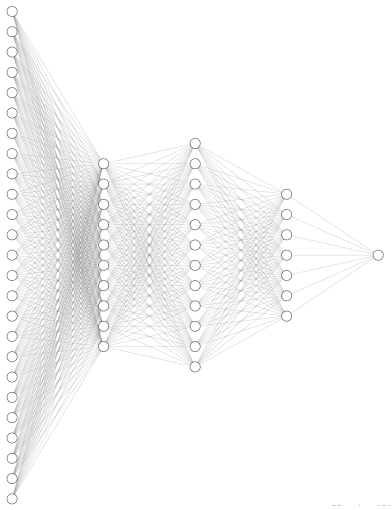

Figure 2: Fully connected neural network. This diagram in every layer (except the last) has about 1/120th of the actual number of neurons.

In addition to our convolutional neural network, we use a vanilla fully connected neural network. Just like in the convolutional neural network, the first step is to reduce the dimensionality. The input has $4(768) = 3072$ neurons, while the second has just $4(300) = 1200$. The third layer has $1200(5/4) = 1500$ neurons, and the fourth layer has $1200(2/3) = 800$. Each neuron has a dropout probability of $.5$, and is followed by a ReLU. Once again, in the last neuron we pass the signal through a sigmoid function in order to get the probability that the input is a glossary term.

3.5 Co-Training Algorithm

The co-training algorithm that we implemented matches the algorithm described by Wang et al. in [9]. Figure 1 in their paper presents the overall architecture of the co-training network and Algorithm 1 in the same paper provides the pseudo-code that we implemented to get our co-training pipeline running. However, in our implementation of the algorithm, each model has its own set of training

data. The most confident predictions from one model are then added to the labelled data for the other model. This allows the models to share examples about which they are very certain.

Although the authors chose to use a CNN and an LSTM as their two models to get different "views" of the data, we found an LSTM approach to be ineffective in our problem domain. This is due to the fact that most glossary terms are 1 to 2 words, with none being longer than 5 words, neutralizing the effectiveness of the memory and sequence tracking provided by an LSTM. Instead, we choose to use two previously described models (CNN and fully connected) as the component models of the co-training pipeline.

4 Experiments

4.1 Datasets and Data Extraction

We evaluate our model on three textbooks. The first two textbooks, on which we did most of our development work, are open-source biology textbooks curated by the Openstax education initiative that are titled *Biology* and *Microbiology*[8][7]. The *Biology* textbook contains 392,311 individual tokens and 2,307 ground truth glossary terms, while the *Microbiology* textbook contains 403,189 individual tokens and 1,744 ground truth glossary terms. The final textbook, which we held out evaluation on until the very end of our project to avoid researcher overfitting is the *Life* textbook written by Sadava et al. We chose to hold out this textbook until the end because it is the actual textbook that underlies the previously described *Inquire* intelligent biology textbook, and glossary extraction for this textbook was the ultimate task we set out to accomplish. The *Life* textbook contains 772,294 individual tokens and 1,902 ground truth glossary terms. Though all three textbooks we chose to use for evaluation are in the domain of biology, our approach makes no assumptions about ontological content and thus is extensible to textbooks in any other scientific domain.

In order to extract sentences and gold terms from all three of these textbooks, which were all originally made available to us in PDF format, we were force to manually copy the text out of each PDF on a page-by-page basis. This was due to the fact that automated text extraction tools performed poorly and led to dirty data, with many invalid characters and improperly concatenated terms. Thus, curating the text corpora that were eventually fed into our models proved to be a very time-consuming process that consumed many person-hours. However, going forward we plan to make all of our extracted data available in text format to make future work on this topic more accessible.

4.2 Preprocessing

We firstly clean the textbook data by removing any existing glossary section from the end of each chapter. For the remainder of our preprocessing pipeline, we rely on the use of *spaCy*, a free open-source library for Natural Language Processing in Python which features tools for POS tagging and word lemmatization[2]. After extracting the chapter text information from each of the textbooks, we use the lemmatization capabilities of *spaCy* to transform each token from the original text into its associated lemma. In addition, we do this for the glossary terms as well so that there is continuity during the evaluation process. This standardizes the appearance of each word, converting all tokens to lower-case, removing plurals, and unifying verb conjugations as well.

We also utilized *spaCy* as a part of speech (POS) tagger on the textbook sentence data. As part of of the task, we generate a set of candidates $C \subseteq N(T)$, using regular expressions on the POS tags generated by *spaCy*. Enough regular expressions were used with the goal of generating a C with $G \subseteq C$. That way, instead of having the neural network read the entire textbook, it must just identify gold-terms from C. Some examples of regular expressions we built to extract candidates include:

<ADJ><NOUN>+*
<ADJ><PROPN>+<PART>?<PROPN>**
<VERB><NOUN>+

Some examples of extracted candidates are *distinct gamete, identical teloblast cell, Van der Waal interaction, chronic subcutaneous infection* and *secondary radial symmetry*.

For *Biology*, this candidate extraction process retrieved about 60,000 candidates, and contained about 93% of the 2,300 gold terms. Thus, even with candidate extraction, positive examples constitute only about 4% of all data. Following the candidate generation and lemmatization steps, we implemented an extensive validation pipeline that ensured that the lemmatization of terms was consistent between candidates and gold terms, ensuring that we did not run into any out-of-vocabulary issues during training and testing.

4.3 Experiment Settings

Both the CNN and fully connected network are trained using Stochastic Gradient Descent with momentum. We use a learning rate $\alpha = .01$, momentum $\beta = .9$ and a batch size of 32. These values were found to give quick convergence. However, in future work more hyperparameters must be tested.

For co-training, the seed set uses 30 positive examples and 200 negative examples. In each iteration, 500 terms are evaluated, and the top 5 most confident predictions are added to the labelled data.

We trained our models on a Microsoft Azure N6 VM with 6 vCPUs, 56 GB of memory and an 8 GB Nvidia Tesla M60 GPU. For our supervised learning experiments, the CNN took anywhere between 10 and 20 minutes for 150 epochs, while the fully connected net took between 30 and 40 minutes for 200 epochs.

4.4 Evaluation Methods

In order to address the difficulty of evaluating the output for a subjective task like glossary-building, we have chosen to analyze our results using both automatic metrics and more qualitative human-centered metrics.

4.4.1 Automatic Evaluation

The three automatic metrics that we use to evaluate our progress in solving the stated problem are the standard metrics of precision, recall, and F1 score. We have specifically chosen not to use a blanket accuracy metric due to the heavy prevalence of negative tokens in our datasets. For this reason, we avoid the accuracy metric in evaluating our models and focus on precision, recall, and F1. In the case of evaluating our co-training results, since we begin with a seed set of gold labelled terms, we must not consider these terms when calculating our metrics. Given a final labelled set L, a gold set G, and our seed set S, let X_+ denote the values of X which have a positive label associated with them in X. Then, the definitions of our metrics are as follows:

$$\text{precision} = \frac{|(L_+ - S_+) \cap (G_+ - S_+)|}{|L_+ - S_+|}$$

$$\text{recall} = \frac{|(L_+ - S_+) \cap (G_+ - S_+)|}{|G_+ - S_+|}$$

$$\text{F1} = 2 \cdot \frac{\text{precision} \cdot \text{recall}}{\text{precision} + \text{recall}}$$

These metrics also apply for evaluation of the fully supervised models, where we consider $S_+ = \varnothing$.

4.4.2 Human Evaluation

The motivation for using human evaluation stems from the fact that glossary building is often a subjective task, and textbook writers often significantly diverge in the types of terms that they choose to include in their glossaries and in their definitions of what constitutes an "important" term. For this reason, we introduce two human evaluation metrics that reflect on the performance of our model. The first evaluation metric involves asking an expert human evaluator to inspect the false positive terms flagged by our model and then make a determination of whether or not these terms should actually belong in a glossary. The purpose of this evaluation metric is to try to shed some light on the ambiguity that exists in the task of glossary creation and establish an updated set of precision, recall, and F1 score values based off of the input of the expert human evaluator. Specifically, we take any original false positive terms classified by the human expert as actually a good glossary term and

relabel them as true positives, and then recalculate all of our automatic terms given these new labels. The second evaluation metric focuses on getting a human domain expert to comparatively rank the quality of three groups of glossary terms. To conduct this evaluation, we walk the domain expert through three rounds, where they are presented 5 terms from each model every round and asked to rank the three models from highest quality to lowest quality. At the end of 10 rounds, we assign a score to each model by assigning points for getting ranked first, second, and third in a given round (5, 3, and 1 points respectively) and summing across all rounds. These final scores allow us to quantify the glossary term quality from a human perspective.

4.5 Results

The automatic metrics (precision, recall, and F1) as well as our performance under human evaluation are detailed in the tables below. CNN and FC are fully-supervised experiments using the CNN and fully connected net, respectively. Co-Train uses both the CNN and FC networks in the co-training algorithm. CNN-B is a CNN that was trained on *Biology*, but makes predictions on *Life*. CNN-B SSL uses this same network in a semi-supervised learning paradigm. Essentially, it is the co-training algorithm using a single model. The model makes predictions, and adds its most confident predictions to its data.

Table 2: Automated metrics (left), and metrics after human input (right).

Biology (OpenStax)			
Model	**Precision**	**Recall**	**F1**
CNN	0.62	0.32	0.43
FC	0.35	0.22	0.27
Co-Train	0.13	0.44	0.20
Baseline	0.25	0.29	0.26
Microbiology (OpenStax)			
Model	**Precision**	**Recall**	**F1**
CNN	0.64	0.23	0.34
FC	0.0	0.0	N/A
Co-Train	0.17	0.41	0.24
Baseline	0.21	0.39	0.27
Biology + *Microbiology* (OpenStax)			
Model	**Precision**	**Recall**	**F1**
CNN	0.61	0.18	0.28
FC	0.0	0.0	N/A
Co-Train	0.15	0.40	0.22
Baseline	0.16	0.19	0.17
Life: The Science of Biology (Sadava)			
Model	**Precision**	**Recall**	**F1**
CNN-B	0.27	0.20	0.23
CNN-B SSL	0.1	0.6	0.17
Co-Train	0.08	0.5	0.14
Baseline	0.15	0.46	0.23

False Positive Evaluation (models trained on *Biology*)			
Model	**New Precision**	**New Recall**	**New F1**
CNN	0.8	0.42	0.55
FC	0.69	0.43	0.53
Power Ranking			
Textbook	**Baseline**	**Our Model**	**Glossary**
Biology	24	32	34
Microbiology	16	36	38

5 Analysis

Across the board, the CNN performs the best in terms of precision, as well as F1 score. We hypothesize this is because it is able to leverage the structure embedded within the word vectors more efficiently than the fully connected network. In fact, in most cases, the fully connected network never learned what a positive example looks like. In both *Microbiolgy* and *Microbiology + Biology*, the fully connected network never predicts a term as a glossary term. This is most likely due to the small percentage of positive examples, while *Biology* had a ratio that was sufficient for learning.
Co-training, however, is on the opposite end of the spectrum. Because the seed set had a higher positive:negative ratio than the true distribution (for purposes of adequately learning what a positive example looks like from such small amounts of data), it predicted positive much more often. This is clear because precision goes down from the CNN, but the recall increases a great deal. It is worth noting that through the co-training approach, the fully connected network begins to predict terms as glossary terms. This can be either from the positive terms given to it by the CNN, or the

positive:negative ratio in the seed set. More experiments must be conducted to find out what causes the fully connected net to start predicting glossary terms.
As previously mentioned, most experiments conducted on *Life* use networks trained elsewhere. This is to simulate our real use case: using a pre-trained model on a new textbook to automatically extract the glossary. The decrease in precision from the CNN evaluated on *Biology* to the CNN evaluated on *Life* makes sense because it was trained using different word vectors. However, it is promising to see a precision of 27%, since this implies that information learned from one textbook still applies to another.
From a human evaluation perspective, we see promising results in our comparison between our best model, the baseline, and the actual true glossary terms. The reason we incorporated human evaluation into this task was because we noticed that our models were producing terms that were being tagged as false positives but to the human eye looked like valid glossary terms. This included terms like *catabolite activator*, *epiglottis*, and *amyloid plaque*. Indeed, when we asked experts in the domain of biology to investigate our false positives and label the ones which they thought belonged in the glossary, we obtained about a 45% re-labelling rate. This goes to show that our model may actually be doing a better job of term prediction than the automatically evaluated statistics of precision and recall can capture. In addition, when evaluating the quality of our model on a power ranking scale in comparison to the baseline and the true glossary terms, we see in the above results table that our model is nearly indistinguishable from the true glossary terms model in terms of the quality of terms that it produces. However, one weakness of our model is that it often hones in on terms that are too specific to be included in the glossary (including terms like species names and certain DNA sequences), which is feedback that we received from our expert human evaluators.

6 Conclusion

We have implemented a system for taking large text corpora and extracting their key terms. Although our domain for this research was biology textbooks and their glossaries, our pipeline can be adapted for any application where labelled examples of key terms are present.
Our primary bottleneck throughout this work was availability of, and accessibility, to data. Not only are there very few positive examples of glossary terms for each textbook, making model training difficult, but simply extracting all of the necessary text data was a very time-consuming process. Much of our time was spent hand-copying textbook data from PDF files into text files and validating our preprocessing pipeline on top of this data. Access to structured, parseable textbook data would be of immense benefit to future iterations of this research.
Because of time constraints, we were also unable to perform as much hyperparameter tuning as we would have liked. Some of the difficulties we had training the fully connected network in a supervised manner could have possibly been mitigated with more informed hyperparameter choices. On the other hand, it is encouraging to see the results we did achieve, especially in the human evaluation setting, given the limited time we had to tune these models.
There are many potential avenues to pursue regarding next steps: We have seen how co-training can help a neural network learn a distribution it otherwise might have never learned in a strictly supervised setting. Given that NLP problems often deal with label-constrained environments, it would be interesting to continue investigating semi-supervised approaches such as co-training. Regarding model architectures, there is reason to believe that a character-level recurrent network would have the potential to effectively learn a distribution for key-terms, especially within scientific literature given the interesting morphological structures present in scientific terms. Finally, we have not yet seen any examples of using BERT to extract key-terms. While we did use BERT to generate contextual word vectors, there is certainly a chance that applying BERT as an end-to-end classification tool would prove very effective.

7 Acknowledgements

The work accomplished on this project would not have been possible without the generous support of John Hewitt, who provided suggestions on a number of different subjects, including evaluation strategies, model improvements, and different loss metrics.

8 Additional Information

- **Mentor**: Our mentor for this project is Dr. Vinay K. Chaudhri, who is a visiting professor in the theory group of the CS department. Our CS224N staff member is Pratyaksh Sharma.
- **External Collaborators**: We have been consulting with John Hewitt, who is a current PhD student in the Stanford NLP department for advice on general NLP model building strategies and best practices for working on an extensive deep learning project. He has not contributed any code to the project nor written any part of any of the submitted project documentation, including this report.
- **Sharing Project**: We are not sharing this project with any other classes.

References

[1] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". In: *arXiv e-prints*, arXiv:1810.04805 (Oct. 2018), arXiv:1810.04805. arXiv: 1810.04805 [cs.CL].

[2] Matthew Honnibal. *spaCy: Industrial-Strength Natural Language Processing*. Accessed: 2019-03-16. 2017. URL: https://spacy.io/.

[3] SRI International. *Inquire: An Intelligent Textbook*. http://inquireproject.com/. Accessed: 2019-03-16.

[4] Armand Joulin et al. "Bag of Tricks for Efficient Text Classification". In: *arXiv preprint arXiv:1607.01759* (2016).

[5] Anita Kulkarni and Rachel Smith. "Automated Glossary Construction of a Biology Textbook". In: Stanford, California, Nov. 2018. URL: http://web.stanford.edu/~vinayc/intelligent-life/Fall-2018-CS229-Project.pdf.

[6] Thiago Alexandre Salgueiro Pardo Merley da Silva Conrado and Solange Oliveira Rezende. "A machine learning approach to automatic term extraction using a rich feature set." In: *HLT-NAACL*. 2013, pp. 16–23.

[7] Nina Parker et al. *Microbiology*. https://cnx.org/contents/e42bd376-624b-4c0f-972f-e0c57998e765. Accessed: 2019-03-16. Nov. 2016.

[8] Connie Rye et al. *Biology*. https://cnx.org/contents/185cbf87-c72e-48f5-b51e-f14f21b5eabd. Accessed: 2019-03-16. Oct. 2016.

[9] Rui Wang, Wei Liu, and Chris McDonald. "Featureless Domain-Specific Term Extraction with Minimal Labelled Data". In: *Proceedings of the Australasian Language Technology Association Workshop 2016*. Melbourne, Australia, Dec. 2016, pp. 103–112. URL: http://www.aclweb.org/anthology/U16-1011.

[10] Han Xiao. *bert-as-service*. Accessed: 2019-03-12. 2019. URL: https://github.com/hanxiao/bert-as-service.