To the Graduate Council:

I am submitting herewith a dissertation written by Tabitha K Samuel entitled "Using Human Interaction with Natural Language Processing Techniques to Reinforce Vocabulary Comprehension and Usage." I have examined the final paper copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

_____

Dr. Michael W. Berry, Major Professor

We have read this dissertation
and recommend its acceptance:

_____

Committee Member 1

_____

Committee Member 2

_____

Committee Member 3

Accepted for the Council:

_____

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

To the Graduate Council:

I am submitting herewith a dissertation written by Tabitha K Samuel entitled "Using Human Interaction with Natural Language Processing Techniques to Reinforce Vocabulary Comprehension and Usage." I have examined the final electronic copy of this dissertation for form and content and recommend that it be accepted in partial fulfillment of the requirements for the degree of Doctor of Philosophy, with a major in Computer Science.

Dr. Michael W. Berry, Major Professor

We have read this dissertation
and recommend its acceptance:

Committee Member 1

Committee Member 2

Committee Member 3

Accepted for the Council:

Dixie L. Thompson

Vice Provost and Dean of the Graduate School

(Original signatures are on file with official student records.)

Using Human Interaction with Natural Language Processing Techniques to Reinforce Vocabulary Comprehension and Usage

A Dissertation Presented for the

Doctor of Philosophy

Degree

The University of Tennessee, Knoxville

Tabitha K Samuel

May 2025

To my grandmother, Sita

Acknowledgments

I want to thank my advisor, Dr. Michael W. Berry, for, first and foremost, encouraging me to embark on this compelling journey and for being a kind and supportive mentor throughout. I would also like to thank Dr. Jillian McCarthy for helping me learn about the science of language. A special thanks to Dr. Amir Sadovnik and Dr. Audris Mockus for helping me scope and correctly frame my research undertaking. Thank you to Readworks for allowing me to make liberal use of their educational K-12 material, which formed the experiment material for this dissertation.

I am grateful to my colleagues at the National Institute for Computational Sciences for their patience as I balanced a full-time job with completing this degree. I will always be humbled by the support of my colleagues and mentors in the Research Computing community, in particular, Rachana Ananthakrishan, Tim Boerner, Dana Brunson, Shafaq Chaudhry, Dhruva Chakravorty, Kathryn Kelley, Rich Knepper, Ruth Marinshaw, Alaine Martaus, Greg Peterson, Winona Snapp-Childs, Anthony Skjellum, Shava Smallen, and Craig Stewart. Your continuous encouragement and reminders to 'just get it done' were instrumental in this process.

To my aunt, Shobhana Chelliah, a computational linguist specializing in endangered languages, thank you for the many enlightening conversations about language and syntax. Thanks to my Manohar and Mallika Samuel, my parents, who always encouraged me to buck traditions and follow my dreams. Finally, none of this would have been possible without my husband, Jim Berrier - thank you for keeping me fed, loved, and sane.

"Reading is the key that opens doors to many good things in life.
Reading shaped my dreams, and more reading helped me
make my dreams come true."
- Ruth Bader Ginsberg

Abstract

This dissertation proposes SENCE: SENtence Curation and Evaluation - a Natural Language Processing (NLP) aid to be used in an educational setting for children. SENCE is designed as an AI-augmented tool for educators, such as general and special education teachers and practicing school-based speech-language pathologists. While several commercially available products incorporate NLP techniques for teaching adults language skills, the field is still nascent for incorporating NLP into teaching aids for children. SENCE uses NLP techniques to reinforce vocabulary comprehension and usage in children. Additionally, it integrates human interaction with NLP techniques, allowing domain specialists to improve results before they are presented to students. SENCE uses off-the-shelf NLP libraries such as spaCy and Stanza in combination with NLP techniques such as lemmatization, part-of-speech tagging, and vocabulary similarity. These methods are integrated to identify key vocabulary words and sentences using those keywords. An evaluation is created based on these keywords and sentences. SENCE thereby creates an automated process to gauge students' vocabulary comprehension over time. The evaluations can be shared between classes and instructors. Further, students can be quickly assessed for retention of words taught earlier in the school year. Through these methods, SENCE provides a novel and easy-to-use NLP-powered application for non-computer scientists to use NLP for everyday classroom tasks.

Table of Contents

Nomenclature

MongoDB	Database collection and column names are italicized
`Code`	Python file names and code are in teletype font

Chapter 1

Introduction

This dissertation proposes **SENCE: SENtence Curation and Evaluation** - a natural language processing aid to be used in an educational setting for children, including those with a developmental language disorder or who have a hearing impairment [20], or who are hearing impaired. SENCE is meant to be used in a vocabulary-teaching setting, where children are taught expanded vocabulary through texts and other media. SENCE is meant to be used at the end of a lesson to reinforce the meaning of words taught during the lesson. It assesses if the student can correctly use the newly introduced word in familiar (text from the lesson being currently discussed) and unfamiliar (use the word correctly from a text unfamiliar to the student) settings.

SENCE also has the capacity to assess the student's grasp of vocabulary meaning through progressively simple to complex sentences. At the end of the lesson, the instructor can get an individualized report of each student's progress in mastering the vocabulary taught that day. SENCE can also be used to assess and reinforce words taught in previous lessons, so that students have multiple pathways to retaining knowledge of vocabulary as they progress through different words over time.

SENCE is meant to be a Natural Language Processing (NLP)-augmented educational assistive tool and is not intended to replace educators in teaching students

to learn vocabulary. The tool has primarily been designed to work with students in grades 3 - 5, but can also be scaled for other learning groups. The overarching goals of SENCE are:

- To lower the manual intervention for educators in identifying important words and associated sentences from a given text.
- To assist educators in identifying and grouping key sentences into simple, medium, and complex sentences based on four different sorting criteria.
- To aid a student in learning the meaning of a word as it is applied in a sentence at their own pace.

To achieve these goals, SENCE has a four-pronged approach:

- Create an accessible interface for non-NLP experts to interface with an NLP program.
- Assess how an intelligent system can receive and learn from human edits.
- Assess the efficiency of an AI-assisted program to parse different formats of text.
- Provide individualized reinforcement of correct vocabulary usage.

SENCE was created with the help of open-sourced natural language processing libraries such as Natural Language Toolkit (NLTK) [5], SpaCy [22], and Stanford Parser [27]. This was an intentional design choice to make this tool accessible and free to all populations while at the same time using libraries that have already undergone rigorous testing and validation by the software community.

1.1 Motivation

SENCE is built on the foundations of Vocabulary Coordinator (VocaCoord) [21]. VocaCoord is the result of a long-term collaborative project between Dr. Michael

Berry [32], Professor of Computer Science at the Department of Electrical Engineering and Computer Science, University of Tennessee, Knoxville, and Dr. Jillian McCarthy [11], Associate Professor in the College of Health Professions and Director of the UT Augmentative and Alternative Communication, Language, and Literacy (UT-AACL) laboratory [58] at the University of Tennessee, Health Science Center. VocaCoord is a novel software that uses speech-to-text to display unique vocabulary in real-time. It was built to assist children who use augmentative and alternative communication (AAC) [36] to learn vocabulary faster. AAC includes any communication method that supplements (augments) or replaces (provides an alternative to) the usual methods of speech and/or writing where these are impaired or insufficient to meet the individual's needs. AAC may also have a role in cases where the child has difficulty learning speech and language. In this and other cases, children may use AAC in the short to medium term or as their life-long means of communication. While AAC is usually thought of as an 'output' system, a means for someone to express themselves, AAC can also be used to support the person's understanding of language and communication [37]. VocaCoord serves as a teaching aid to display real-time visual representations of words being taught in a lesson. VocaCoord takes academic vocabulary words spoken by a teacher and displays them in written form, providing a visually static text representation of a dynamic and very temporary auditory event, along with a definition and a picture of the word. This additional visual cue aids the student in remembering the meaning of the word faster than just the audio cue.

SENCE is a variant of VocaCoord. It automates some of the manual intervention processes that VocaCoord relies on for every lesson. While VocaCoord provides an additional visual cue to reinforce the knowledge of a word, SENCE uses NLP methods to provide individual reinforcement and assessment of continued memory of word meaning. It focuses on a child's progress in understanding the word being taught in familiar and unfamiliar sentences, as well as in a wide variety of simple to complex sentences. With VocaCoord, SENCE can provide an all-around, comprehensive aid to reinforce the meaning of words and assess a child's grasp and retention of a

word over time in multiple contexts. Another manner in which SENCE builds on VocaCoord is while VocaCoord provides a uniform platform for all students and does not change prompts based on the student's knowledge of vocabulary, SENCE automatically adjusts the type and complexity of sentences introduced to a student based on their comprehension level. This individualized attention allows students to learn at their own pace without feeling discouraged. At the same time, instructors can now assess each student's area of vocabulary strength and weakness and provide feedback as required.

The building of SENCE also provides an assessment of off-the-shelf NLP tools. In building SENCE, several off-the-shelf NLP packages such as SpaCY [22], NLTK [5], Stanford Parser [27], Stanza [43], Yake [9], Rake [45], Bert [12], and TextRank[30] were evaluated for accuracy in syntactic and semantic vocabulary parsing. A detailed analysis of this comparison is discussed in Section 7.2.1.

Finally, SENCE provides a pilot example of stitching several off-the-shelf NLP libraries together to form a comprehensive, user-friendly NLP tool. Not all tools perform all language tasks well. SENCE shows which packages outperform others for specific language tasks, and how they can work collaboratively to form a comprehensive educational aid for vocabulary learning.

The rest of this dissertation is laid out as follows: Chapter 2 and Chapter 3 discuss background work in audiology and speech pathology, and NLP. Chapter 4 describes the research problem addressed; Chapter 5 and Chapter 6 discuss SENCE architecture and database design. Chapter 7 is an extensive discussion of SENCE methods and results obtained, while Section 7.7 discusses results from media other than classroom lesson-based texts. Finally, Chapter 8 discusses the conclusions of this work and its broader impacts, and Chapter 9 discusses proposed future work.

Chapter 2

Background Work in Audiology and Speech Pathology

According to the National Institute of Health's National Institute on Deafness and Other Communication Diseases (NIDCD) fact sheet [60] on Specific Language Impairment (SLI), Specific language impairment (SLI) is a communication disorder that interferes with the development of language skills in children who have no hearing loss. SLI can affect a child's speaking, listening, reading, and writing. SLI is also called developmental language disorder, language delay, or developmental dysphasia. It is one of the most common developmental disorders, affecting approximately 7 to 10 percent of children in kindergarten. According to Bishop in 'What Causes Specific Language Impairment in Children?' [6], Children who are unable to speak because of physical disability, and those who cannot hear what others say to them, will nevertheless learn to communicate by other means, provided they are exposed to alternative systems of communication such as sign language. There are, however, exceptions to this general rule of speedy and robust language acquisition: Children with specific language impairment (SLI) have major problems in learning to talk, despite showing normal development in all other areas. Thus, a typical 7- or 8-year-old child with SLI may talk like a 3-year-old, using simplified speech sounds, with

words strung together in short, ungrammatical strings—e.g., "me go there," rather than "I went there." From Storkel et al [55], Children with SLI are slow to learn new words, needing 2–3 times as many exposures as their peers [19]. Vocabulary deficits, in turn, affect reading decoding and comprehension [38], leading these children to fall further behind the academic achievement of their peers [35]. In addition to this academic cost, there is a social cost to vocabulary deficits, with low vocabulary being linked to low popularity among peers [18]. The previously mentioned Storkel study presented the results of a preliminary clinical trial with twenty-seven kindergarten children with SLI. The trial used interactive book reading to teach 30 new words. Word learning was assessed at 4 points during treatment through a picture naming test. In this study, they determined that 36 exposures to a word were the ideal number of exposures for a child with SLI to learn the meaning of a word.

Chapter 3

Background Work in Natural Language Processing

Mimicking the ways humans learn and understand language has been one of the greatest challenges of Natural Language Processing (NLP). Not only does the system have to learn grammar, but more importantly, the system has to learn context. Much work has been done in NLP in the areas of not only teaching a machine words but also grammar and context. Word Sense Disambiguation (WSD) [29] is the method of teaching a system the actual meaning of a word according to its context. In Rahman et. al, [44] an unsupervised learning method to detect the correct sense of a word in a sentence is presented. To correctly classify word sense, the method uses WordNet [31], NLTK [5], and co-location scores. As an extension to this, Jordan and Mitchell [26] provide an NLP approach to author and mark short answers in freeform text. They used the UK's Open University's Intelligent Assessment Technologies (IAT) engine [24], coupled with the OpenMark assessment system [8] to grade free-form text answers from students. This was accomplished by creating human-curated templates that the machine would then use to grade students' answers. An example of a template that is based on the answer 'Lava erupts from the mountain' is shown in Figure 3.1.

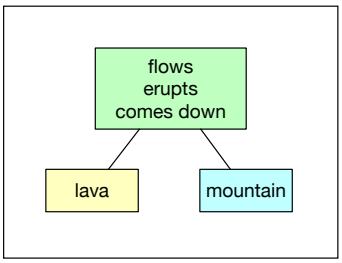

Figure 3.1: Example of an answer template.

The IAT system breaks down the answer from the student into phrases and then compares how closely the answer matches the corresponding human/Subject Matter Expert curated template. Each imperfect answer is responded to with feedback from the OpenMark tool, with prompts towards the correct answer. If a student is close to the right answer, the Open Mark tool allows the student to retry answering the question a preset number of times. Again, these prompts are human-created, and the system picks the prompts based on the answer provided by the student.

There have been significant strides in the area of AI tools to assist in NLP tasks. Primary among them is BERT [12]. BERT (Bidirectional Encoder Representations from Transformers) applies the bidirectional training of Transformer, a popular attention model, to language modeling. This is in contrast to previous efforts, which looked at a text sequence either from left to right or combined left-to-right and right-to-left training. Horev in [23] shows that a language model that is bi-directionally trained can have a more profound sense of language context and flow than single-direction language models. In [25], Jawahar et al. discuss the suitability of BERT for various NLP tasks such as parsing phrasal syntax, surface tasks such as sentence length and presence of word in a sentence, syntactic tasks such as the depth of the syntactic tree, and semantic tasks such as checking for tense and sensitivity to random replacement of a noun or verb.

Outside of BERT, there have been several other NLP tools to assist in feature extraction tasks. Some of them are RAKE, YAKE, TextRank, and SgRank [10]. Rapid Keyword Extraction algorithm (RAKE) uses co-location and co-occurrence of

words to extract high-frequency words. Yet Another Extraction Algorithm (YAKE) is a lightweight, unsupervised automatic keyword extraction method that relies on statistical text features extracted from individual documents to identify the most relevant keywords in the text. After pre-processing the text, YAKE uses five features - TCase (Casing), TPos (Term Position), TFNorm (Term Frequency Normalization), TRel (Term Related To Context), and TSent (Term Different Sentence) to rank keyterms [53]. TextRank, based on Google's PageRank algorithm [39], is a graph-based keyterm extraction algorithm. In this algorithm, text units that best identify the text at hand are chosen as vertices, and the relationship between them is represented by the edges. The graph-based ranking algorithm is then iterated until convergence, and the top x vertices are picked as keyterms after post-processing. SGRank, based on TextRank, also uses unsupervised learning, and combines graphical and statistical methods to identify keywords in a given text.

The CoreNLP [28] is a major milestone in the evolution of natural language parsers due to its high accuracy, extensive language support, flexibility in output formats, well-documented features, and robust research-backed foundation. A natural language parser is a program that works out the grammatical structure of sentences, for instance, which groups of words go together (as phrases) and which words are the subject or object of a verb. CoreNLP comes from the Stanford NLP Group - a part of the Stanford AI Lab (SAIL). It includes members of the Linguistics Department, the Computer Science Department, the Psychology Department, and the Graduate School of Education, among others, who work together on algorithms that allow computers to process, generate, and understand human languages. CoreNLP is a one-stop shop for natural language processing written in Java. It enables users to derive linguistic annotations for text, including token and sentence boundaries, parts of speech, named entities, numeric and time values, dependency and constituency parses, co-reference, sentiment, quote attributions, and relations. CoreNLP currently supports eight languages: Arabic, Chinese, English, French, German, Hungarian, Italian, and Spanish.

spaCy is a popular Python library that contains the linguistic data and algorithms that can process natural language texts. spaCy is easy to use because it provides container objects that represent elements of natural language texts, such as sentences and words. These objects, in turn, have attributes that represent linguistic features, like parts of speech. spaCy offers pretrained models for English, German, Greek, Spanish, French, Italian, Lithuanian, Norwegian Bokmäl, Dutch, and Portuguese. In addition, spaCy offers built-in visualizers that can be invoked programmatically to generate a graphic of the syntactic structure of a sentence or named entities in a document. The spaCy library also natively supports advanced NLP features such as word and sentence vectors [61].

Chapter 4

Research Problem

The rapid advancements in NLP have paved the way for developing intelligent systems capable of understanding and generating human language. However, there remains a significant gap in making these technologies accessible and usable by individuals who are not experts in AI or computer science. The proposed research aims to address this gap by focusing on four key areas:

4.1 Accessible interfaces to an NLP solution

SENCE creates a human-friendly interface to a series of interlinked NLP solutions. Through multiple prompts that mimic human behavior, SENCE guides a user who is unfamiliar with AI through word and sentence selection prompts, thereby creating an easy-to-use, easily navigable system to create personalized lesson aids. This involves designing a user-centric interface that simplifies interaction with complex NLP models. This democratizes access to advanced language processing tools.

4.2 Blending human input with NLP responses

At each stage of data curation and sentence capture, SENCE allows for human input into the correctness of its selections. It then stores this feedback and takes it into

account the next time a similar choice is to be made by the program. This allows SENCE to improve its efficiency and accuracy over time. Allowing for human input in the word and sentence selection processes allows for language expertise to strengthen the output of SENCE, something that it does not have on its own. This combined human and NLP system allows for scalable and personalized lesson aids while allowing for human expertise in the process.

4.3 Assessing how NLP tools can parse different media formats

This research investigates how efficiently and accurately off-the-shelf NLP tools can parse different formats of text media. Formats of text media, in this sense, mean text from textbooks, online articles, transcripts from TV shows, and storybooks. SENCE seeks to answer the research question: Can a data curation process that works for one type of media work as effectively for another without any special tuning?

4.4 Scaffolding complexity to reinforce vocabulary

SENCE provides vocabulary knowledge assessments to students with scaffolding in increased complexity. This way, a student is guided through simpler sentences before they are asked to correctly identify vocabulary usage in more complex sentences. Additionally, students are shown sentences from the passage that they have just encountered, and finally shown the same keywords in sentences that are from passages unknown to them. This approach tests a student's capacity to correct use and identify the meaning of words in familiar and unfamiliar settings. Progress on students' usage patterns is stored and shown to instructors at the end of the lesson, so that instructors are aware of each student's progress in word awareness and know each student's areas of weakness and strengths for the words being taught.

Chapter 5

Proposed Architecture

The basic premise of SENCE is to be an educational aid for instructors teaching critical vocabulary to all students, including those with and without developmental language disorders.. This tool comes into use at the end of a lesson, where it reinforces students' vocabulary comprehension of the vocabulary just taught on an individual basis. In order to accomplish this, SENCE does the following:

5.1 Corpus parsing and storage

For all passages, and before selecting words and sentences for a specific lesson, SENCE does the following:

- Parse and store passages: SENCE parses a given passage and stores it in its database for future use. It cleans the text of special characters, extra spaces, and other formatting, storing the passage as plain text.

- Capture keywords from the passage: For a given stored passage, SENCE identifies all keywords in the text. For example, to teach a student about the words volcano, or lava, the instructor might choose a passage about mountains and volcanoes. SENCE will parse through the stored passage in its database

and capture and store these keywords. Instructors are given the opportunity to add additional keywords if necessary for the passage at this stage.

- Capture sentences associated with the keywords being taught: SENCE captures all the sentences in the lesson associated with the keywords and stores them in its database. This now serves as an exhaustive corpus for reinforcing students' correct vocabulary usage.

These three steps as shown in Figure 5.1 are done once for each new passage. Every time a passage is brought up to be taught, SENCE assesses if these steps have been performed. If they have, it skips to the next step. There is no user modification of sentences at this stage because the initial corpus is treated as the source of truth for the following features of SENCE.

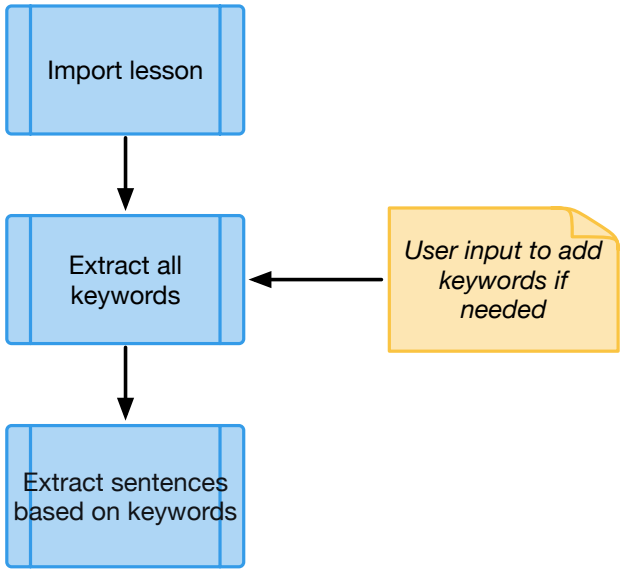

Figure 5.1: Data diagram of corpus parsing and storage

5.2 Lesson-specific keyword and sentence selection and storage

Once a lesson, its keywords, and associated sentences have been stored in its database, SENCE is now ready to perform lesson-specific tasks. The goal of this section is to pick keywords for a lesson and sentences associated with those keywords. Sentences are then classified as simple, medium, and complex sentences based on sort conditions. Currently, SENCE can classify sentences based on i) sentence length, ii) the syntactic dependency of a keyword relative to the sentence, iii) the number of Tier 2 and Tier 3 words in a sentence, and iv) the number of keywords in a sentence. In speech-language pathology, vocabulary is often differentiated on the basis of the commonality of usage of the word. They are, therefore, broken into tiers of usage or tier words. A deeper discussion of tier words is found in Chapter 6. A data diagram of the steps performed is shown in Figure 5.2.

SENCE performs the following steps in this phase:

- **Obtain lesson keywords and sort type from the instructor:** Instructors are requested to choose a passage from existing passages in SENCE. Once they choose a passage, they are shown the exhaustive list of keywords for this lesson. Instructors then pick the four or five words they want to teach in the current session. They also pick one of the four sort types for sentence classification.

- **Check if sentences have already been classified for this sort type:** SENCE next checks if sentences have already been classified for this sort type. If not, it retrieves all the sentences picked for this lesson from its database, picks the sentences for this sort type, and classifies and stores them as simple, medium, and complex sentences. Users can modify the sentences stored at this stage. For example, if SENCE picks a sentence "It spews lava," a user can store a modified version, such as "The mountain spews lava," of this sentence.

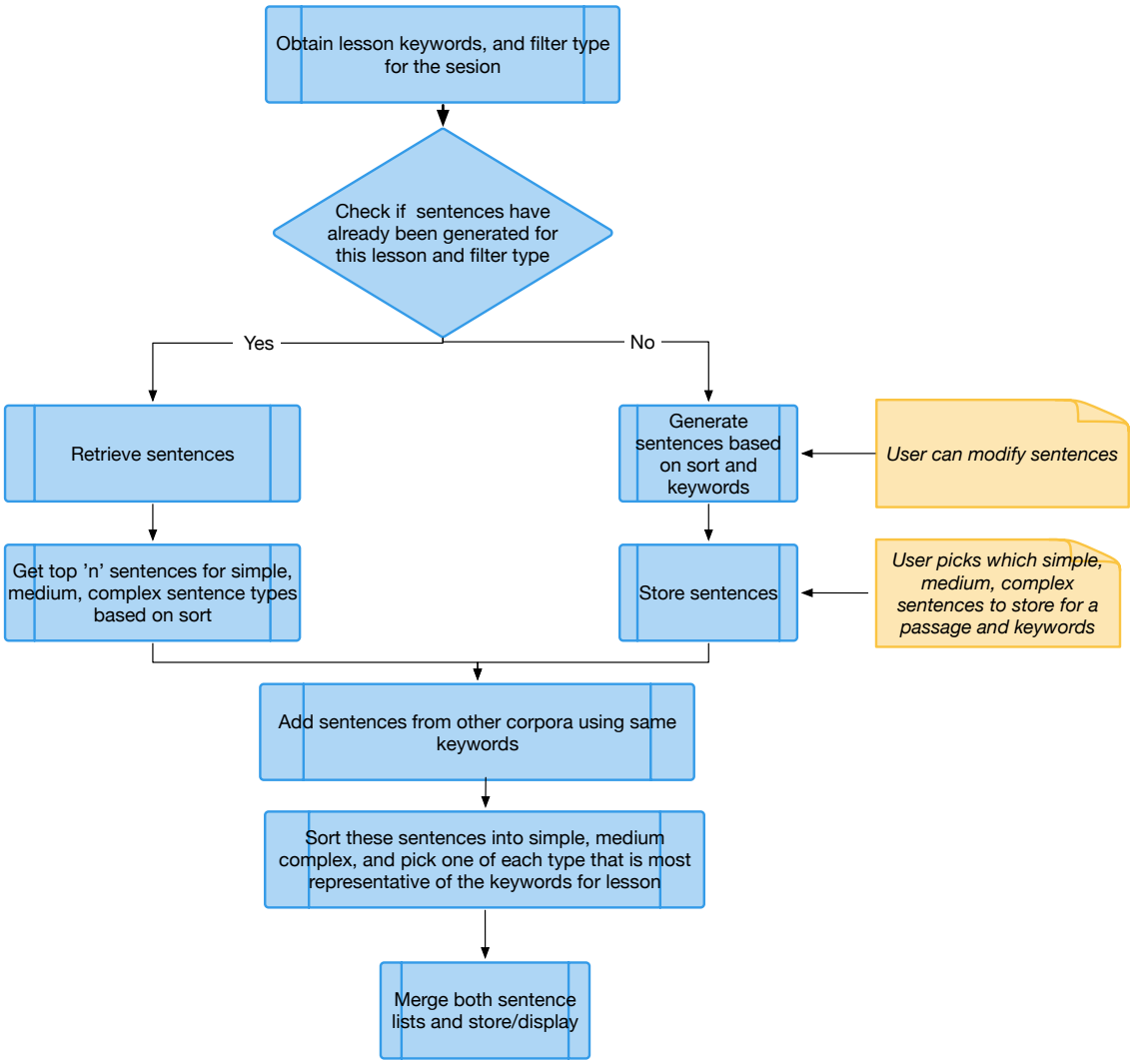

Figure 5.2: Flow of operations for sentence selection with sort applied.

Instructors can also pick which simple, medium, and complex sentences they want to store for the given sort type and passage. This optional down-select happens because it is possible that SENCE can generate more than the 4-5 sentences required for each complexity type, and it is left up to the instructor to select which sentences are the most representative of the words being taught.

- **Add sentences from other corpora to the current set of sentences:** In order to assess the student's range of comprehension of the word, SENCE picks sentences from other passages in its database that contain one or more of the words being taught in the lesson. This is then classified into simple, medium, and complex sentences based on the sort criteria chosen by the instructor.

- **Stores sentences for the lesson based on keywords and sort:** The final list of curated sentences from the lesson and from outside of it is stored in the database for the instructor to use in the current session and for the aid of future lessons based on the same keywords and sort criteria. This way, instructors can reuse previous iterations of the lesson without having to generate sentences from scratch for every lesson. Instructors can thereby build on the work of previous sessions.

5.3 Major Libraries and Software Used

SENCE uses MongoDB [33] as its database of choice. MongoDB is a lightweight NoSQL distributed database program. Because data doesn't need to fit within the confines of a strict relationship, MongoDB can operate as a general data store. This type of database provides several advantages. The primary benefit of using a NoSQL system is that it provides developers with the ability to store and access data quickly and easily without the overhead of a traditional relational database. They do not follow a rigid database schema, thereby making it quicker to add data in a different form without having to make any additional changes to the database. Additionally,

MongoDB stores records as JSON collections, making it easy to store and retrieve data from the database in Python.

SENCE uses spaCy [22] predominantly for all of its NLP work. spaCY is used to parse and clean text. It is used to remove 'stop words' when determining critical words in a passage. Stop words are high-frequency words in passages that are often removed before a passage is analyzed as they do not confer additional meaning to a text. In English, examples of stop words include:

- Articles: a, an, the
- Conjunctions: and, but, or
- Prepositions: in, on, at, with
- Pronouns: he, she, it, they
- Common verbs: is, am, are, was, were, be, being, been

SpaCY is also used for chunking text, finding the position of words in a sentence, and sentence editing. Details of how this is done are covered in Section 7.4.3 and Section 7.5.1 below.

Stanza is used in SENCE primarily to lemmatize text. It performs better than spaCY for accuracy in lemmatization. The usage of Stanza in SENCE is detailed in chapter Section 7.3.1.

`pyInputPlus` is a Python module to provide `input()` - and `raw_input()` -like functions with additional validation features. SENCE uses `pyInputPlus` wherever user input is required, so that SENCE can perform immediate, initial validation of user input before it is accepted by the program. `pyInputPlus` also provides a uniform interface for command-line interactions with the system. For instance, choices are presented as a numbered list, where the user only has to enter the choice number of the option they wish to select. This reduces the number of typos and other kinds of user-initiated errors. An example of this is shown in Figure 5.3. The user only needs

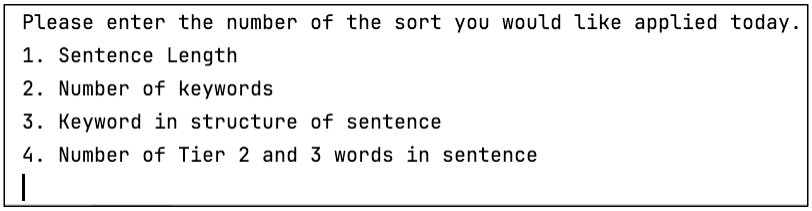

Figure 5.3: Example of `pyInputPlus` Choice menu

to type in the number 1-4 instead of the entire choice in words. `pyInputPlus` also has in-built validation methods to ensure only a choice of what is present is chosen. So, if a user enters the number 5 or an alphanumeric input, it will return with an error message and prompt the user to re-enter their choice.

Chapter 6

Database Design

This chapter describes the MongoDB collections (synonymous with a SQL table) in SENCE. The first three collections shown in Figure 6.1 are the *passage*, *keywords* and *tier_words* collections.

The *passage* collection stores the unedited corpus of text from the lesson. It stores the title of the lesson, the text, the total number of words in the passage, the total number of sentences, the average sentence length in the passage, and the length of the longest sentence in the passage. It also stores the grade level of the lesson if available. Finally, media in the collection refers to whether the lesson comes from a storybook, a textbook, or a television show.

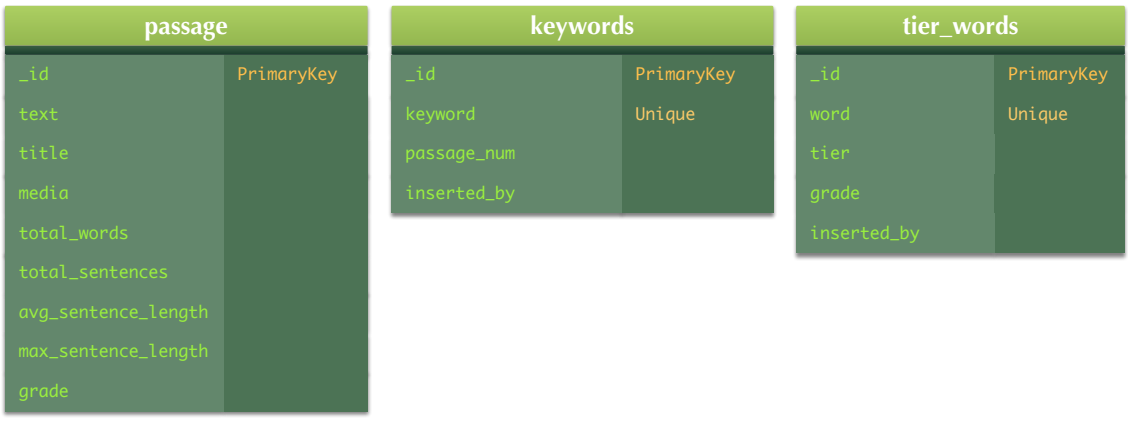

Figure 6.1: Data model diagrams of passage, keywords, and tier_words collections.

The *keywords* collection is a comprehensive store of all the critical words identified for a given lesson. It also stores which lesson the keyword is associated with, and if the keyword was identified by the system or added later by an instructor.

The *tierwords* collection is independent of the other two collections. It is a comprehensive store of tier 2 and tier 3 words. In speech-language pathology, vocabulary is often differentiated on the basis of the commonality of usage of the word. They are, therefore, broken into tiers of usage, or tier words. Tier 1 words are basic words that are commonly used in everyday language. Examples are *run, see, book, and school*. Tier 2 words are high-frequency words used by language users across content areas. Because they are not used in spoken language as frequently, Tier 2 words can present challenges to some students when they are used in text.

Some examples of tier 2 words are *cite, formulate, and evaluate*. Tier 3 words are not used frequently except in specific content areas. These words are necessary for understanding the content presented in academic areas. Examples are *respiration, pi, amendment, and protagonist*. Tier words serve as a useful indicator of the complexity of the vocabulary being taught in the lesson and, therefore, one of the sort types used in selecting sentences for vocabulary comprehension assessment in SENCE. In addition to the 2506 tier 2 and 3 words identified in SENCE (1288 tier 2 and 1218 tier 3 words), instructors can add additional tier words as needed. Tier 2 and 3 words were collected from [7], [52], [57] and [56].

The next two collections in the SENCE database are the *sentences* and *modified_sentences collections*. The data model diagrams for these collections are shown in Figure 6.2. The *sentences* collection stores all the sentences that contain keywords identified for the passage. Aside from the sentence, this collection stores the ids of keywords, the length of the sentence, the passage id the sentence comes from, and the parts of speech (pos_present) present in the sentence. The *pos_present* field helps instructors sort sentences based on the parts of speech they would like to concentrate on for the lesson.

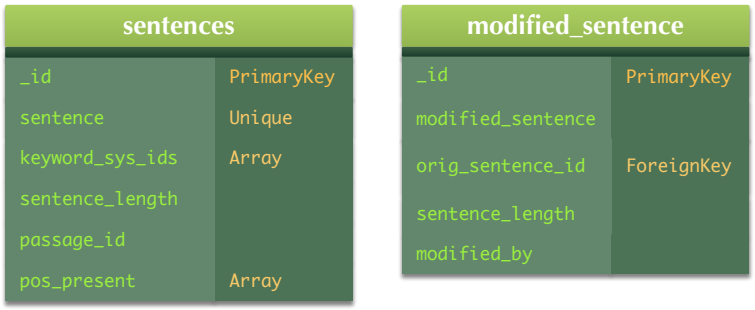

Figure 6.2: Data model diagrams of sentences and modified_sentence collections.

The *modified_sentence* collection stores versions of a sentence from the *sentences* collection that the instructor has modified. For example, if a lesson has a sentence, "They used music and dance in sacred rituals.", the instructor might want to modify the sentence to say, "The Cherokee used music and dance in sacred rituals." for added context. To maintain the body of truth in the *sentences* collection, the *modified_sentence* collection exists to store any modifications instructors might want for original sentences.

The final set of collections in SENCE is *sentences_for_passage* and *student_response*. The data model diagrams of these collections are shown in Figure 6.3. The *sentences_for_passage* collection stores all the sentences identified to be used for a lesson, down-selected from the comprehensive list of all sentences from a passage based on sort type. For example, for a lesson based on teaching the keywords 'lava', 'mountain', and 'volcano', there could be 15 sentences with those keywords in them. Since the instructor may not want to display all 15 sentences to a student, they ask for sentences sorted based on the length of the sentences and request only five sentences back. SENCE sorts, filters, and stores the sentences in the sentences_for_passage collection. This collection also stores the username of the instructor performing the keywords and sort criteria filtering for the lesson, the *_id* of the passage being taught, *sort_type*, *sentence_id*, and *difficulty_level*, i.e., if the sentence has been classified as a simple, medium, or complex sentence. This collection also identifies whether this was a system-sorted sentence or a user-added

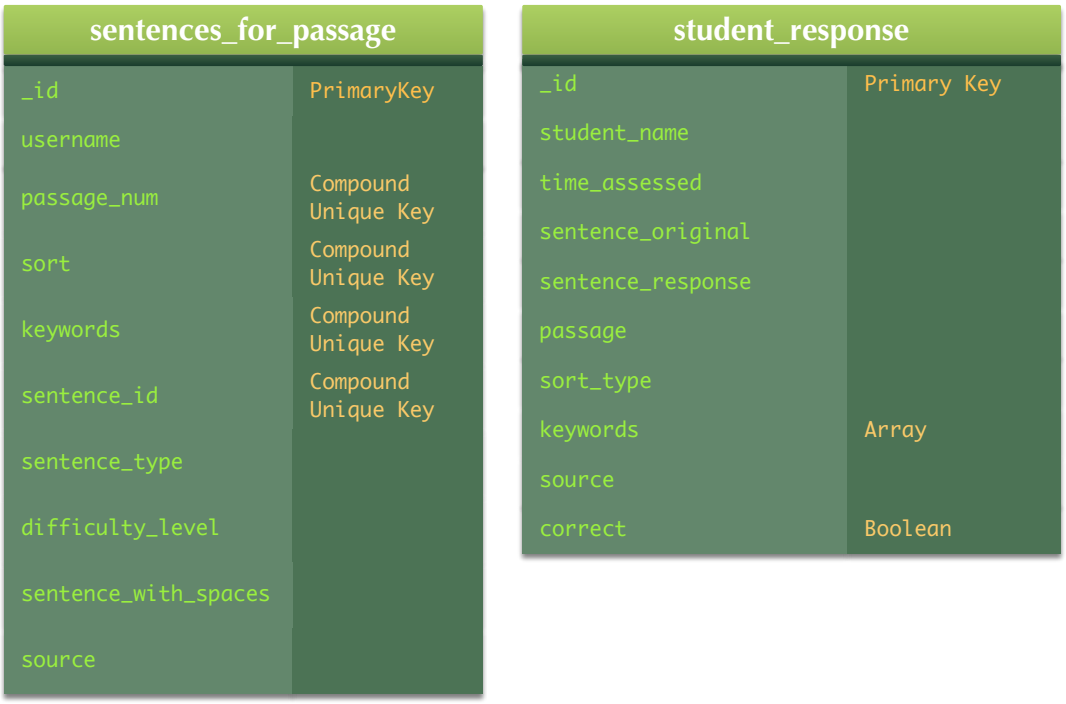

Figure 6.3: Data model diagrams of lesson, sentences_for_passage, and student_response collections.

sentence using the *sentence_type* field. The *sentence_with_spaces* field stores a version of the sentence with a string of dashes substituted for the keywords chosen for the lesson. This is used in the assessment phase of SENCE.

The final collection in SENCE is the *student_response* collection. This collection stores the results of how a student performed in correctly identifying the usage of a keyword in a sentence for a given lesson. This collection stores the name of the student (*student_name*), the name of the passage used for the lesson, the original sentence used for evaluation, the student's response, the keywords and sort type used for the assessment, whether the sentence came from the lesson taught or outside corpora, and a value of True or False if the student filled in the right or wrong keywords. This helps instructors evaluate if the student has a set of keywords or a style of sentences that presents more difficulty than others.

Chapter 7

Description of Methodology with Results

This chapter will describe the methodology used to implement the SENCE and the results obtained at each step. It will also outline the challenges encountered and measures taken to circumvent them.

7.1 Corpus parsing and storage

The first step in implementing SENCE was to create a method to extract text, clean and parse it, and store it with relevant metadata in MongoDB. For this section, and the sections in this chapter that follow, the passage shown in Figure 7.1 will serve as the base example.

As one can see, the text is heavily formatted, with differing font sizes, meta information like chapter number, grade level of reading, source of material, and the name of the author. The text is also bolded in some areas, and regular in the rest. Finally, the text also has information such as **Side note fact:" that does not constitute separate, stand-alone sentences.

Erupt!

Joan Marie Galat

National Geographic Kids – Reading Level 3

CHAPTER 1:

OUR FIERY WORLD

Several times an hour, lava shoots out of a volcano in Italy called Stromboli. Stromboli has been spewing gas and spitting molten rock for more than 2,000 years! It is one of Earth's most active volcanoes. Full eruptions can occur from only minutes to hours apart!

Volcanoes begin deep underground. Hot liquid rock pushes upward. It escapes in a powerful burst. Lava, ash, and steam pour from the mountain. In an instant, the landscape around the volcano changes.

****Side note fact: A volcano can destroy and entire town. **Side note fact: Besides Stromboli, a volcano erupts somewhere on Earth every week. Earth's surface is made of giant slabs of constantly moving rock. These giant pieces of rock, or tectonic plates, cover the globe. They form the crust, our planet's thin outer layer. The crust floats on the mantle – 1,800-mile-thick layer of hot liquid, or molten, rock called magma. The mantle flows just enough to slowly move the plates. Gravity causes the heavy plates to sink slowly into the mantle. Inch by inch, the plates have crept along for hundreds of millions of years.**

****Side note fact: The magma that forms most volcanoes comes from just a few miles below Earth's surface. **Side note fact: Tectonic plates are hundreds to thousands of miles across and 10 to 125 miles thick.**

BENEATH YOUR FEET Moving tectonic plates can make mountains and volcanos. Tectonic plates don't just float along. They bump, pull apart, and slide below one another. Plates crashing together can buckle and ground to form mountains. A volcano can form when a tectonic plate is forced downward into the mantle.

First, heat and pressure inside Earth melt part of the plate, forming magma. The new magma creates pressure. The pressure helps force the magma upward. If a spot in Earth's crust has a hole, called a vent, molten rock and ask can
escape. That's a volcano!

When the rising magma escapes through the top of the volcano's vent, it gets a new name. Now it is lava.

****Side note fact: Heat inside Earth can create hot springs and bubbling mud**
pools around the volcano.

When hot magma rises through the crust, instead of the edges of tectonic plates, it is called a hot spot. Volcano mountains can form when heat and pressure, miles below Earth's crust, form magma that rises through cracks in the crust. Shallow areas of magma collect beneath the crust and ooze upward through the cracks.

****Side note fact: A collapsing volcano can form a caldera more than 60 miles wide. If the magma reaches a vent, a hole that runs from deep underground to Earth's surface, a volcanic eruption occurs. Some eruptions make an explosion. Others**
cause lava to simply flow out. A violent explosion can cause a volcano to collapse and form a giant bowl-shaped area called a caldera.

****Side note fact: Lava turns to solid rock as it cools. Each eruption spreads more lava and makes the mountain grow larger.**

CONES AND DOMES

There are four main types of volcanoes. Cinder cone volcanoes form when exploding lava hardens into glassy rock
fragments. The falling cinders create a steep, cone-shaped hill around the vent.

****Side note fact: Cinders contain gases that look like bubbles frozen in rock. Composite volcanoes form mountains with separate layers of lava, ash cinders, blocks, and bombs.**

Shield volcanoes form when lava flows in all directions. The liquid rock hardens into a large wide cone instead of a cone with steep sides.

Volcanoes with lava domes form when lava is too thick and sticky to flow very far. Towering domes may reach more than half a mile high.

****Side note fact: Some lava domes have been growing for 100 years.**

Figure 7.1: Text of the Erupt lesson that is used for examples in the rest of this chapter.

In order for SENCE to properly parse sentences and present them to instructors and students in a way that makes sense, this text needs to be cleaned before it can be used further. There are two types of text cleaning performed in this work - automated and manual.

The text was available as a Word document. This was then copied to a plain text editor to remove all the formatting, resulting in a form that can be ingested by SENCE.

7.1.1 SENCE corpus parsing

SENCE corpus parsing is done by the `insertLessonsToMongo.py` program. When run, the program checks to make sure the instructor has selected the correct program to run, then requests the title, grade, and the way the instructor would like to upload the lesson. There are two ways the instructor can upload a lesson. One is by providing a file path on the system where SENCE is being used, and the second is by directly pasting the text into the program. In the example as shown in Figure 7.2 below, the instructor chose to save a passage with the title of "Erupt-chapter 1" for lesson grade 3 and chose to enter the passage contents by providing a file path.

```
Would you like to import a lesson today? yes
Please enter the title of the text: Erupt Chapter 1
Please enter the lesson grade: 3
Please enter the number of the way you would like to import the lesson:
 1. File path
2. Enter text here
1
Enter file name with the full path (eg: C:\path_to_file\documentname.extension): erupt_1.txt
```

Figure 7.2: Uploading a passage to SENCE

7.1.2 SENCE text parsing

Once the passage has been entered by the instructor, SENCE then performs the following steps to clean, parse, and collect meta information about the content before saving it in the MongoDB collection called passage.

- First, all extra whitespace, such as multiple spaces or newline characters, is removed.
- Then, meta-information about the passage is collected. This information includes word count, sentence count, average sentence length, and the length of the longest sentence in the passage.
- Finally, the cleaned passage and meta information are stored in the *passage* collection as shown in Figure 7.3.

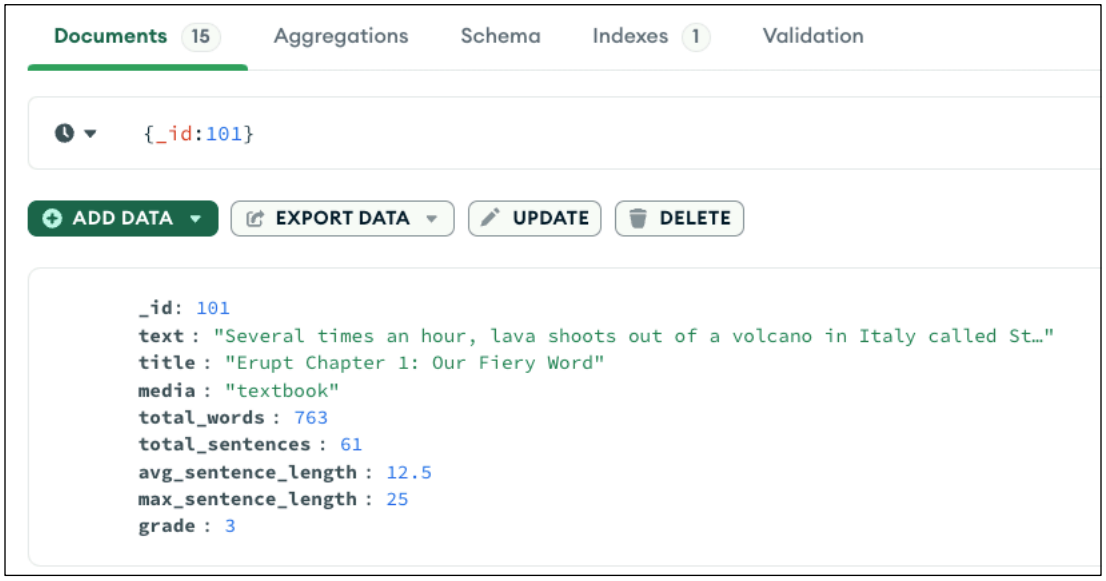

Figure 7.3: Erupt! Chapter 1 stored in the MongoDB Passage collection

7.1.3 Error Checking

SENCE uses PyInputPlus for initial error checking of user input. As shown in Figure 7.4, SENCE ensures only numeric responses for grade level and import options. SENCE also checks to ensure that the file path option is valid, and prompts the user to reenter a valid file path if the previous file path entered was inaccurate, i.e., it leads to FileNotFound errors.

7.1.4 Collection constraints

The passage collection has one constraint - a unique constraint on the *_id_* column as shown in Figure 7.5. This constraint ensures the *_id_* is a unique primary key for the collection, i.e., each row in the collection can be uniquely identified by the *_id_* column alone.

```
Would you like to import a lesson today? yes
Please enter the title of the text: Erupt - Chapter 1
Please enter the lesson grade: a
'a' is not a number.
Please enter the lesson grade: 3
Please enter the number of the way you would like to import the lesson:
 1. File path
2. Enter text here
something else
'something else' is not a valid choice.
Please enter the number of the way you would like to import the lesson:
 1. File path
2. Enter text here
1
Enter file name with the full path (eg: C:\path_to_file\documentname.extension): erupt_1
We did not find a file at that location. Would you like to try again? Please input yes or no: yes
Enter file name with the full path (eg: C:\path_to_file\documentname.extension):
```

Figure 7.4: Error checking when uploading a passage to SENCE

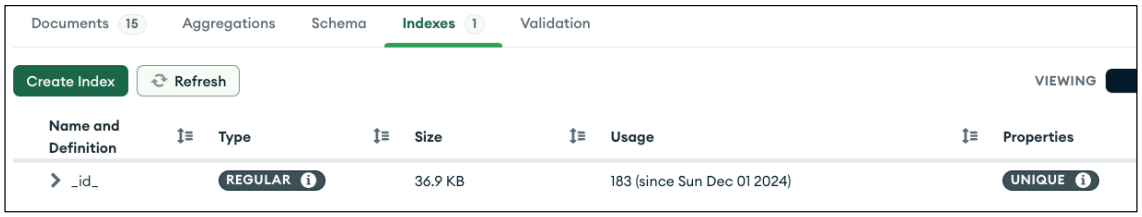

Figure 7.5: Passage collection constraints in MongoDB

7.2 Keyword extraction

The next step in the evolution of SENCE is the extraction of all keywords in the passage. Keywords for the purposes of SENCE are defined as critical words essential to the context of the passage. A subset of these keywords is intended to be used as words to be taught and reinforced to the student as a part of the lesson. For example, on a passage about volcanoes, keywords could be *volcano, lava, mountain, spew* and *hot*. The objective of this segment of SENCE is to automate the process of extracting keywords from a lesson, and then involve instructor input into adding additional keywords if they wish to do so.

SENCE first presents a list of all available passages in the SENCE MongoDB database. It then requests the instructor to choose a passage to extract keywords for, as shown in Figure 7.6. It then checks to see if keywords have already been extracted for the passage. If they have not, SENCE performs the following steps:

- The passage is first converted to lower case. This is to make sure that SENCE treats words such as “hello”, “Hello”, and “HELLO” the same.

- The passage is then lemmatized. Lemmatization maps a word to its lemma (dictionary form). It does this with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the lemma [46]. For instance, the word ‘was’ is mapped to the word ‘be’. The goal of lemmatizing the passage is to reduce the number of words in the passage to only the set of words that are truly unique. For example, the sentence “The mountain erupted

today, but it did not erupt yesterday." is lemmatized to "the mountain erupt today, but it do not erupt yesterday.". Lemmatizing is a necessary first step in keyword extraction because it prevents different forms of a word from being counted as different words.

- Stopwords are then removed from the lemmatized passage. Stop words are commonly used words, such as prepositions and articles, that add little value in identifying keywords.

- The text is then parsed to first identify the part of speech tag of each word, and the words that are identified as adjectives, adverbs, verbs, nouns, and proper nouns are chosen. Please note that for this version of SENCE, only adjectives, adverbs, verbs, nouns, and proper nouns are included in keyword selection; this can be modified to add other parts of speech if necessary.

- This list of words is then ordered by the frequency of appearance of each unique word in the list. The top ten are chosen and are presented as the keywords for the passage.

If the passage already has keywords available for it in the database, SENCE presents that list of keywords along with what other instructors might have added for the passage. The instructor is then asked if they would like to add additional keywords to this list. This is shown in Figure 7.7.

The SENCE collection, called *keywords*, is used to store all the keywords for a passage. An excerpt of stored keywords for the Erupt! passage referenced above is shown in Figure 7.8. An interesting database design feature of how keywords are stored in this collection is the approach to reduce the redundancy of words in the collection.

```
Index number Title                                                    Grade
101          Erupt Chapter 1: Our Fiery Word                          3
102          Erupt Chapter 2: DANGER                                  3
103          Water Takes Three Forms                                  2
104          The Invention of Paper                                   2
105          Louis Armstrong                                          2
106          Light and Objects                                        3
107          The History of Juneteenth                                3
108          Musical Instruments: Brass Instruments                   3
109          The Milky Way Galaxy                                     4
110          American Government - James Madison: A Man with a Plan   4
111          The Music of the Cherokee                                4
112          What Is a Volcano?                                       4
113          The Amazing Water Molecule                               4
114          Jazz                                                     2
115          Paper Crane: A Symbol of Peace                           2
116          The Tale of Peter Rabbit                                 2
117          The Princess and the Goblin - Chapter 7 - The Mines      4
118          Bill Nye the Science Guy: Photosynthesis                 3
119          Continue to know with WHRO: Rocks!                       4
Enter the Index number of the chapter you want to create default keyword list for today:
114
```

Figure 7.6: The instructor asked to pick a passage for keyword extraction.

For example, the word 'mountain' could be a keyword for multiple passages. Instead of storing the keyword as a separate entry for each passage, SENCE stores each keyword identified in any passage only once in the collection. It uses the set [34] feature of MongoDB to instead store all the passages (*passage_ids*) where the keyword has been identified. This reduces duplication of keywords in the collection. SENCE also stores information in the *inserted_by* field whether the keyword was identified by the system (SENCE) or by a user.

7.2.1 Evaluation of different keyword extraction methods

Several off-the-shelf NLP packages were evaluated for keyword extraction correctness and how closely they resembled what a human would pick. The software packages evaluated were spaCy, Yake, Rake, Bert, Textacy, SgRank, and TextRank. Table 7.1 shows the underlying methods for key term extraction for each NLP package evaluated.

```
Enter the Index number of the chapter you want to create default keyword list for today:
112

System Keywords for this passage are:
volcano form lava magma erupt crack active chamber mantle


Others have added the following keywords:
mountain


Would you like to input other custom keywords? yes
Please input your keywords separated by a space. dormant
Please enter your usernametsamuel

Process finished with exit code 0
```

Figure 7.7: Showing results available and asking if the instructor would like to add additional keywords.

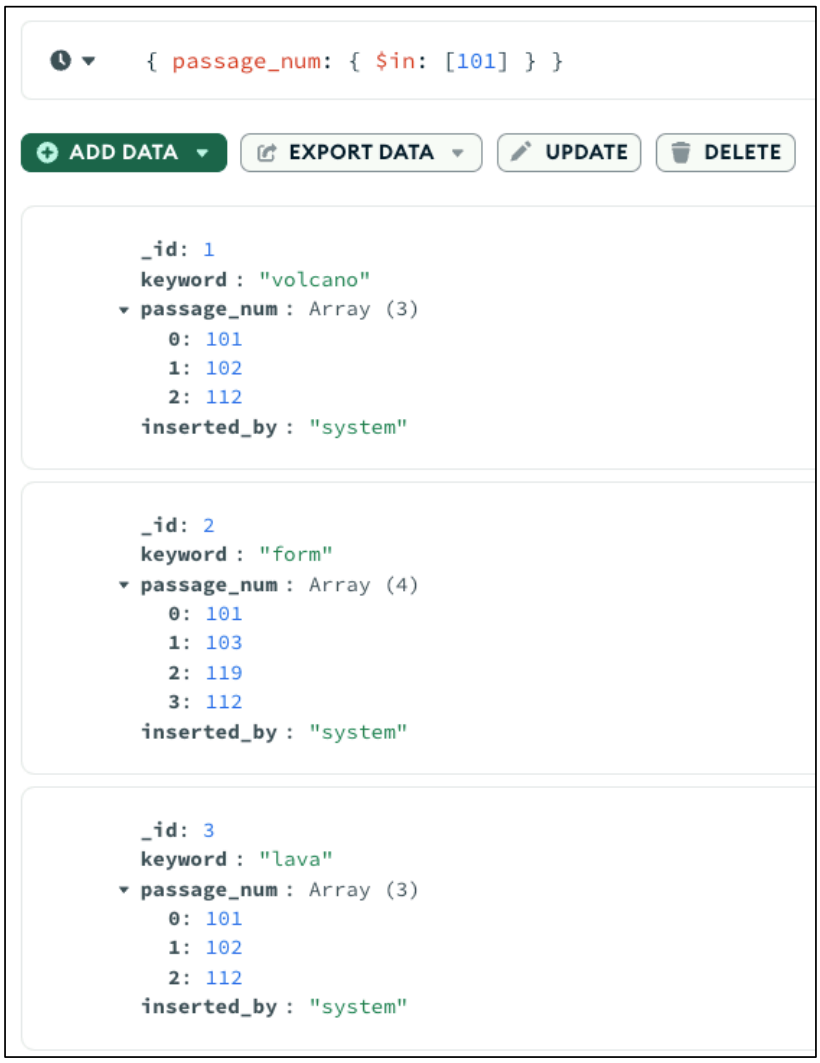

Figure 7.8: MongoDB collection for keywords.

Generally, statistical key term approaches such as those used by Yake and Rake solely rely on the statistical properties of word occurrences to identify key terms. Statistical properties include term frequency, word position, and co-occurrence. Statistical methods do not take into account semantic information or the relationship between words when identifying keywords. Graph-based methods build a graph where nodes represent words or phrases, and edges represent relationships based on co-occurrence within a window of words. Words that are highly connected are chosen as key terms. This approach adds limited semantic information while identifying key terms since it uses word relationships while choosing key terms. The most advanced of the three types are transformer-based approaches, which use deep learning to understand the context and semantics of text. Transformers capture the meaning of words based on their context within sentences, allowing them to understand nuanced meanings and relationships. This results in highly accurate keywords that reflect the richer semantic content of the text.

A subject matter expert in language and speech helped assess the quality of results and determine which packages most closely matched human choices. The comparisons were made for ten passages covering different subjects such as science, art (such as passages on music and dance), and social studies. Of all of these, the method that most closely resembled human choices was the method mentioned above using spaCy's implementation of lemmatization and part-of-speech identifier. Table 7.2 shows results from the Erupt! passage.

Table 7.1: Underlying algorithms of different NLP packages evaluated.

NLP Package	Underlying approach to key term extraction
Bert	Transformer
Rake	Unsupervised learning focusing on statistical properties
SgRank	combines statistical and graph-based methods
Textacy	graph-based method
TextRank	graph-based method
Yake	Unsupervised learning focusing on statistical properties
spaCy	uses both statistical models and transformer-based models

Table 7.2: Results of keyword extraction with different NLP packages.

NLP Package	Keywords extracted
Bert	volcano, eruption, volcanic, magma, erupt, stromboli, lava, molten, explosion, explode
Rake	pressure inside earth melt part, hot liquid rock pushes upward, look like bubbles frozen, ** side note fact, bubbling mud pools around, towering domes may reach, large wide cone instead, volcanoes begin deep underground, new magma creates pressure, cinder cone volcanoes form
SgRank	volcano, layer, change, rock, harden, form, cone, collect, plate, magma
Textacy	time, Cinder cone volcano form, lava domes form, hot liquid rock, active volcano, volcano change, composite volcano, lava harden, liquid rock harden, molten rock
TextRank	volcano, form, plate, lava, rock, magma, Earth, crust, mile, fact
Yake	volcano, Earth, fact, form, note, plate, side, lava, magma, rock
SENCE	volcano, form, lava, plate, rock, magma, earth, note, fact, crust, mountain

The remainder of the results are available in Appendix B. The next closest were TextRank, followed by BERT, though BERT seemed to prefer nouns dramatically to other parts of speech when selecting keywords from a passage. The choice was made to use spaCy instead of TextRank so as to minimize the number of libraries used in SENCE. spaCy is used extensively for other NLP tasks in SENCE, and therefore, it was practical to utilize it for keyword selection as well. Additionally, TextRank and SENCE are the only two methods that allow a user to choose which parts of speech can be allowed in keyword selection. This allows for specific subsets of words to be considered for selection. For example, if a user only wanted to consider nouns, adjectives, adverbs, and verbs for keyword selection, they can do so with SENCE and TextRank. This cannot be accommodated in other methods. Comparison of results between SENCE, TextRank, and BERT across three passages is shown in Table 7.3. Full text of the passages is available in Appendix C.

Figure 7.9 shows a comparison of the average number of nouns, proper nouns, verbs, adverbs, and adjectives chosen by each of the keyword algorithms used. This analysis was performed over 6 different texts - two with Science subject matter, two with Social Studies subject matter, and two with Arts subject matter.

It can be seen from this table that BERT and SgRank predominantly favor nouns and adjectives, at the cost of verbs. This could be because BERT is a transformer-based model that uses contextual information and semantic importance from a passage to perform text analysis. Nouns and adjectives typically carry significant semantic weight and contextual relevance in a sentence. Hence, it leads to BERT choosing nouns and adjectives over verbs and adverbs.

Yake, TextRank, and SENCE perform similarly overall; SENCE does better at selecting a variety of parts of speech as key terms when it comes to non-science topics than Yake and TextRank. This can be attributed to the fact that Yake and TextRank use statistical and graph-based methods, respectively, for key term extraction. They, therefore, do well when keywords are more frequent, such as those in a science lesson.

Table 7.3: Comparison of keyword extraction from three passages between BERT, TextRank and SENCE

Passage: Light and Objects	
BERT	light, skylight, glass, transparent, sunlight, opaque, shadow, happen, window, sky
Text Rank	light, object, opaque, transparent, path, glass, skylight, happen, line, travel
SENCE	light, object, transparent, path, pass, opaque, hit, glass, happen, line
Passage: American Government - James Madison: A Man with a Plan	
BERT	state, confederation, independence, government, national, nation, constitution, constitutional, country, law
Text Rank	state, government, Madison, United, convention, State, national, leader, agree, set
SENCE	state, madison, government, national, virginia, convention, plan, united, agree, constitution
Passage: Water Takes Three Forms	
BERT	boil, evaporate, gas, liquid, water
Text Rank	change, form, freezer, gas, happen, heat, ice, liquid, solid, turn, vapor
SENCE	change, form, gas, ice, liquid, shape, solid, turn, vapor, water

Overall	NOUN	PROPER NOUN	VERB	ADVERB	ADJECTIVE
Yake	6.8	0.5	1.7	0.2	0.8
Bert	7.0	0.5	1.0	0.0	1.3
SgRank	8.7	0.0	0.7	0.0	0.7
TextRank	7.2	0.2	2.0	0.0	0.7
SENCE	7.5	0.2	2.2	0.0	0.3
Social Studies	**NOUN**	**PROPER NOUN**	**VERB**	**ADVERB**	**ADJECTIVE**
Yake	5.5	1.5	2	0	1
Bert	7	1.5	1	0	0
SgRank	8.5	0	0	0	1.5
TextRank	6	0.5	2.5	0	1
SENCE	6.5	0.5	3	0	0
Arts	**NOUN**	**PROPER NOUN**	**VERB**	**ADVERB**	**ADJECTIVE**
Yake	7	0	1.5	0.5	1
Bert	7	0	0	0	3
SgRank	9.5	0	0.5	0	0
TextRank	8	0	1.5	0	0.5
SENCE	7.5	0	2	0	0.5
Science	**NOUN**	**PROPER NOUN**	**VERB**	**ADVERB**	**ADJECTIVE**
Yake	8.0	0.0	1.5	0.0	0.5
Bert	7.0	0.0	2.0	0.0	1.0
SgRank	8.0	0.0	1.5	0.0	0.5
TextRank	7.5	0.0	2.0	0.0	0.5
SENCE	8.0	0.0	1.5	0.0	0.5

Figure 7.9: Analysis of number of parts of speech per keyword extraction method across different passages.

However, non-science-based passages, such as those describing music or art, have more descriptive language that requires a deeper understanding of the relationship between words to retrieve a richer distribution of key terms. Transformer-based approaches such as BERT and spaCy, which use a mix of transformers and graph-based approaches, perform better at this task due to their complex, pre-trained models performing key term extraction tasks.

Another advantage of SENCE over multi-word phrase generators, like *cinder cone volcanoes form* and *pressure inside earth melts* (see Table 7.2) is that they introduce more tier 2 and tier 3 words instead of more complex phrases that are more appropriate to the target audience, i.e., children between grades of 3 to 5 who use augmentative and alternative communication.

An opportunity for further improvement when retrieving keywords is to group composite words together instead of highlighting them as separate words. For example, composite words like 'tectonic plate' or 'Milky Way' are currently identified as two different words by SENCE. It would be helpful to have them identified as a singular composite word, as that is how they appear in a passage.

7.2.2 Error Checking

As shown in Figure 7.10, SENCE ensures that only numeric responses are given for index number inputs and only a 'yes' or 'no' answer is given for the Yes/No question.

7.2.3 Collection Constraints

There are two constraints on this *keywords* collection as shown in Figure 7.11.

The *_id* and *keyword_text* columns have unique constraints on them. This constraint on the *_id* column serves as a primary key, and the constraint on the *keyword_text* column prevents users from adding duplicate values.

```
114          Jazz                                                    2
115          Paper Crane: A Symbol of Peace                          2
116          The Tale of Peter Rabbit                                2
117          The Princess and the Goblin - Chapter 7 - The Mines     4
118          Bill Nye the Science Guy: Photosynthesis                3
119          Continue to know with WHRO: Rocks!                      4
Enter the Index number of the chapter you want to create default keyword list for today:
erupt Chapter 2: DANGER
'erupt Chapter 2: DANGER' is not a number.
102

System Keywords for this passage are:
volcano lava rock note fact ash volcanic gas flow hot


Would you like to input other custom keywords?
Blank values are not allowed.
Would you like to input other custom keywords? n

Process finished with exit code 0
```

Figure 7.10: Error checking when generating keywords for a passage

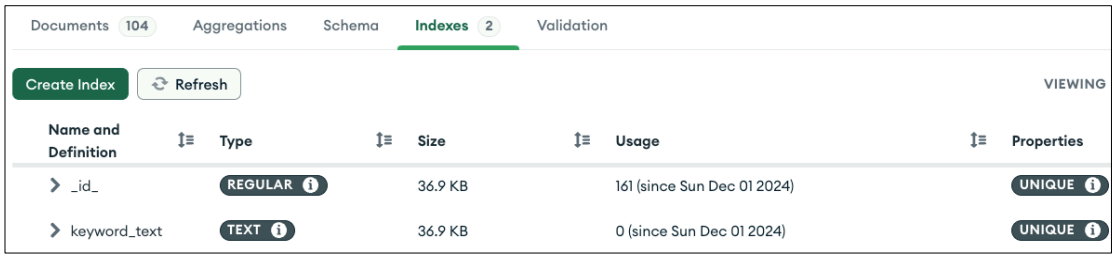

Figure 7.11: Constraints for 'keywords' collection

7.3 SENCE Sentence retrieval based on keywords

Once keywords are generated for a passage, the next step is to retrieve all the sentences in that passage associated with those keywords. Again, care needs to be taken to identify all forms of a base word in the sentence. For example, if one of the keywords is 'erupt', the sentence, 'The mountain erupted yesterday.' should be retrieved. The module `createSystemSentencesforMongo.py` retrieves all sentences that have the identified keywords in them. The module does the following steps:

- Asks the instructor to choose the passage to retrieve sentences.
- Retrieves all the keywords for the passage.
- If no keywords have been generated for the passage, it requests the instructor to first generate keywords, and then come back to retrieve sentences.
- If keywords are found, SENCE lemmatizes each sentence.
- Each lemmatized sentence is then checked against the list of keywords for the passage.
- If there is one or more matches to keywords for a sentence, the sentence is then inspected for different parts of speech contained in the sentence. As future work, sentence retrieval for student evaluation can happen based on the types of parts of speech present in a sentence.
- IDs of the matched keywords and passage containing them are retrieved, and stored in the MongoDB collection *sentences*. Only ids of the passage and keywords are stored in this collection since the original passage and keywords are already available in the *passage* and *keywords* collections, respectively. This way, the amount of storage needed to store these mappings is minimized. An example of two entries in the *sentence* collection in MongoDB is shown in Figure 7.12.

Figure 7.12: Example of two entries in the 'sentence' collection in MongoDB

This list of keywords stored in the MongoDB collection *sentences* serves as the source of truth for all further sentence retrieval and usage in SENCE. To preserve provenance, sentences are not modified at this stage for accuracy or better definition. Modifications of sentences are done at a later stage.

7.3.1 Choice of lemmatizer

While spaCy is used extensively in SENCE, the one area where it has a significant drawback is with lemmatizing words, specifically with lemmatizing words in capital letters. Using the sentence 'I love Dogs and Cats' as an example, spaCy produces 'I love Dogs and cat' as a lemmatized version of the sentence. It does not correctly lemmatize Dogs to dog. Stanza produces the accurate result 'I love dog and cat'. This difference in results can be attributed to the difference in underlying algorithms of how spaCy and Stanza handle lemmatization. spaCy uses a rule-based approach [48], while Stanza uses neural network models trained on Universal Dependencies (UD) treebanks for lemmatization [54].

A rule-based approach to lemmatization first performs a morphological analysis to identify the root forms of the word and its affixes. It then looks up a lookup table to find a match for the lemmatized word. If no match is found, it then applies linguistic rules to transform a given word to its lemmatized form. The advantage of rule-based methods is that they run faster and are simpler to use than neural network-based models. The disadvantage is that since they depend on pre-defined rules, rule-based algorithms may not handle irregular word forms or complex morphological transformations as accurately as neural network models.

Stanza uses neural network models for lemmatization. The training data is the Universal Dependencies treebanks [59], which contain sentences annotated with part-of-speech tags, morphological features, and lemmas. The neural network algorithms also place greater emphasis on the context of the word in the sentence, which produces more accurate lemmas. This allows Stanza to perform better than spaCy across a

variety of text and text forms, and therefore Stanza is used to lemmatize words in a sentence before keywords are extracted. The lemmatization process ensures that the keyword collection only has unique words as much as possible, and reduces the possibility of multiple forms of a word appearing as keywords.

7.3.2 Error checking

As with previous modules, SENCE uses pyInputPlus to ensure that only numeric values are entered for passage_ids, and prompts the instructor to reenter a value if a non-numeric value is entered.

7.3.3 Collection Constraints

There are three constraints in the *sentences* collection - a unique constraint on *_id_* to provide a primary key for the collection, a unique constraint on the *sentence* column to ensure there is only one copy of a sentence in the collection, and a text index on the same *sentence* column to facilitate quicker retrieval of a sentence from the collection based on a given search criteria. These constraints are shown in Figure 7.13.

7.4 Sort Criteria

The next step for SENCE after keyword and sentence extraction is to identify the best representative sentences of the passage to test the student's word comprehension of the keywords in the lesson. At this stage, the instructor is also requested to down-select the number of keywords to be used for that lesson. Four to five keywords

Name and Definition	Type	Size	Usage	Properties
> _id_	REGULAR	36.9 KB	251 (since Sun Dec 01 2024)	UNIQUE
> sentence_1	REGULAR	106.5 KB	86 (since Sun Dec 01 2024)	UNIQUE
> sentence_text	TEXT	147.5 KB	0 (since Sun Dec 01 2024)	

Figure 7.13: Constraints on the 'sentences' collection in MongoDB

are typically selected for a session in SENCE. Once keywords being taught in that session have been identified, there are two steps to creating the list of sentences that best represent this subset of keywords. The first is to identify a mechanism for picking sentences from the passage. These approaches are called 'sorts' in SENCE. For SENCE's current scope, four different sort criteria have been created.

1. Sentence Length: Group a set of sentences consisting of one or more of the keywords into groups of simple, medium, and complex sentences based on the length of the sentence.

2. Number of keywords: Group a set of sentences consisting of one or more keywords into groups of simple, medium, and complex sentences based on the number of keywords in the sentence, i.e., the more keywords in the sentence, the more complex the sentence.

3. Syntactic dependency of keyword relative to the sentence: Group a set of sentences consisting of one or more of the keywords into groups of simple, medium, and complex sentences based on the syntactic dependency of the keyword to the sentence. If the keyword is the subject of a sentence, it is classified as a simple sentence, if the keyword is the object of a sentence it is classified as a medium-complexity sentence, all other dependencies are classified as complex sentences.

4. Number of Tier 2 and 3 words in the sentence: Vocabulary is often differentiated on the basis of the commonality of usage of the word. They are, therefore, broken into tiers of usage or tier words. This is described in more detail in 3. In this sorting type, the objective is to group a set of sentences consisting of one or more keywords into groups of simple, medium, and complex sentences based on the number of tier 2 and tier 3 words in the sentence.

The following four sub-sections describe the implementation of these sorts in detail. `sentenceSorterByFilter.py` is the module that implements the sorting and is available in Appendix D.

7.4.1 Sort type: Sentence length

From all the sentences retrieved, the first step is to find the length of the longest sentence in the set. This is needed to define the bounds that define what goes in the simple, medium, and complex sentence buckets. The bounds for the length of simple and medium sentences are defined below:

$$simple\_upperbound = math.ceil(max\_lengthof sentence/3) + 2$$

$$medium\_upperbound = (simple - 2) * 2$$

These two formulae were derived after several permutations to define upper and lower bounds and ensure an adequate representation of sentences in each bucket. For most other permutations, an overwhelming number of sentences were classified as medium-length rather than simple or complex. Table 7.4 shows the upper bounds for simple and medium-length sentences for sentences of various lengths. Sentences are classified as complex sentences if they are greater than the upper bound for medium sentences.

Figure 7.14 shows the sentences identified as simple, medium, and complex sentences based on the keywords 'volcano', 'form', 'lava', and 'mountain' and using sentence length as the sort type. Please note that the two underlined sentences in each bucket are the sentences that SENCE picked as the two most representative of the keywords among the list of sentences in that bucket.

Table 7.4: Upper bounds for sentence lengths of simple and medium sentences for different length sentences.

Sentence length	Simple sentence upper bound	Medium sentence upper bound	Sentence length	Simple sentence upper bound	Medium sentence upper bound
8	5	6	17	8	12
9	5	6	18	8	12
10	6	8	19	9	14
11	6	8	20	9	14
12	6	8	21	9	14
13	7	10	22	10	16
14	7	10	23	10	16
15	7	10	24	10	16
16	8	12	25	11	18

7.4.2 Sort type: Number of keywords

Since the number of keywords in a sentence will be small, the policy to implement boundaries for identifying simple, medium, and complex sentences based on the number of keywords present in a sentence is straightforward. Medium sentences are expected to have at most twice the number of keywords as those in simple sentences, and sentences having more than the upper bound for medium sentences are classified as complex sentences.

$$bucket\_size = math.floor(num\_kw/3)$$

where num_kw is the number of key words taught in the session

$$simple\_upperbound = bucket\_size$$

$$medium\_upperbound = simple\_upperbound + bucket\_size + 1$$

Keywords: volcano form lava mountain
Filter Used: Sentence Length

SIMPLE SENTENCES

- Stromboli is one of Earth's most active volcanoes.
- Volcanoes begin deep underground.
- Lava, ash, and steam pour from the mountain.
- In an instant, the landscape around the volcano changes.
- A volcano can destroy and entire town.
- They form the crust, our planet's thin outer layer.
- That's a volcano!
- Now it is lava.
- Others cause lava to simply flow out.
- Lava turns to solid rock as it cools.
- There are four main types of volcanoes.
- Shield volcanoes form when lava flows in all directions.
- Some lava domes have been growing for 100 years.

MEDIUM SENTENCES

- Several times an hour, lava shoots out of a volcano in Italy called Stromboli.
- Besides Stromboli, a volcano erupts somewhere on Earth every week.
- Moving tectonic plates can make mountains and volcanos.
- Plates crashing together can buckle and ground to form mountains.
- A volcano can form when a tectonic plate is forced downward into the mantle.
- First, heat and pressure inside Earth melt part of the plate, forming magma.
- A collapsing volcano can form a caldera more than 60 miles wide.
- Each eruption spreads more lava and makes the mountain grow larger.
- Cinder cone volcanoes form when exploding lava hardens into glassy rock fragments.
- Composite volcanoes form mountains with separate layers of lava, ash cinders, blocks, and bombs.

COMPLEX SENTENCES

- The magma that forms most volcanoes comes from just a few miles below Earth's surface.
- When the rising magma escapes through the top of the volcano's vent, it gets a new name.
- Heat inside Earth can create hot springs and bubbling mud pools around the volcano.
- Volcano mountains can form when heat and pressure, miles below Earth's crust, form magma that rises through cracks in the crust.
- A violent explosion can cause a volcano to collapse and form a giant bowl-shaped area called a caldera.
- Volcanoes with lava domes form when lava is too thick and sticky to flow very far.

Figure 7.14: Results of sentences generated with the Sentence Length sort type.

Figure 7.15 shows the sentences identified as simple, medium, and complex sentences based on the keywords 'volcano,' 'form,' 'lava,' and 'mountain' and using 'number of keywords' in the sentence as the sort type. Please note that the two underlined sentences in each bucket are the sentences that SENCE picked as the two most representative of the keywords among the list of sentences in that bucket.

7.4.3 Sort type: Syntactic Dependency of Keyword relative to the Sentence

This sort type looks at where the keyword lies in the structure of a sentence. The sentence is classified as simple, medium, or complex if the keyword is present in the subject, object, or other part of the sentence, respectively. The NLP concept of chunking [13] is used to determine the subject of a sentence. Chunking a sentence means breaking the sentence down into smaller, more parseable components. The chunking process also helps identify the chunk's relationship to the rest of the sentence. For example, packages such as spaCy not only provide tools to chunk a sentence, but they also provide information such as whether the chunk is the subject, object, or root of the sentence.

The following steps are performed for each sentence in the passage:

- The sentence is first converted to a spaCy document.
- Each spaCy document is divided into chunks. spaCy provides a method called 'noun_chunks' [50] for a spaCy tokenized document that provides the noun chunks in a sentence. Noun chunks are "base noun phrases" – flat phrases that have a noun as their head. One can think of noun chunks as a noun plus the words describing the noun. For example, the sentence "'The quick brown fox jumps over the lazy dog' provides 'The quick brown fox' and 'the lazy dog' as noun chunks present in the sentence.

Keywords: volcano form lava mountain
Filter Used: Number of keywords in sentence

SIMPLE SENTENCES

- Stromboli is one of Earth's most active volcanoes.
- Volcanoes begin deep underground.
- In an instant, the landscape around the volcano changes.
- A volcano can destroy and entire town.
- Besides Stromboli, a volcano erupts somewhere on Earth every week.
- That's a volcano!
- When the rising magma escapes through the top of the volcano's vent, it gets a new name.
- Heat inside Earth can create hot springs and bubbling mud pools around the volcano.
- There are four main types of volcanoes.
- They form the crust, our planet's thin outer layer.
- First, heat and pressure inside Earth melt part of the plate, forming magma.
- Now it is lava.
- Others cause lava to simply flow out.
- Lava turns to solid rock as it cools.
- Some lava domes have been growing for 100 years.

MEDIUM SENTENCES

- Several times an hour, lava shoots out of a volcano in Italy called Stromboli.
- Lava, ash, and steam pour from the mountain.
- Each eruption spreads more lava and makes the mountain grow larger.
- The magma that forms most volcanoes comes from just a few miles below Earth's surface.
- A volcano can form when a tectonic plate is forced downward into the mantle.
- A collapsing volcano can form a caldera more than 60 miles wide.
- A violent explosion can cause a volcano to collapse and form a giant bowl-shaped area called a caldera.
- Moving tectonic plates can make mountains and volcanos.
- Plates crashing together can buckle and ground to form mountains.

COMPLEX SENTENCES

- Volcano mountains can form when heat and pressure, miles below Earth's crust, form magma that rises through cracks in the crust.
- Cinder cone volcanoes form when exploding lava hardens into glassy rock fragments.
- Shield volcanoes form when lava flows in all directions.
- Volcanoes with lava domes form when lava is too thick and sticky to flow very far.
- Composite volcanoes form mountains with separate layers of lava, ash cinders, blocks, and bombs.

Figure 7.15: Results of sentences generated with the Number of Keywords in Sentence sort type.

- spaCy, however, does not have a method to retrieve verb chunks similarly. Therefore, the following method was implemented:

```
def return_verb_chunks(sentence_tokenized):
verb_chunks = []
for token in sentence_tokenized:
    if token.pos_ == "VERB" or token.pos_ == "ADV":
        span = sentence_tokenized[token.i:token.i + 1]
        verb_chunks.append(span)
return verb_chunks
```

  In this code block, the method return_verb_chunks accepts a tokenized spaCy sentence. It then checks the part of speech of each token in the sentence. If the part of speech is a verb or adverb, it retrieves the corresponding span from the tokenized sentence. It gathers this list of spans and returns all instances of verbs and adverbs in the sentence.

- The noun and verb chunks together give all the chunks in the sentence.

- SENCE then iterates over each chunk and determines if the lemmatized version of the chunk has a keyword in it.

- If it does, it then classifies it as a simple, medium, or complex sentence depending on the following parameters:

  Simple sentence: If the keyword is in the subject of the sentence, it is identified as a simple sentence.

  Medium sentence: If the keyword is the object of the sentence, the sentence is identified as a medium sentence.

  Complex sentence: All other sentences are classified as complex sentences.

The syntactic dependency or the location of the keyword in the sentence is identified using the 'root.dep_' attribute of the spaCy chunk. The dependency list for spaCy is inherited from ClearNLP [1]. The dependency label for the subject is 'nsubj' (nominal subject), and objects are either 'dobj' (direct object) or 'pobj' (object of a preposition).

Figure 7.16 shows the sentences identified as simple, medium, and complex sentences based on the keywords 'volcano', 'form', 'lava', and 'mountain' and using the 'syntactic dependency of keyword relative to the sentence' sort type. Please note that the two underlined sentences in each bucket are the sentences that SENCE picked as the two most representative of the keywords among the list of sentences in that bucket.

7.4.4 Sort type: Number of Tier 2 and 3 words in sentence

As described in Chapter 6, vocabulary is often differentiated on the basis of the commonality of usage of the word. They are, therefore, broken into tiers of usage or tier words. SENCE, therefore, uses the number of tier 2 and tier 3 words (as described in Chapter 6) in a sentence as a sort type to classify sentences as simple, medium, and complex.

SENCE has a pre-built collection of 2,506 tier words - 1288 tier 2 words and 1,218 tier 3 words.

- Sentences with matching keywords are retrieved from the 'sentences' collection in MongoDB.
- Each sentence is then tokenized, i.e., split into its individual words using spaCy.
- All stop words are removed from this set of words.
- Each token is then lemmatized.

Keywords: volcano form lava mountain
Filter Used: Syntactic dependency of keyword relative to the sentence

SIMPLE SENTENCES

- Several times an hour, lava shoots out of a volcano in Italy called Stromboli.
- Volcanoes begin deep underground.
- The magma that forms most volcanoes comes from just a few miles below Earth's surface.
- Lava turns to solid rock as it cools.
- Shield volcanoes form when lava flows in all directions.
- Volcanoes with lava domes form when lava is too thick and sticky to flow very far.

MEDIUM SENTENCES

- Moving tectonic plates can make mountains and volcanos.
- Others cause lava to simply flow out.
- There are four main types of volcanoes.

COMPLEX SENTENCES

- They form the crust, our planet's thin outer layer.
- Plates crashing together can buckle and ground to form mountains.
- A volcano can form when a tectonic plate is forced downward into the mantle.
- First, heat and pressure inside Earth melt part of the plate, forming magma.
- Now it is lava.
- Volcano mountains can form when heat and pressure, miles below Earth's crust, form magma that rises through cracks in the crust.
- A collapsing volcano can form a caldera more than 60 miles wide.
- A violent explosion can cause a volcano to collapse and form a giant bowl-shaped area called a caldera.
- Composite volcanoes form mountains with separate layers of lava, ash cinders, blocks, and bombs.

Figure 7.16: Results of sentences generated with the Syntactic dependency of keyword relative to the sentence sort type.

- This lemmatized list of words is then used as a match parameter for the tier_words collection in MongoDB, and all matches are returned.
- The instructor is then shown the list of tier 2 and 3 words and asked if they would like to add more tier words to the passage.
- If yes, the additional words are added.
- If new tier words were added, sentences are checked again to determine if the new tier words are present in them.
- SENCE then determines the largest number of tier words in a sentence. It then defines the upper bounds for simple and medium sentences as:

$$simple\_upperbound = math.floor(most\_tier\_words)/3$$

$$medium\_upperbound = simple\_upperbound * 2$$

  Complex sentences are sentences that have more tier words than the medium upper bound.

Figure 7.17 shows the sentences identified as simple, medium, and complex sentences based on the keywords 'volcano', 'form', 'lava', and 'mountain' and using the 'number of tier 2 and tier 3 words in the sentence' sort type. Please note that the two underlined sentences in each bucket are the sentences that SENCE picked as the two most representative of the keywords among the list of sentences in that bucket.

7.4.5 Comparison of sort types

The four sort types mentioned above, sentence length, number of keywords, syntactic dependency of keyword relative to the sentence, and number of tier 2 and tier 3 words in a sentence, are four examples of different types of sorts that can be applied to a set of sentences.

Keywords: volcano form lava mountain
Filter Used: Number of tier 2 and tier 3 words in the sentence

SIMPLE SENTENCES

- They form the crust, our planet's thin outer layer.
- That's a volcano!
- Now it is lava.
- A collapsing volcano can form a caldera more than 60 miles wide.
- Lava turns to solid rock as it cools.
- There are four main types of volcanoes.
- Shield volcanoes form when lava flows in all directions.

MEDIUM SENTENCES

- Several times an hour, lava shoots out of a volcano in Italy called Stromboli.
- Volcanoes begin deep underground.
- Lava, ash, and steam pour from the mountain.
- In an instant, the landscape around the volcano changes.
- A volcano can destroy and entire town.
- Moving tectonic plates can make mountains and volcanos.
- Plates crashing together can buckle and ground to form mountains.
- A volcano can form when a tectonic plate is forced downward into the mantle.
- First, heat and pressure inside Earth melt part of the plate, forming magma.
- Others cause lava to simply flow out.
- Cinder cone volcanoes form when exploding lava hardens into glassy rock fragments.
- Volcanoes with lava domes form when lava is too thick and sticky to flow very far.
- Some lava domes have been growing for 100 years.

COMPLEX SENTENCES

- Stromboli is one of Earth's most active volcanoes.
- Besides Stromboli, a volcano erupts somewhere on Earth every week.
- The magma that forms most volcanoes comes from just a few miles below Earth's surface.
- When the rising magma escapes through the top of the volcano's vent, it gets a new name.
- Heat inside Earth can create hot springs and bubbling mud pools around the volcano.
- Volcano mountains can form when heat and pressure, miles below Earth's crust, form magma that rises through cracks in the crust.
- A violent explosion can cause a volcano to collapse and form a giant bowl-shaped area called a caldera.
- Each eruption spreads more lava and makes the mountain grow larger.
- Composite volcanoes form mountains with separate layers of lava, ash cinders, blocks, and bombs.

Figure 7.17: Results of sentences generated with the number of Tier 2 and Tier 3 words sort type.

They are by no means exhaustive, and other sort types can be identified here, especially as the reading grade of the intended audience increases. Having said that, the sort-type sentence length will be best for the first introduction of the word to the student, as the sentences do not have complex structures and have fewer new words in each sentence. Once the students have been introduced to a new word and can correctly identify the word in sentences based on the sentence length sort, the number of keywords and the number of tier 2 and tier 3 words sorts can be used. These sentences, while longer and more complex than sentences from the first type of sort, still have the keywords appear toward the beginning of the sentence and are, therefore, easier to guess.

The most complex of the four sorts discussed is the sort based on the syntactic dependency of the keyword relative to the sentence. Sentences identified by this sort tend to be longer, and the keywords tend to appear in the middle and end of the sentence.

This is best suited for students who have been through exercises featuring the other sort types, and this evaluates if they can correctly identify keywords irrespective of whether the keyword is the subject, or object, or has another syntactic dependency to the rest of the sentence.

Figure 7.18 shows the top two sentences picked by SENCE for each of the four types of sorts. The color coding shows the same sentences being picked by different sort types for different complexities. For instance, the sentence 'Lava, ash and steam pour from the mountain' with a cell background of light yellow is picked as a Simple sentence for the Sentence Length sort, medium sentence for number of keywords and number of tier 2 and tier 3 words sorts. This is noticed for other sentences picked too. Four sentences appear in at least three of the sorts as either simple, medium, or complex sentences. This is because the sentence similarity method used to pick the top two sentences (explained in Section 7.4.6) for each sort and sentence complexity level (simple, medium, or complex) consistently picks sentences that closely match the string of keywords chosen by the instructor.

Sort criteria	Simple sentences	Medium sentences	Complex Sentences
Sentence length	Lava, ash, and steam pour from the mountain	Cinder cone volcanoes form when exploding lava hardens into glassy rock fragments	Heat inside Earth can create hot springs and bubbling mud pools around the volcano.
	Shield volcanoes form when lava flows in all directions	Composite volcanoes form mountains with separate layers of lava, ash cinders, blocks, and bombs.	Volcano mountains can form when heat and pressure, miles below Earth's crust, form magma that rises through cracks in the crust
Number of keywords	Besides Stromboli, a volcano erupts somewhere on Earth every week.	Lava, ash, and steam pour from the mountain.	Cinder cone volcanoes form when exploding lava hardens into glassy rock fragments.
	Heat inside Earth can create hot springs and bubbling mud pools around the volcano	A violent explosion can cause a volcano to collapse and form a giant bowl-shaped area called a caldera.	Shield volcanoes form when lava flows in all directions.
Syntactic Dependency of Keyword relative to the Sentence	Several times an hour, lava shoots out of a volcano in Italy called Stromboli	Others cause lava to simply flow out.	A violent explosion can cause a volcano to collapse and form a giant bowl-shaped area called a caldera.
	Shield volcanoes form when lava flows in all directions.	There are four main types of volcanoes	Composite volcanoes form mountains with separate layers of lava, ash cinders, blocks, and bombs
Number of tier 2 and 3 words	Now it is lava.	Lava, ash, and steam pour from the mountain	Heat inside Earth can create hot springs and bubbling mud pools around the volcano.
	Shield volcanoes form when lava flows in all directions	Cinder cone volcanoes form when exploding lava hardens into glassy rock fragments.	A violent explosion can cause a volcano to collapse and form a giant bowl-shaped area called a caldera

Figure 7.18: Comparison of sentences chosen by SENCE for the four sort types.

Since these four sentences have most of the keywords chosen for the sort (volcano, form, lava, mountain) for which they were generated, they feature repeatedly among the sort results.

Another result to note is that while the most of the sentences have fully formed contexts within the sentence itself, and do not necessarily need a preceding sentence to give it context, the 'syntactic dependency of keyword relative to the sentence' sort tends to pick sentences that on their own do not provide context and therefore will be hard to fill in the missing words. For example, with the sentence 'Others cause lava to simply flow out', it is hard for the student to understand what is meant by Others. This occurrence of picking sentences with missing context can be attributed to the fact that when SENCE is forced to pick sentences with key words in the object of the sentence (which is the defining criterion for medium complexity sentences for this type of sort), there are not many sentences to choose from. This can be addressed by adding other syntactic dependency criteria to pick medium complexity sentences for this sort type.

7.4.6 Flow of Operations

Figure 7.19 shows the sequence of operations for selecting sentences based on the sort criteria chosen. `storeSentencesForLessonBySort.py` is the Python file that has the main code for this section. There are two other helper files - `SentenceSorterByFilter.py` and `PickSentencesOtherPassages.py` that are called from this file.

SENCE picks sentences and classifies them as simple, medium, and complex sentences in the following sequence of steps:

- The instructor is first asked for the index of the lesson to be taught, sort type, and keywords taught in the lesson. This is shown in Figure 7.20

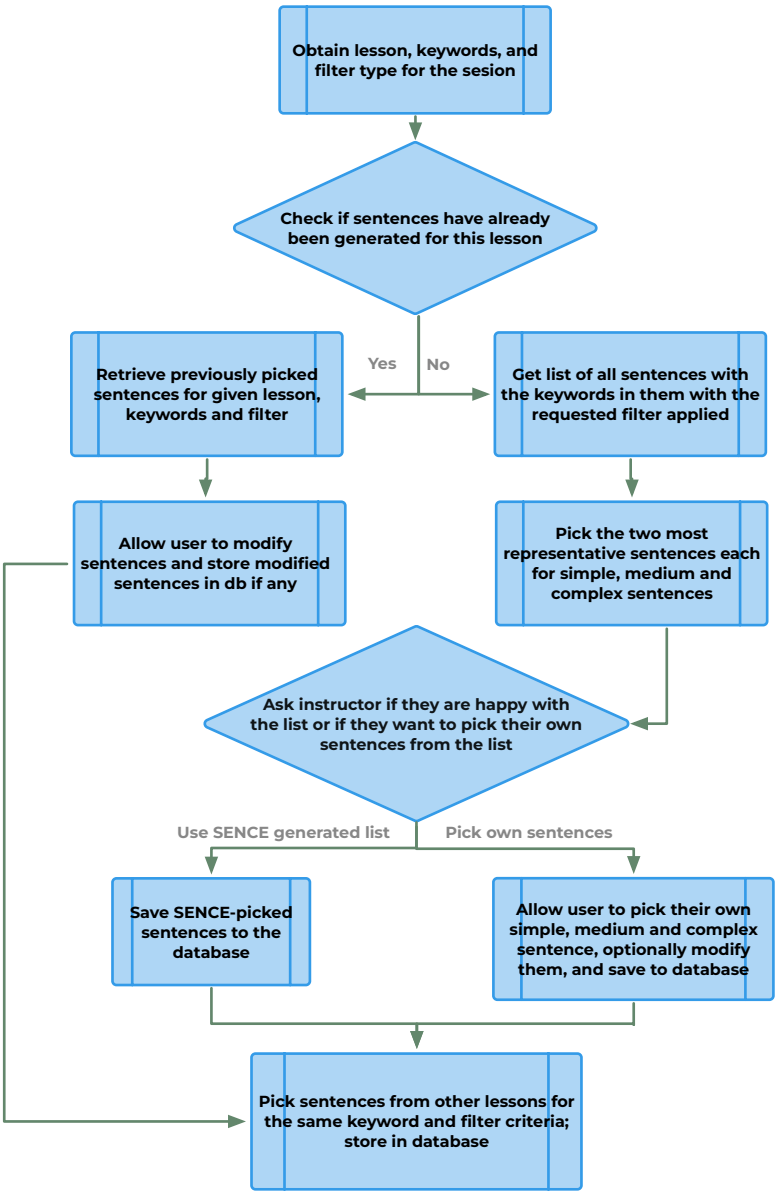

Figure 7.19: Sequence of operations for selecting sentences based on sort criteria.

```
Index number     Title
101      Erupt Chapter 1: Our Fiery Word
102      Erupt Chapter 2: DANGER
103      Water Takes Three Forms
104      The Invention of Paper
105      Louis Armstrong
106      Light and Objects
107      The History of Juneteenth
108      Musical Instruments: Brass Instruments
109      The Milky Way Galaxy
110      American Government - James Madison: A Man with a Plan
111      The Music of the Cherokee
112      What Is a Volcano?
113      The Amazing Water Molecule
114      Jazz
115      Paper Crane: A Symbol of Peace
Enter the Index number of the chapter you want to pick sentences for today: >? 101

System Keywords for this passage are:   volcano form lava plate rock magma earth note fact crust
Others have added the following keywords:   mountain

 Please enter the keywords you want to use in today's class: >? volcano form lava mountain
Please enter the number of the sort you would like applied today.
1. Sentence Length
2. Number of keywords
3. Keyword in structure of sentence
4. Number of Tier 2 and 3 words in sentence
>? 1
```

Figure 7.20: Selecting lesson, keywords, and sort type for sentence generation.

- SENCE then checks to see if sentences have already been generated for this lesson based on the keywords and sort type chosen. This is done by:
  - Checking the *sentences_for_passage* collection if sentences are present for this criterion of lesson, keywords, and sort type choice.
  - If sentences are present, it then goes to the *sentences* collection to select the equivalent sentence text since only ids are stored in the *sentences_for_passage* collection.
  - SENCE then asks the instructor if they are satisfied with the selected sentences or if they would like to modify any of the sentences present.
  - If the instructor would like to modify sentences, modifications are done as per the Section 7.4.6 below, and the modified sentences are saved to the *modified_sentences* collection.
- If sentences have not already been saved for the selected lesson, keywords, and sort type, SENCE performs the following steps:
  - Pick all sentences from the *sentences* collection that satisfy the lesson, keywords, and sort criteria.
  - Check to see if there are modified versions of any of the picked sentences. If there are, substitute the original text of the sentence with the modified text.
  - These sentences are then sorted according to the sort type chosen as per the steps described in Section 7.4.
  - The top two sentences that most closely align with the set of chosen keywords are then chosen for simple, medium, and complex difficulty questions. This down-select of the number of sentences is done using the 'similarity' [51] method in spaCy. The list of keywords is converted to a spaCy tokenized document. Each sentence is also converted into a

spaCy tokenized document. SENCE then uses the 'similarity' method of spaCy to find the similarity between the two tokenized documents. First, the sentence to be compared and the string of keywords are converted to vectors. These vectors capture the semantic meaning of words based on their context in large corpora. Words with similar meanings have vectors that are close to each other in this space. SpaCy provides pre-trained word vectors for several languages. The vectors are trained on large datasets and capture semantic relationships between words. spaCy performs word and sentence vectorization as follows: First, the text is tokenized into individual words or tokens. Next, each token is mapped to its corresponding vector from spaCy's pre-trained vector table. For phrases, sentences, or documents, the vectors of individual tokens are averaged to create a single vector representation. Once spaCy has the vector forms of the keyword string and the sentence to compare, the similarity between the two is calculated using the cosine similarity of their vector representations. Cosine similarity measures the cosine of the angle between two vectors, providing a value between -1 and 1. The higher the similarity score, the closer the sentence is to the list of keywords chosen. The sentences with the top two scores are chosen for simple, medium, and complex difficulty sentences. An example of how spaCy similarity works is shown below:

$$compare\_term = nlp(\text{“lava mountain form flow”})$$

$$sentence\_1 = nlp(\text{“Lava flows from the mountain.”})$$

$$sentence\_2 = nlp(\text{“This is a beautiful cat”})$$

$compare\_term.similarity(sent\_1)$ returns 0.14297191358389474

$compare\_term.similarity(sent\_2)$ returns 0.0713364052717547

- The instructor is then shown this list of six sentences (two each for simple, medium, and complex sentences) and asked if they would like to save these sentences or modify them. If the instructor chooses the latter, the steps in Section 7.4.6 are followed.
- Sentences are then saved to the *sentences_for_passage* collection.
- Sentences from other lessons based on the keywords and sort choice selected are picked as described in Section 7.4.6.

Modifying sentences:

The instructor is shown the list of sentences generated and asked if they would like to modify any of the sentences. If the instructor indicates that they would, they are asked to supply the sentence ids of the sentences they would like to modify. For each sentence to be modified, SENCE performs the following steps:

- Check the *modified_sentences* collection to see if the sentence has already been modified by the current instructor. If it has, verify with the instructor that they would like to modify it again. If the instructor says yes, SENCE asks for an alternate sentence version. It then uses the 'set' [34] attribute of MongoDB to ensure that only a single instance of {*sentence_length*, *modified_sentence*, *modified_by*} is stored in the database at a time.
- If the sentence has not been modified previously, the instructor is asked for an alternate version of the sentence, and this new sentence is added to the *modified_sentences* collection.

Picking Sentences from other corpora:

Sentences are picked from other corpora using a method that is very similar to picking sentences from the lesson described above. SENCE first picks sentences from lessons that are **not** the lesson being taught and then sorts them into simple, medium, and

complex buckets based on the sort criteria chosen. The methodology here is the same as that described in Section 7.4. Finally, **one** sentence that is most representative of the keywords is chosen using spaCy's similarity measure as described above.

Ensuring there are enough sentences in each bucket

An issue that was encountered on an occasional basis was that there was an uneven spread of sentences between the simple, medium, and complex sentence buckets. Logic needed to be added to ensure that the upper bounds for simple and medium sentences could be adjusted on the fly so that there would be sentences provided for all three buckets. There was also logic needed to make sure that SENCE was not stuck in an endless loop when there were simply no sentences present to fulfill the criteria.

7.5 Storing sentences based on search criteria

Once SENCE has presented the final list of sentences, and the instructor accepts the results, the next step is to store these sentences in MongoDB. These final sentences from the passage and other corpora are stored in the *sentences_for_passage* collection. The following fields are stored:

- ***id:*** A unique value for every entry.
- ***username***: The name of the instructor curating this sentence list.
- ***passage_num***: The id of the passage chosen for the lesson and sort type.
- ***keywords:*** The keywords used for the passage and sort type.
- ***sort***: The type of sort used.
- ***sentence_type***: This has a value of 'system' or 'user' and indicates if the sentence is unmodified (system) or modified by any user (user).

- ***sentence_id***: The id of the sentence as it appears in the 'sentence' or 'modified_sentence' collection, depending on the value of 'sentence_type.'

- ***difficulty_level***: This is one of three values - simple, medium, or complex, depending on the sort type used for the lesson.

- ***source:*** This has one of two values - passage or other. Passage indicates that the sentence came from the passage being taught; other indicates that the sentence is from other corpora.

- ***sentence_with_spaces***: As a preparatory step for the next phase of SENCE, i.e., assessing the student's vocabulary comprehension, each sentence identified for assessment is stored in this collection with dashes replacing the keywords used in the lesson. For example, for the sentence 'Cinder cone volcanoes form when exploding lava hardens into glassy rock fragments.' the *sentence_with_spaces* field would hold 'Cinder cone ———— form when exploding ———— hardens into glassy rock fragments.'

7.5.1 Identifying and replacing keywords in a sentence with dashes

To replace keywords with dashes, SENCE first needs to identify where the keywords are located in the sentence. spaCy's Matcher [49] is used for this. Matcher matches sequences of tokens based on pattern rules. Figure 7.22 shows the code in SENCE that accomplishes this. First, all the lemmatized versions of the keywords are added to a Matcher instance. Then the matcher method is run on the spaCy tokenized sentence. If there are any matches present, Matcher provides three attributes for each match - the id of the match, the token position for the start of the match, and the token position for the end of the match. For example, passing the kw_list argument with a single entry of 'volcano' and the sentence 'Cinder cone volcanoes form when exploding lava hardens into glassy rock fragments.' to the code in Figure 7.22 produces a

```
match_id of 15733814258258881866, start and end indices of 2 and 3,
and the text span 'volcanoes'.
```

It must be noted here that Matcher only provides case-sensitive matches, and therefore, the sentence and keywords must first be converted to the same case before being used. Once SENCE knows the start and end indices of all the keywords in the sentence, it replaces tokens in those indices with a dash instead. This is then converted to a text form from the spaCy tokenized form and stored in the database. This code is shown in Figure 7.23.

7.6 Assessing students' vocabulary comprehension

When the instructor has confirmed that there are keywords, sentences, and a filtered list of sentences based on the sort of choice generated for the lesson, the instructor is ready to administer the assessment to the student. The instructor is first presented with a list of lessons, keywords, and sort types available for assessments. This is shown in Figure 7.21.

```
  Index number  Title                                    Keywords                       Sort type
--------------  ---------------------------------------  -----------------------------  ----------------------------------------
             0  Water Takes Three Forms                  shape gas water vapor          Number of Tier 2 and 3 words in sentence
             1  Erupt Chapter 1: Our Fiery Word          form volcano lava mountain     Sentence Length
             2  Bill Nye the Science Guy: Photosynthesis breathe sugar plant animal     Number of keywords
             3  The Amazing Water Molecule               earth hydrogen exist gas water Number of keywords
             4  The Tale of Peter Rabbit                 gate little rabbit time        Sentence Length
             5  Erupt Chapter 2: DANGER                  volcano ash lava flow          Number of keywords
             6  Water Takes Three Forms                  shape water gas form           Keyword in structure of sentence
             7  Bill Nye the Science Guy: Photosynthesis sugar animal breathe plant     Keyword in structure of sentence
Enter the ID number (first column) of the chapter you want to run the assessment for today.
 Please note that if you do not find the chapter you would like to use, you might need to run sentence generation for them first
 before this step:
```

Figure 7.21: List of lessons with associated keywords and sorts available for assessment.

```
    # this method accepts a sentence and a list of keywords for the lesson and identifies
    # which keywords are present in the sentence
    4 usages
    def get_keywords_in_sentence(self, sentence, kw_list):
        # kw_list = [AssessSentences.keywords_hash[k_id] for k_id in kw_ids]
        match_list = []
        doc = self.nlp(sentence)

        matcher = Matcher(self.nlp.vocab)
        for item in kw_list:
            matcher.add( key: "matched kw",  patterns: [[{'LEMMA': item}]])
        matches = matcher(doc)
        for match_id, start, end in matches:
            span = doc[start:end]
            # store the matched keyword, and the starting and ending token index
            match_list.append({'keyword': span, 'start': start, 'end': end})
        return match_list
```

Figure 7.22: Identifying keywords in a sentence using spaCy Matcher.

```
for kw_item in matched_keywords:
    new_sent = sent[0: kw_item['start']].text + sent[
        kw_item['start'] - 1].whitespace_  # If tokens have been
    # skipped, add them in (with trailing whitespace if available)
    new_sent += sentence_blank + sent[kw_item['start']].whitespace_  # Replace token, with trailing
    # whitespace if available
    new_sent += sent[kw_item['end']:].text  # add remainder of the sentence
    sent = self.nlp(
        new_sent)  # for more than one keyword in the sentence, regenerate the tokenized form of
    # the sentence to repeat this process
```

Figure 7.23: Code that replaces tokens with dashes.

Once the lesson, keywords, and sort type have been picked for the assessment, the student will execute the next section. The student is presented with a list of increasingly difficult sentences sorted by the simple, medium, and complex sentences of the sort type chosen. The student is also shown the list of words (keywords) that are to be chosen to replace the dashes in the sentence. This is shown in Figure 7.24.

There are checks in place to ensure that the student enters the exact number of words to replace the dashes, and there are helpful error messages provided to rectify this. This is done using the inputCustom method of PyInputPlus. This method accepts a custom validation instead of the default validation it provides, such as checking if the input is a string, integer, date, or choice in a menu item. Once all the questions have been answered, SENCE then compares the given input to what is expected and stores a value of correct or wrong for the response.

Determining correctness of response

SENCE performs the following steps to evaluate if the response is correct or not.

- It first replaces the response vocabulary instead of dashes in the sentence.
- It then checks if the sentence is a match to the original sentence. If it is, it marks the response as correct.
- If there isn't a direct match, SENCE then goes through each vocabulary response and compares it to the original keyword expected. This comparison

```
Use one of the following words for each blank provided:  earth, hydrogen, exist, gas, water
Liquid ----------- is a jumbled bunch of water molecules.water
As ice forms, ----------- molecules arrange themselves neatly in a crystal structure.gas
----------- molecules cling to each other because of a force called ----------- bonding.water hidrogen
When ----------- molecules escape from liquid water and float into the air, they turn into an invisible
 ----------- called water vapor.water gas
It ----------- in three states on -----------: liquid, -----------, and solid:exists earth gas
Please enter your (student's) name: Mary
Results have been saved.
```

Figure 7.24: Student performs the assessment.

is done using NLTK's edit distance method [2]. The edit distance method calculates the Levenshtein edit distance between two strings. The edit distance is the number of characters that need to be substituted, inserted, or deleted to transform string1 into string2. If the edit distance is two or less, the correct keyword replaces the vocabulary response. This is done to reduce responses being marked as incorrect due to misspellings. An example of this is shown in Figure 7.26. The student responds with the word 'hidrogen' instead of hydrogen. SENCE auto-corrects the word and evaluates it as a correct response.

- Once the keyword comparison is done, and the vocabulary response is possibly updated, the sentence is checked again against the original sentence. If the sentences match, the response is marked as correct. Otherwise, the response is marked as incorrect.

7.6.1 Displaying results

The final component of SENCE is displaying student responses to the instructor. The instructor is presented with a list of lessons, keywords, and sort assessments that have been completed. This is shown in Figure 7.25.

Once the instructor picks the assessment to review, the assessment criteria are shown, followed by the original sentence, student response, and SENCE's evaluation of whether the response was correct or not. This is shown in Figure 7.26.

```
   Index   Passage title              Keywords                          Filter type                     Student name    Time of assessment
   Number
---------  -------------------------  --------------------------------  ------------------------------  --------------  -------------------
       1  Erupt Chapter 2: DANGER    volcano, ash, lava, flow          Number of keywords              tsamuel         2025-01-27 12:50:32
       2  The Amazing Water Molecule earth, hydrogen, exist, gas, water Number of keywords              Tim             2025-01-30 14:24:30
       3  The Amazing Water Molecule earth, hydrogen, exist, gas, water Number of keywords              Mary            2025-02-04 09:42:04
       4  Water Takes Three Forms    shape, water, gas, form           Keyword in structure of sentence tsamuel        2025-01-27 17:07:17
Please enter the index number of the results you will like to see: 2
```

Figure 7.25: Instructor shown available assessments for review.

```
Filter criteria used:
Passage:  The Amazing Water Molecule
Keywords:  earth, hydrogen, exist, gas, water
Student name:  Tim
Time of assessment:  2025-01-30 14:24:30

Original sentence:  Liquid water is a jumbled bunch of water molecules.
Student response:  Liquid water is a jumbled bunch of water molecules.
Student response is  CORRECT.

Original sentence:  As ice forms, water molecules arrange themselves neatly in a crystal structure.
Student response:  As ice forms, gas molecules arrange themselves neatly in a crystal structure.
Student response is  WRONG.

Original sentence:  Water molecules cling to each other because of a force called hydrogen bonding.
Student response:  water molecules cling to each other because of a force called hidrogen bonding.
Student response is  CORRECT.

Original sentence:  When water molecules escape from liquid water and float into the air, they
 turn into an invisible gas called water vapor.
Student response:  When water molecules escape from liquid water and float into the air, they turn
  into an invisible gas called water vapor.
Student response is  CORRECT.

Original sentence:  It exists in three states on Earth: liquid, gas, and solid:
Student response:  It exists in three states on earth: liquid, gas, and solid:
Student response is  CORRECT.

 Student answered 4 of 5 sentences correctly.
The percentage of sentences answered correctly is: 80.00%
```

Figure 7.26: Sentence evaluations shown to the instructor.

7.7 Evaluation of keyword and sentence extraction from different media

One of the research questions this dissertation attempts to answer is whether all types of media, such as text from textbooks or online lessons, storybooks, and transcripts from television shows, can be treated the same or do they require different levels of pre- and text-processing. In this section, results with texts from storybooks and transcripts from television shows will be discussed.

7.7.1 Results with Storybook texts

Two storybook transcripts were retrieved from Project Gutenberg [40]. Project Gutenberg is an online library of free eBooks whose mission is to encourage the creation and distribution of eBooks.

The two texts chosen for analysis with SENCE were The Tales of Peter Rabbit [42] and Chapter 7 of The Princess and the Goblin [41]. These texts are available in the public domain in the US.

Pre-processing: The pre-processing for both texts was more complicated than that of texts from textbooks and online lessons. For example, since the Tales of Peter Rabbit is a heavily illustrated book, as most children's literature is. The text-only version of the transcript had several mentions of [ILLUSTRATION] to show where an illustration would be present. In order to ensure that [ILLUSTRATION] would not be picked up as a keyword by SENCE, instances of this word had to be removed prior to ingestion by SENCE. Both texts also had multiple instances of extraneous spaces and newlines, and these had to be cleaned prior to ingestion as well. Raw text from the Peter Rabbit transcript showing these issues is available in the Appendix at Appendix A.1. A final observation in the pre-ingestion stage is that even though only short stories (in the case of Peter Rabbit) and one chapter (in the case of Princess

and the Goblin) were used, the text is much longer in length compared to a similar grade lesson material.

SENCE generated keywords for Storybooks

Peter Rabbit: peter, mcgregor, Mr, run, little, time, rabbit, garden, gate, jacket

Princess and the Goblin: night, goblin, work, curdie, father, away, miner, ore, mountain, great

Keyword generation: The box above shows the top ten keywords generated by SENCE for these passages. From both passages, we can see the presence of names of individuals (peter, mcgregor, curdie) present in the top ten keywords selected. This could indicate that proper nouns should not be included in keyword selection for a given passage.

Retrieving sentences based on sort and keywords: The following are the top two simple, medium, and complex sentences generated from the Peter Rabbit text, using the keywords: little, time, rabbit, and gate, and using 'Sentence Length' as the sort type.

Simple Sentences:

- One table-spoonful to be taken at bed-time.
- Mr. McGregor was after him in no time.

Medium sentences:

- Mr. McGregor hung up the little jacket and the shoes for a scare-crow to frighten the blackbirds.
- He found a door in a wall; but it was locked, and there was no room for a fat little rabbit to squeeze underneath.

Complex sentences:

- Now my dears,' said old Mrs. Rabbit one morning, 'you may go into the fields or down the lane, but don't go into Mr. McGregor's garden: your Father had an

accident there; he was put in a pie by Mrs. McGregor.'

- Flopsy, Mopsy, and Cottontail, who were good little bunnies, went down the lane to gather blackberries: But Peter, who was very naughty, ran straight away to Mr. McGregor's garden, and squeezed under the gate!

As expected for a storybook text, the sentence lengths for medium and complex sentences are much longer than those of lesson-based texts. Another interesting point of note is that sentences from storybooks have fewer sentences with multiple keywords in them than lesson-based texts. This is seen by using the 'Number of keywords in sentence' sort for the same passage. Using the same set of keywords as above, there are two sentences generated for simple sentences and none for medium or complex sentences. Sentences generated for simple sentences using these keywords and sort type are:

- Mr. McGregor hung up the little jacket and the shoes for a scarecrow to frighten the blackbirds.

- Once upon a time there were four little Rabbits, and their names were– Flopsy, Mopsy, Cotton-tail, and Peter.

7.7.2 Television shows

Two television show transcripts were retrieved for the purposes of this dissertation work. They were retrieved from eMediaVA. eMediaVA is Virginia's premier digital media content library for educators and students, providing access to thousands of free, relevant, standards-of-learning-aligned digital learning resources for classrooms [15]. The two transcripts chosen for analysis with SENCE were an episode of Bill Nye the Science Guy [14] and an episode of Continue to Know with WHRO [16]. The raw texts for these transcripts are available at Appendix A.3 and Appendix A.2.

Pre-processing: The pre-processing of text prior to ingestion by SENCE was by far the most cumbersome of the three media types evaluated. Television shows tend to be extremely conversational in nature, and therefore, a lot of data cleaning needs

to be done before it can be ingested. The transcripts also contain meta information such as the name of the speaker before the transcript of what they say, and this type of information is of little value while teaching children vocabulary. An attempt was made to ingest a non-lesson-based television show - an episode of Daniel Tiger. It quickly became apparent that it would be difficult to retrieve functional keywords from its transcript. The raw text for this transcript is available at Appendix A.4. Therefore, only lesson-based television show transcripts were chosen to test this type of media.

One of the issues specific to pre-processing transcripts of television shows was the presence of special notations, such as musical notes. This is seen in the Bill Nye the Science Guy: Photosynthesis episode Appendix A.3 transcript. It is also necessary to parse functional text from conversational banter. Finally, even a ten-minute third or fourth-grade television show transcript has significantly more text than a typical lesson for the same grade level.

SENCE generated keywords for television show transcripts

Bill Nye the Science Guy - Photosynthesis episode: plant, carbon, dioxide, food, different, sugar, oxygen, chemical, animal, breathe

Continue to Know with WHRO: Rocks! episode: rock, form, solid, change, igneous, sedimentary, surface, earth, pressure, heat

Keyword generation: As seen in the results in the box above, since the television shows chosen were lesson-based, there are far more practical words that can be used for vocabulary comprehension with SENCE than those generated from storybooks. A direction for future work based on the keywords generated for the Bill Nye the Science Guy passage is to enhance keyword generation to include multi-word keywords. For example, carbon and dioxide are identified as two separate keywords in this passage, when it would be more practical to have them appear as a single keyterm.

Retrieving sentences based on sort and keywords: The following are the top two simple, medium, and complex sentences generated from the Bill Nye the Science Guy: Photosynthesis transcript, using the keywords: plant, sugar, animal, breathe and using 'Keyword in structure of sentence' as the sort type.

Simple Sentences:

- Plants take in carbon dioxide from the air and give off oxygen.
- It's the same blue indicator liquid, and it stays blue because these plants are absorbing carbon dioxide and giving off oxygen.

Medium sentences:

- For example, here's an apple, and apples are sweet, but it's a different flavor from, say, sugar from a sugar cane or orange juice from an orange or maple syrup from a maple tree.
- Different types of plants live all over the world in different places, and plants keep the whole world alive.

Complex sentences:

- Mm, plant's sugar, it is a good deal.
- When plants make food, they put the oxygen that we breathe into the air.

As with storybook texts, sentences chosen from television transcripts tend to be longer. They are also more conversational in nature. Since the topic of the transcript, however, is lesson-based, there are more sentences available when it comes to more than one keyword in the sentence. Using the same set of keywords as above, and the ''Number of keywords in sentence' sort, there are more sentences generated than with storybook transcripts. However, SENCE could only still retrieve one sentence of complex complexity for this type of sort. The SENCE retrieved sentences for these keywords, and the sort type are below:

Simple Sentences:

- Plants take in carbon dioxide from the air and give off oxygen.
- It's the same blue indicator liquid, and it stays blue because these plants are absorbing carbon dioxide and giving off oxygen.

Medium sentences:

- Mm, plant's sugar, it is a good deal.
- When plants make food, they put the oxygen that we breathe into the air.

Complex sentences:

- Now, plants take in carbon dioxide given off by animals and fungus and some other plants and make oxygen, which animals like us breathe.

Chapter 8

Conclusions and Broader Impacts

The following are the significant takeaways from this research:

Assessment of NLP off-the-shelf tools: The NLP ecosystem is flush with many NLP libraries for basic and complex NLP tasks. It is tricky to determine which libraries are best suited for different tasks. Each also incurs its own compute and storage costs. While many libraries might do the same task, not all produce the same results. For example, for the sentence 'Several times an hour, lava shoots out of a volcano in Italy called Stromboli.' spaCy identified the word 'shoots' as a noun. In contrast, Stanza correctly identified the part of speech as a verb.

As discussed in Section 7.2.1, different key term extraction algorithms produce significantly different results. Some allow for fine-tuning and filtering, while others do not. It is essential to correctly identify which package is a good fit for each NLP problem on a case-by-case basis.

Another point of note is that comprehensive packages like spaCy and NLTK can do the vast majority of most basic NLP tasks, such as word and sentence tokenization, collect basic sentence metrics such as length of sentence, and find parts-of-speech and lemma versions of words. A researcher does not have to deploy a plethora of tools to do simple NLP tasks; they will be able to do most of what they need to do with just one or two.

Finally, it was also noticed that sometimes packages treat the same process differently between multiple methods in the same package. Taking the example of the word tokenizer and Matcher [49] tools in spaCy, it was found that the spaCy word tokenizer counted 'Mr.' as one token. In contrast, spaCy's Matcher treated it as two. This can lead to unexpected results if unaccounted for.

Comparing keyword extraction algorithms: While there are several off-the-shelf keyword / keyterm extraction algorithms available, not all of them work exactly alike. Before they can be used, they all need pre-processing steps, such as removing stop words and lemmatizing. Additionally, packages such as Rake and Textacy generate long phrases as key terms instead of single and composite words. BERT leans toward selecting more nouns and adjectives at the cost of verbs. spaCy (which is the underlying keyword extractor in SENCE) and TextRank perform similarly. However, it was noted that SENCE performed better at identifying different parts of speech (such as nouns, adjectives, adverbs, and verbs) as compared to TextRank when it comes to non-science-based lessons (such as lessons on arts or social studies).

Comparing sentence sort types: There were four different sort types for sentence sets discussed in this dissertation: sentence length, number of keywords, syntactic dependency of keyword relative to the sentence, and the number of tier 2 and tier 3 words in a sentence. They are by no means exhaustive, and other sort types can be identified here, especially as the reading grade of the intended audience increases. The simplest sort type that can be introduced just after a new vocabulary word is introduced to a student is the sort type based on sentence length. This is because the keywords tend to appear at the beginning of the sentence, and there are few new words in each sentence compared to the other sort types. The intermediate level sort types are sorts based on the number of keywords, and the number of tier 2 and tier 3 words in a sentence. Sentences here tend to be longer, but keywords still appear at the beginning of the sentence and are therefore easier for students to get right. The most complex type of sort is the sort based on the syntactic dependency

of the keyword relative to the sentence. This is because the keywords tend to have different and more complex syntactic dependencies from the rest of the sentence.

NoSQL databases are a different approach to data storage than traditional relational databases: NoSQL databases such as MongoDB have a different approach to data storage than traditional SQL databases such as PostgreSQL or MySQL. The traditional approaches of foreign keys and joins do not have 1-1 equivalents in MongoDB. NoSQL databases are primarily used for text storage and are a convenient way to store information in a format similar to JSON files. It is cumbersome to implement relational database structures in a NoSQL database. Since NoSQL databases are primarily used for text storage, they also do not lend well to heavy in-database analysis. Data is expected to first be retrieved and then analyzed within the confines of software code. Using an example of counting the number of sentences in the sentences collection grouped by passage, the SQL code of

```
select count(1) from sentences group by passage_id
order by passage_id ASC;
```

translates to

```
db.sentences.aggregate(
{$group : { _id :  "$passage_id", count : {$sum : 1}}}
).sort({"_id":1})
```

in MongoDB.

Volume of text needed for reasonable sentence retrieval: Through multiple iterations of running Sentence Retrieval based on sort types in SENCE, it was evident that there needs to be a base minimum of keywords specified to retrieve enough sentences to fill the simple, medium, and complex sentence buckets adequately. For example, if there were only two keywords specified, SENCE would not find many sentences from the lesson being taught since there might not be many sentences in the passages that feature only one of two keywords. This reduces the quality of sentences

that are finally chosen for student assessment. In the same light, there also needs to be a maximum number of keywords per sort type chosen. Otherwise, the risk is that there are too many blanks in the sentence, making it unintelligible for the student to get right.

Storybooks are not ideal candidates for retrieving sentences based on keywords: It was interesting to note that while storybooks have much longer text lengths (number of sentences) than the average lesson-based text, the number of sentences retrieved for the top ten keywords was still relatively small. This is because there are fewer highly repeated words in conversational language, such as those available in storybooks and television shows, than there are in lesson-based texts.

8.1 Broader Impacts

There are three significant, broader impacts of this work. They are discussed below.

Provides a straightforward, user-friendly interface to an NLP application: This tool is primarily designed for instructors without a background in natural language processing or experience in using NLP libraries. It provides automation powered by NLP to everyday classroom tasks through an easily understood and navigable interface. SENCE uses human language prompts to guide instructors through the process of keyword generation and sentence extraction. Similarly, it uses friendly and easy-to-understand language to help students complete the assessments.

Versatility of usage: Unlike many artificial intelligence solutions that require stable and often high-speed internet connections, SENCE can operate without either. Since the models and libraries used in SENCE are pre-downloaded, SENCE can operate in areas with unreliable or no internet. This has broad applications for vocabulary training in rural and remote areas.

Open to community contributions: Since SENCE is entirely built using open-source software, it is freely available for the education and NLP community to improve. Improvements for advancing SENCE are discussed in the following chapter.

Chapter 9

Future Work

This chapter discusses directions for future work in SENCE. This is discussed from two perspectives - technology and user experience.

Large Language Models: Since the target student population for this research was children between the 3rd and 5th grade, the usage of large language models (LLMs) [3] in SENCE was deemed to be out-of-scope. However, for later grades and advanced reading levels, LLMs can prove to be a critical improvement. While SENCE currently only has access to sentences from passages directly supplied to it, LLMs can provide richer corpora for sentence retrieval. This is because LLMs come pre-trained on large amounts of data, such as books, articles, and websites.

Additionally, Retrieval-Augmented-Generation (RAG) [4] would be a valuable addition to SENCE. RAGs can be used for testing students' vocabulary comprehension based on synonyms and associated vocabulary usage. For instance, in a lesson based on weather, for keywords rain, clouds, and thunder, a RAG can also present vocabulary such as humidity, forecast, and tornado, irrespective of whether they appear in the lesson being taught.

Scaling SENCE: The current purpose of SENCE is to serve as a proof-of-concept for educators and NLP practitioners on what is possible with off-the-shelf open-source and lightweight AI tools. It is not currently implemented for large-scale

deployment. Web development work, such as creating a more engaging graphical user interface, developing web session hooks for retrieving user information (such as username, timestamps, etc), and scaling for multiple instructor-student-lesson sessions, is in the scope of future work.

Multi-word key terms: SENCE currently only identifies single words as keywords for a passage. The next iteration of SENCE should include multi-word key terms as well. For example, the word carbon dioxide would be frequently used in a passage about gases. SENCE currently identifies carbon and dioxide as two different keywords. In a future iteration, it will be effective to identify carbon dioxide as a single keyterm.

Survey comparing human and SENCE choices for keyword generation and sentence retrieval: A survey to compare human and SENCE choices for keyword generation and sentence retrieval is currently waiting for Institutional Review Board (IRB) review. The survey is in collaboration with Dr. McCarthy and the UT Augmentative and Alternative Communication, Language, and Literacy (UT-AACL) laboratory. This survey will examine how practicing teachers, general education and special education, practicing school-based Speech-Language Pathologists (SLPs), pre-service teachers, and pre-service SLPs select vocabulary they might work on with students. It also compares how they might classify sentences as simple, medium, and complex based on the four sort types of SENCE and compares that with the results that SENCE produces. The survey is ready to be distributed and has been set up using Qualtrics. Keyword generation will compare human choices with results from SENCE, YAKE, Rake, Bert, Textacy, SgRank, and TextRank. Results from this survey will be published at both Education and NLP-centric conferences, such as The 2025 Conference on Empirical Methods in Natural Language Processing [17] or the 20th Workshop on Innovative Use of NLP for Building Educational Applications [47].

Bibliography

[1] Center for Language and Information Research (2024). ClearNLP Guidelines. Accessed: 2025-01-15. 52

[2] 2024, NLTK Project. (2024). nltk.metrics.distance module. Accessed: 2025-01-15. 69

[3] Amazon Web Services, Inc (2024a). What is LLM (Large Language Model)? Accessed: 2025-01-15. 81

[4] Amazon Web Services, Inc (2024b). What is RAG (Retrieval-Augmented Generation)? Accessed: 2025-01-15. 81

[5] Bird, S., Klein, E., and Loper, E. (2009). *Natural language processing with Python: analyzing text with the natural language toolkit.* " O'Reilly Media, Inc.". 2, 4, 7

[6] Bishop, D. V. (2006). What causes specific language impairment in children? *Current directions in psychological science*, 15(5):217–221. 5

[7] Burnaby School District Blogs (2024). Grades K-12 Tier 2 vocabulary list. Accessed: 2025-01-15. 21

[8] Butcher, P. G. and Jordan, S. E. (2010). A comparison of human and computer marking of short free-text student responses. *Computers & Education*, 55(2):489–499. 7

[9] Campos, R., Mangaravite, V., Pasquali, A., Jorge, A., Nunes, C., and Jatowt, A. (2020). Yake! keyword extraction from single documents using multiple local features. *Information Sciences*, 509:257–289. 4

[10] Danesh, S., Sumner, T., and Martin, J. H. (2015). Sgrank: Combining statistical and graphical methods to improve the state of the art in unsupervised keyphrase extraction. In *Proceedings of the fourth joint conference on lexical and computational semantics*, pages 117–126. 8

[11] Department of Audiology and Speech Pathology, University of Tennessee Health Science Center (2025). Jillian H. McCarthy Faculty page. Accessed: 2025-01-15. 3

[12] Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. 4, 8

[13] Dudhabaware, R. S. and Madankar, M. S. (2014). Review on natural language processing tasks for text documents. In *2014 IEEE International Conference on Computational Intelligence and Computing Research*, pages 1–5. 49

[14] eMediaVA (2020). Transcript of bill nye the science guy - photosynthesis episode. 73

[15] eMediaVA.org (2025a). emediava. 73

[16] eMediaVA.org (2025b). "rocks! (5th grade). 73

[17] emnlp (2024). The 2025 Conference on Empirical Methods in Natural Language Processing . Accessed: 2025-01-15. 82

[18] Gertner, B. L., Rice, M. L., and Hadley, P. A. (1994). Influence of communicative competence on peer preferences in a preschool classroom. *Journal of Speech, Language, and Hearing Research*, 37(4):913–923. 6

[19] Gray, S. (2005). Word learning by preschoolers with specific language impairment. 6

[20] Hall, N. E. (1997). Developmental language disorders. In *Seminars in Pediatric Neurology*, volume 4, pages 77–85. WB SAUNDERS COMPANY. 1

[21] Hecker, O., McCarthy Maeder, J., Berry, M., and Schwarz, I. (2018). Effectiveness of a speech-to-text vocabulary application for children who are hearing impaired. In *American Speech Language Hearing Association Annual Convention*. American Speech Language Hearing Association. 2

[22] Honnibal, M. and Montani, I. (2017). spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. *To appear*, 7(1):411–420. 2, 4, 18

[23] Horev, R. (2018). Bert explained: State of the art language model for nlp. *Towards Data Science*, 10. 8

[24] Intelligent Assessment Technologies Limited (2024). Intelligent Assessment Technologies. Accessed: 2025-01-15. 7

[25] Jawahar, G., Sagot, B., and Seddah, D. (2019). What does bert learn about the structure of language? In *ACL 2019-57th Annual Meeting of the Association for Computational Linguistics*. 8

[26] Jordan, S. and Mitchell, T. (2009). e-assessment for learning? the potential of short-answer free-text questions with tailored feedback. *British journal of educational technology*, 40(2):371–385. 7

[27] Klein, D. and Manning, C. D. (2002). Fast exact inference with a factored model for natural language parsing. *Advances in neural information processing systems*, 15. 2, 4

[28] Manning, C. D., Surdeanu, M., Bauer, J., Finkel, J. R., Bethard, S., and McClosky, D. (2014). The stanford corenlp natural language processing toolkit. In *Proceedings of 52nd annual meeting of the association for computational linguistics: system demonstrations*, pages 55–60. 9

[29] McCarthy, D. (2009). Word sense disambiguation: An overview. *Language and Linguistics compass*, 3(2):537–558. 7

[30] Mihalcea, R. and Tarau, P. (2004). Textrank: Bringing order into text. In *Proceedings of the 2004 conference on empirical methods in natural language processing*, pages 404–411. 4

[31] Miller, G. A. (1995). Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41. 7

[32] Min H. Kao Department of Electrical Engineering and Computer Science, University of Tennessee Knoxville (2025). Michael Berry Faculty page. Accessed: 2025-01-15. 3

[33] MongoDB, Inc. (2024a). MongoDB. Accessed: 2025-01-15. 17

[34] MongoDB, Inc. (2024b). MongoDB $set. Accessed: 2025-01-15. 32, 63

[35] Morgan, P. L., Farkas, G., and Wu, Q. (2011). Kindergarten children's growth trajectories in reading and mathematics: Who falls increasingly behind? *Journal of learning disabilities*, 44(5):472–488. 6

[36] Murray, J. and Goldbart, J. (2009a). Augmentative and alternative communication: a review of current issues. *Paediatrics and child health*, 19(10):464–468. 3

[37] Murray, J. and Goldbart, J. (2009b). Augmentative and alternative communication: a review of current issues. *Paediatrics and Child Health*, 19(10):464–468. 3

[38] Ouellette, G. P. (2006). What's meaning got to do with it: The role of vocabulary in word reading and reading comprehension. *Journal of educational psychology*, 98(3):554. 6

[39] Page, L., Brin, S., Motwani, R., and Winograd, T. (1999). The pagerank citation ranking: Bringing order to the web. Technical Report 1999-66, Stanford InfoLab. Previous number = SIDL-WP-1999-0120. 9

[40] Project Gutenberg (2025a). Project Gutenberg. Accessed: 2025-01-15. 71

[41] Project Gutenberg (2025b). The Princess and the Goblin by George MacDonald. Accessed: 2025-01-15. 71

[42] Project Gutenberg (2025c). The Tale of Peter Rabbit by Beatrix Potter. Accessed: 2025-01-15. 71

[43] Qi, P., Zhang, Y., Zhang, Y., Bolton, J., and Manning, C. D. (2020). Stanza: A Python natural language processing toolkit for many human languages. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*. 4

[44] Rahman, N. and Borah, B. (2022). An unsupervised method for word sense disambiguation. *Journal of King Saud University-Computer and Information Sciences*, 34(9):6643–6651. 7

[45] Rose, S., Engel, D., Cramer, N., and Cowley, W. (2010). Automatic keyword extraction from individual documents. *Text mining: applications and theory*, pages 1–20. 4

[46] Schütze, H., Manning, C. D., and Raghavan, P. (2008). *Introduction to information retrieval*, volume 39. Cambridge University Press Cambridge. 29

[47] SIGEDU (2024). 20th Workshop on Innovative Use of NLP for Building Educational Applications . Accessed: 2025-01-15. 82

[48] spaCy (2024a). Linguistic Features - Lemmatization . Accessed: 2025-01-15. 43

[49] spaCy (2024b). Matcher. Accessed: 2025-01-15. 65, 78

[50] spaCy (2024c). spaCy - Linguistic Features: Noun Chunks. Accessed: 2025-01-15. 49

[51] spaCy (2024d). spaCy - Word vectors and semantic similarity. Accessed: 2025-01-15. 61

[52] Speech Therapy Store (2024). 2,000+ Core Tier 2 Vocabulary Words. Accessed: 2025-01-15. 21

[53] Srivastava, R., Singh, P., Rana, K., and Kumar, V. (2022). A topic modeled unsupervised approach to single document extractive text summarization. *Knowledge-Based Systems*, 246:108636. 9

[54] Stanford NLP Group. (2024). Neural Pipeline - Lemmatization . Accessed: 2025-01-15. 43

[55] Storkel, H. L., Komesidou, R., Fleming, K. K., and Romine, R. S. (2017). Interactive book reading to accelerate word learning by kindergarten children with specific language impairment: Identifying adequate progress and successful learning patterns. *Language, speech, and hearing services in schools*, 48(2):108–124. 6

[56] Susan E Wagner High School (2024). TIER 2 and TIER 3 VOCABULARY TERMS – COMMON CORE STATE STANDARDS. Accessed: 2025-01-15. 21

[57] Teachers pay teachers (2024). Tier 3 Vocabulary Words. Accessed: 2025-01-15. 21

[58] The University of Tennessee Health Science Center (2022). Augmentative-Alternative Communication, Language & Literacy Lab (UT-AACL). Accessed: 2025-01-15. 3

[59] Universal Dependencies (2024). Universal Dependencies. Accessed: 2025-01-15. 43

[60] U.S. Department of Health and Human Services - National Institutes of Health (2019). NIDCD Fact Sheet — Voice, Speech, and Language Specific Language Impairment . [Accessed 01-02-2025]. 5

[61] Vasiliev, Y. (2020). *Natural language processing with Python and spaCy: A practical introduction*. No Starch Press. 10

Appendix

Appendix A

Raw Text from story books and television show transcripts

A.1 Raw Text from Project Gutenberg's The Tale of Peter Rabbit transcript

Once upon a time there were four little Rabbits, and their names were–

Flopsy, Mopsy, Cotton-tail, and Peter.

They lived with their Mother in a sand-bank, underneath the root of a very big fir-tree.

'Now my dears,' said old Mrs. Rabbit one morning, 'you may go into the fields or down the lane, but don't go into Mr. McGregor's garden: your Father had an accident there; he was put in a pie by Mrs. McGregor.'

[Illustration]

[Illustration]

'Now run along, and don't get into mischief. I am going out.'

Then old Mrs. Rabbit took a basket and her umbrella, and went through the wood to the baker's. She bought a loaf of brown bread and five currant buns.

[Illustration]

[Illustration]

Flopsy, Mopsy, and Cottontail, who were good little bunnies, went down the lane to gather blackberries:

But Peter, who was very naughty, ran straight away to Mr. McGregor's garden, and squeezed under the gate!

[Illustration]

[Illustration]

First he ate some lettuces and some French beans; and then he ate some radishes;

And then, feeling rather sick, he went to look for some parsley.

[Illustration]

[Illustration]

But round the end of a cucumber frame, whom should he meet but Mr. McGregor!

Mr. McGregor was on his hands and knees planting out young cabbages, but he jumped up and ran after Peter, waving a rake and calling out, 'Stop thief!'

[Illustration]

[Illustration]

Peter was most dreadfully frightened; he rushed all over the garden, for he had forgotten the way back to the gate.

He lost one of his shoes among the cabbages, and the other shoe amongst the potatoes.

After losing them, he ran on four legs and went faster, so that I think he might have got away altogether if he had not unfortunately run into a gooseberry net, and got caught by the large buttons on his jacket. It was a blue jacket with brass buttons, quite new.

[Illustration]

[Illustration]

Peter gave himself up for lost, and shed big tears; but his sobs were overheard by some friendly sparrows, who flew to him in great excitement, and implored him to exert himself.

Mr. McGregor came up with a sieve, which he intended to pop upon the top of Peter; but Peter wriggled out just in time, leaving his jacket behind him.

[Illustration]

[Illustration]

And rushed into the tool-shed, and jumped into a can. It would have been a beautiful thing to hide in, if it had not had so much water in it.

Mr. McGregor was quite sure that Peter was somewhere in the tool-shed, perhaps hidden underneath a flower-pot. He began to turn them over carefully, looking under each.

Presently Peter sneezed–'Kertyschoo!' Mr. McGregor was after him in no time.

[Illustration]

[Illustration]

And tried to put his foot upon Peter, who jumped out of a window, upsetting three plants. The window was too small for Mr. McGregor, and he was tired of running after Peter. He went back to his work.

Peter sat down to rest; he was out of breath and trembling with fright, and he had not the least idea which way to go. Also he was very damp with sitting in that can.

After a time he began to wander about, going lippity–lippity–not very fast, and looking all round.

[Illustration]

[Illustration]

He found a door in a wall; but it was locked, and there was no room for a fat little rabbit to squeeze underneath.

An old mouse was running in and out over the stone doorstep, carrying peas and beans to her family in the wood. Peter asked her the way to the gate, but she had such a large pea in her mouth that she could not answer. She only shook her head at him. Peter began to cry.

Then he tried to find his way straight across the garden, but he became more and more puzzled. Presently, he came to a pond where Mr. McGregor filled his water-cans. A white cat was staring at some gold-fish, she sat very, very still, but now and then the tip of her tail twitched as if it were alive. Peter thought it best to go away without speaking to her; he had heard about cats from his cousin, little Benjamin Bunny.

[Illustration]

[Illustration]

He went back towards the tool-shed, but suddenly, quite close to him, he heard the noise of a hoe–scr-r-ritch, scratch, scratch, scritch. Peter scuttered underneath the bushes. But presently, as nothing happened, he came out, and climbed upon a wheelbarrow and peeped over. The first thing he saw was Mr. McGregor hoeing onions. His back was turned towards Peter, and beyond him was the gate!

Peter got down very quietly off the wheelbarrow; and started running as fast as he could go, along a straight walk behind some black-currant bushes.

Mr. McGregor caught sight of him at the corner, but Peter did not care. He slipped underneath the gate, and was safe at last in the wood outside the garden.

[Illustration]

[Illustration]

Mr. McGregor hung up the little jacket and the shoes for a scare-crow to frighten the blackbirds.

Peter never stopped running or looked behind him till he got home to the big fir-tree.

He was so tired that he flopped down upon the nice soft sand on the floor of the rabbit-hole and shut his eyes. His mother was busy cooking; she wondered what he had done with his clothes. It was the second little jacket and pair of shoes that Peter had lost in a fortnight!

[Illustration]

I am sorry to say that Peter was not very well during the evening.

His mother put him to bed, and made some camomile tea; and she gave a dose of it to Peter!

'One table-spoonful to be taken at bed-time.'

[Illustration]

[Illustration]

But Flopsy, Mopsy, and Cotton-tail had bread and milk and blackberries for supper.

THE END

A.2 Raw Text from Continue to Know with WHRO - Rocks! transcript

Hello, everyone. I am so glad you came to join me today. My name is Mrs. Willis and today during our Science time, we are going to talk about rocks, the basics of rocks. We are going to talk about the different types of rocks and how rocks are created through the rock cycle. So let's get started.

Okay, so let's get started with the types of rocks. But first, what is a rock? A rock is made from one or more minerals. And when these minerals are put together it creates rocks. Rocks are broken down into three different categories or three different groups. The first one is igneous rock. Igneous rocks are formed from melted and cooled, magma or lava. So underneath the Earth's surface, we have magma. And when that hot magma melt and cools, it turns into an igneous rock, like basalt. But when magma comes through the Earth's surface it becomes lava. When that lava becomes cool, it turns into a type of rock like obsidian or the mahogany rock. So igneous rocks are created from melted and cooled, magma or lava. The next type of rock we have is sedimentary rocks. Sedimentary rocks are layers of sediment, cemented together with a lot of pressure. Sediments are broken down pieces of rocks. Broken through weathering and or erosion. Different types of rocks would include

shale or limestone. We also find fossils in sedimentary rocks. The third type of rock is metamorphic rock. Metamorphic rocks are existing rocks that are created by heat and pressure. The base word for metamorphic is morph-, and that means to change. So that helps us remember that metamorphic rock changes, like gneiss, is one type of metamorphic rock.

So now that we have the three different types of rocks, let's see how they get created through the rock cycle. So here I have the rock cycle, which is a cycle of how rocks get formed or get created. So if we were to look at igneous rocks, and as they breakdown in the sediments, through weathering and erosion they become sedimentary rocks. If the rocks go through an amount, an extensive amount of heat and pressure, they become metamorphic rocks. And as metamorphic rocks can become melted and cooled, it turn back into igneous rocks. But it's not an ever growing cycle. It can go other directions. So as in if the igneous rocks undergo a lot of heat and pressure, they turn into metamorphic rocks. And then as metamorphic rocks breakdown through weathering and erosion, they can become sedimentary rocks. And as sedimentary rocks also become melted and cooled, they can turn into igneous rocks. Wow, I know that that just seemed very confusing.

So let's make this a little bit simpler. You can also do this if you have some modeling clay, or Play-Doh around your house. So I'm first going to take some Play-Doh and I created a volcano. Okay. So as the magma comes through the Earth's surface and becomes lava, that lava becomes melted and cooled. And what happens is it becomes igneous rock. So right now I am creating an igneous rock. Igneous rocks are created from melted and cooled magma or lava. So let's say that going through weathering and erosion. This rock is going to be broken into smaller pieces called sediments. Sediments are smaller pieces of rock broken down from weathering. So I have some sediments here and these sediments are going to get pressed together or cemented together to create a sedimentary rock. Sedimentary rocks are created from segmented sediments pressed together. So now over time, this rock is going to undergo a lot of heat and pressure, over time, and then as it changes or morphs together, it

starts to become a metamorphic rock. Metamorphic rocks are existing rocks that get changed through an intense amount of heat and pressure over time. So again, we this rock can also be broken down into sediments and turn into a sedimentary rock. Or as it goes through a process of intense heat and gets melted and cooled, it can turn into an igneous rock. So with your modeling clay or Play-Doh, you can also create different types of rocks or put your rocks through the rock cycle. So now what we are going to do, is we are going to watch a video from eMediaVa.org, reviewing the three different types of rocks in the rock cycle. As you are watching this video, I want you to pay attention to how rocks are created, and different types of rocks. They will show different examples of these types of rocks as well. So, make sure you are staying tuned.

Video Start:

Idaho's beautiful mountains may look really solid, but like all rocks, they are constantly changing. That has to do with the rock cycle. There are three major types of rocks, Igneous, sedimentary, and metamorphic. Igneous rocks are formed when magma cools and become solid. This can happen below the surface of the Earth's crust, or closer to the surface like in the pits of volcanoes. Most of the Earth is made up of igneous rock. Now, sedimentary rocks are formed when stuff like sand and dirt are put down in layers, then squeezed by a large amount of pressure until the layers become solid. Over 75% of the Earth's land surface, is covered by sedimentary rock. Metamorphic rocks are rocks that already existed but were transformed by heat or pressure. Now as soon as rocks are formed, they begin to change. Rocks breakdown into smaller pieces through weathering. Heat, wind, water freezing and thawing, can all wear away and crack apart rocks. Plants and animals break up rocks too. The particles of the original rocks are removed by erosion, or swept away or blown away to a new location. When the deposits are buried, heat and pressure make them into new rocks, and the rock cycle begins all over again. How rocks are formed is one way to classify or identify rocks. Another way is to figure out what it's made of. A rock is made up of one or more minerals. Minerals are made up of one or

more of 92 different natural elements that combine together in different ways. Some rocks like gold and diamonds are very valuable, and others are well useful for a good skipping stones. Rocks take a long time to form and they are always changing. We live on them, build our homes out of them, and even wear them. Rocks and minerals are essential to our lives.

Video End:

Now, hope you enjoyed that video from eMediaVa.org. So let's do some review. Let's start with do you remember how rocks are made? Or what is a rock? Did you say made from one or more minerals? Then, you are correct. Rocks are created from one or more minerals together. There are three types of rocks; igneous rocks, sedimentary rocks, and metamorphic rocks. We talked about igneous rocks were created from melted magma or lava. You remember where magma is located? You say underneath the Earth's surface? Good job. So when magma underneath the surface is cooled, it also becomes igneous rocks. This as well as once magma comes through the Earth's surface and becomes lava, that lava is cooled and become igneous rocks, like our example. So, as rocks breakdown through weathering and erosion. They become sediments. You remember what sediments are? Did you say small pieces of rock? Good. So once these small pieces of rocks or small pieces of sediments are pressed together or cemented together, they become sedimentary rocks. You remember what we said you can find in sedimentary rocks? You remember fossils? Good job. Fossils can be found in sedimentary rocks. So again, as those sediments breakdown and they get cemented together, they are sedimentary rocks. Now, sedimentary rocks or igneous rocks can become metamorphic rocks. Metamorphic rocks again are existing types of rocks through heat and pressure, form or change into a new rock, like our example we have here. Through a lot of heat and pressure, these rocks become metamorphic rocks. This all happens through a certain process. You remember the name of that process when rocks are all created? Did you happen to say the rock cycle? Then, you would be right. The rock cycle is the way that different rocks get created. It gets created through weathering and erosion, melted and being cooled,

and a lot of heat and pressure. So, the next time that you are outside, and you happen to be looking around at different rocks you find in the ground, see if you can identify them as igneous, sedimentary, or metamorphic. Well guys, our time is up for today. I am so glad you were here today, and I hope you enjoyed our time together. Remember to always be discovering, always be investigating, and always ask questions. Have an awesome day.

A.3 Raw Text from WHROTV - Bill Nye the Science Guy - Photosynthesis

This corn is fresh. It's brand new. But it's made by the oldest living things on Earth, plants. ♪Bill Nye, the science guy ♪♪Bill Nye, the science guy ♪♪Bill, Bill, Bill, Bill ♪

- Hi, friends, Al Gea here. You wanna talk about selection? We got it all here. You want something green, take a look at this. Huh, this baby's nicer than my own lawn at home, and it is loaded. It's been completely re-seeded, it's filled with chlorophyll, and there are no moles, no mushrooms, no dandelions. This baby is ready to mow! ♪Bill, Bill, Bill, Bill, Bill ♪♪Bill, Bill, Bill, Bill, Bill ♪♪Bill Nye, the science guy ♪

- [Narrator] Brought to you by plants. They're nothing to sneeze at. - Did you know that without plants there would be no animals on earth? That includes animals like us humans. See, in a way, plants make all the air we breathe and all the food we eat. Different types of plants live all over the world in different places, and plants keep the whole world alive. Plants are amazing. See, plants take energy from the sun and make their own food. Now, when we say food, we mean sugar. For example, here's an apple, and apples are sweet, but it's a different flavor from, say, sugar from a sugar cane or orange juice from an orange or maple syrup from a maple tree. Different types of plants make different types of sugar. Take a look at this. If we burn the sugar, at first it'll turn brown. That's caramel. It's sweet, and it's good, but if we let it keep going, caramel burns, and it turns black. That's what we call carbon, and it forms

a gas called carbon dioxide. That's carbon hooked up with oxygen, carbon dioxide. Now, this carbon dioxide goes up this tube and bubbles in this special detector liquid. Pretty soon, with enough carbon dioxide, it's gonna turn a different color, a kind of orange-y yellow, but it takes a minute.

- [Narrator] Is it soup yet?

- It's almost almost ready.

- [Narrator] Ah.

- See, the detector chemical has changed color, it's not blue anymore, it's orange. That's because we were making carbon dioxide gas. It's the same gas we make all day, all the time, every time we breathe. Watch, here's some of the same detector chemical.

- See, it's changed color because I'm giving off the same gas as in that chemical reaction. Now, plants can do the same chemical reaction only the other way around. Plants take in carbon dioxide from the air and give off oxygen. Take a look at this. It's the same blue indicator liquid, and it stays blue because these plants are absorbing carbon dioxide and giving off oxygen. Now, this process where plants take in water and sunlight and make food is called photosynthesis. It means making from sunlight. Now, the key to photosynthesis is a green chemical called chlorophyll. Chlorophyll is what makes you get grass stains on your pants when you fall down. Now, plants take in carbon dioxide given off by animals and fungus and some other plants and make oxygen, which animals like us breathe. See, it's a great deal for everybody. Mm, plant's sugar, it is a good deal.

- When plants make food, they put the oxygen that we breathe into the air. It's photosynthesis, and you can see it using this piece of lettuce. Just put the piece of lettuce in a jar full of water. Then turn the jar over into a bowl. And fill the bowl with water. Then put the jar in a place where it'll get lots of sun. Wait about 24 hours. See those little bubbles? That's oxygen that the lettuce is making by exchanging carbon dioxide and water for food. Woo, ain't that funky now? Hi, how 'bout this?

This is nicer than my own lawn at home, and you wouldn't do this on a Beamer. Hey, friends, come on down, and see me today!

- Plants have found ways to live all over the earth, even where you least expect it.

- Well, the car's covered with grass. Let's go. Plants! You ready?

- [Cameraman] Yeah!

- Plants are everywhere! Yep, it's a grass-covered vehicle. It's a plant-mobile is what it is.

- [Child] Bill Man, the science guy, huh?

- Yeah, yeah.

- [Child] You look different!

- [Motorcyclist] Bill Nye, the science guy!

- [Bill Nye] Hiya, guys.

- [Child] You can't hardly breathe in here.

- [Bill Nye] Yeah you can! You know they say cars and plants don't mix. It's okay! You know, all living things depend on plants. The car's not alive, but the grass is.

- [Person In Helicopter] What do you mean what color is it? It's a grass car, it's green!

- See the air rushing over the blades of grass? Taking carbon dioxide from the air, light from the sun and water, and the grass is making its own food, it's growing. It's also making oxygen for us to breathe. The water, well, we provide that when it rains or when we hose it down from time to time. It's science. Science! It's a lot of work, it takes a lot of energy to make an apple pie. Just think how much energy it takes an apple tree to grow an apple. Now, why do they do it, why do they bother? Well, please, consider the following. See, apple trees are plants, and they can't move around. They're stuck, they're planted, they're plants for crying out loud! So, they have to find a way to spread their seeds out. See, plants don't want the young plants growing underneath them, otherwise they'd be competing for the same patch of soil and the same patch of sunlight. So, plants like apple trees grow these big, yummy, delicious fruit, bringing tremendous amount of water up to their

roots, getting a tremendous amount of energy from the sun, just so that animals will come along and take the fruit somewhere else. See, the fruit isn't used for fertilizer, it's used only to get animals to carry the seeds to some new place so that the seeds will grow away from the old trees. That's the only reason apple trees make apples. Isn't that wild? See, plants have all kinds of tricks for moving their seeds around, like dandelions use the wind. And maple trees make these cool helicopter seeds. See, plants have this whole thing figured out. Well, it's a good use of energy when you think about it. Thanks for joining me and consider the following.

- [Man On TV] Each seed of the dill weed plant is attached to a feather-like carrier. The wind carries the seeds away to a new area for growth, perhaps the distance of several miles.

- [Narrator] This is Master Plant Theater.

- Good evening, I am Aleister Appleseed. Fruit, as we all know, is the part of the plant where the seeds are stored. Plants that want their seeds moved by a hungry bird or animal produce fruit that is tasty and good-looking. Take for example the tomato. Thank you. Note the attractive red color and the taste is delicious. An animal would be inclined to eat this fruit. The seeds would pass through the digestive system and into the soil, which one hopes would lead to more tomatoes. Thank you. Fruit, an amazing way that plants have adapted to living on earth.

- [Aleister] Thank you.

- [Child] Here you go. Take that!

A.4 Raw text from Daniel Tiger - Baby is Here: At the hospital transcript

*Come On, let's go.

*I'm so excited

*Mom! Hi!

*Hi Daniel, I love you so much.

*I love you too. Ugga Mugga

*(gurgles) is that the baby?

*Mm-hmm. Wow, Is it a boy or a girl

*It's a girl, a girl?

*Ooh, the babies a girl baby

*I have a baby sister, this is grr-ific

*Can I see her?-Yes big brother, go meet your baby sister.

*The baby is in there! let's look.

*She so-so little

*Do you want to hold her?-can I?

*Sure, come sit in this chair over here

*Okay, I Going to hold the baby.

*Big Brother Daniel, Meet your baby sister

*Hi Baby Sister, I'm your big brother Daniel.

*Ga, Ga? (Gurgling)

*Look at her little noses and her little ears and her little paws.

*Oh look, she's touching my paw.

*Mom, Dad, Grandpere, Look, she's touching my paw.

*Maybe she's saying Hi!

*Hi baby sister, I couldn't wait to meet you.

*Playing music in the park made the waiting time easier and reading my favorite baby book, Margaret's Music.

*Margaret's Music? I Always loved that book and that nome. Me too?

*I had a Grandma Margaret who was very special to me.

*Do you think.. We should name the baby... Margaret?

*Baby Margaret? I like that!

*You look like a Baby Margaret.

*You Do Look Like A Baby Margaret

*Margaret Tiger, I Like the sound of that.

*Margaret it is.

*My Baby Sister Is named Margaret. Hello baby Margaret! Hello!

*I want To give Baby Margaret the book.

*Grandpere, do you have the... present?

*I do! I do!

*Here it is big brother Daniel.

*I'm so excited to give this book to Baby Margaret.

*Here you go Baby Margaret, this is for y—- (wailing) Oh, the baby's crying

*I can't give her the book if she's crying

*I guest I'll give to her later. Why is Margaret Crying?

*Well... Maybe She's Hungry.

*I'm going to feed her and then we'll be ready to go home.

*OK, OK, Daniel Will you help me get the baby Carriage ready?

*It's Outside.-OK!

*Thank you my helper tiger.

Appendix B

Keywords comparison from different Keyterm extraction packages

Water Takes Three Forms	
Library	Keywords extracted
SENCE	form, gas, water, liquid, solid, vapor, shape, ice, change, turn
Yake	water, liquid, gas, vapor, solid, ice, shape, form, call, change
Rake	water vapor becoming liquid, water vapor also, take ice cubes, vapor loses heat, see liquid water, change liquid water, water vapor, liquid water, called vapor, often see
Bert	water, liquid, gas, boil, evaporate, solid, container, ice, flow, shape
Textacy	liquid water, water vapor, water boil, gas form, solid form, gas escape, ice cube, shape, cup, container
SgRank	form, vapor, water, cube, boil, escape, solid, liquid, ice, gas
TextRank	water, liquid, form, vapor, gas, ice, solid, change, turn, happen

The Invention of Paper	
Library	Keywords extracted
SENCE	paper, china, use, invent, cai, lun, process, emperor, know, person
Yake	paper, China, make, Cai, Lun, invent, person, hemp, silk, good
Rake	people started using cai lun ', china named cai lun came, cai lun used tree bark, papermaking process throughout china, china around 100 b, cai lun, plant called hemp, process also made, people used, paper made back
Bert	papermaking, paper, invent, inventor, silk, bamboo, china, write, lun, material
Textacy	new paper, China, Cai Lun, B.C. People, turtle shell, emperor, cheap way, process, tree bark, fishing net
SgRank	way, shell, paper, net, bark, People, Lun, turtle, tree, new
TextRank	paper, use, China, Lun, Cai, process, invent, emperor, start, good

The History of Juneteenth	
Library	Keywords extracted
SENCE	people, juneteenth, celebrate, enslave, day, holiday, year, emancipation, proclamation, state
Yake	Juneteenth, people, Emancipation, Proclamation, American, Texas, enslave, June, Union, celebrate
Rake	union soldiers finally arrived, story begins years earlier, freed people began celebrating, many people hope juneteenth, took union soldiers, slave owners knew, president lincoln gave, nation celebrate freedom, people even read, african american stories
Bert	emancipation, juneteenth, june, proclamation, holiday, lincoln, slavery, day, slave, 1865
Textacy	enslaved people, Juneteenth, important news, national holiday, african american story, People, date important, historic day, Emancipation Proclamation, northern state
SgRank	story, news, state, soldier, people, important, holiday, american, War, Proclamation
TextRank	people, Juneteenth, celebrate, day, state, American, Emancipation, news, year, Proclamation

The Music of the Cherokee	
Library	Keywords extracted
SENCE	water, use, play, music, cherokee, chant, dance, drum, ritual, sing
Yake	Cherokee, chant, music, drum, dance, ritual, sing, play, water, ago
Rake	would stretch animal skins, cherokee mainly played flutes, cherokee use similar melodies, cherokee still play music, drummer would put, hard outer skin, chorus responds together, cherokee turned gourds, water would change, many different ways
Bert	cherokee, tribe, rhythmic, ceremonial, traditional, ritual, music, chant, tradition, drum
Textacy	Cherokee music tradition, Cherokee chant, Cherokee dance, Cherokee tribe, Cherokee life, dance music, interesting musical history, water drum, traditional dance, musical tradition
SgRank	skin, music, drum, history, dance, melody, life, word, voice, section
TextRank	Cherokee, chant, music, dance, drum, change, use, ritual, play, instrument

Musical Instruments: Brass Instruments	
Library	Keywords extracted
SENCE	use, musician, brass, instrument, tube, air, length, metal, end, mouthpiece
Yake	instrument, brass, tube, air, length, make, mouthpiece, U-shaped, musician, end
Rake	militaries still use brass instruments, brass instruments include trumpets, valves make different notes, enters tubes wound round, brass instruments make sound, brass instruments became used, brass instrument depends, time period called, looks like gold, brass instruments
Bert	brass, instrument, trumpet, horn, trombone, musician, sound, metal, musical, loud
Textacy	brass instrument, musical instrument, loud, military use, metal trumpet, bold, shiny metal, tube, small hole, ancient Egypt
SgRank	instrument, hole, trumpet, use, period, occasion, metal, horn, cup, buzz
TextRank	instrument, brass, tube, use, air, end, length, mouthpiece, trumpet, metal

Appendix C

Text of passages used for Keyword extraction comparison in Chapter 7

C.1 Water Takes Three Forms, Grade 2, Topic area: Science

Text: Water comes in three forms: liquid, solid, and gas. Water can be a liquid. It flows. It has no shape of its own. A liquid takes the shape of its container. Water can be a solid. Solids have their own shape. Water in its solid form is called ice. Water can be a gas. Gas has no shape. Water in its gas form is called vapor. You can see liquid water after it changes to a solid. Pour water into a cup. Put the cup into the freezer. The next day, the water will have turned into ice. Ice can change back to liquid water. Observe this: Take ice cubes from the freezer. Put a few of them on a plate. They will melt and turn into liquid water. Heat can change liquid water to a gas. What happens when a pot of water boils? Bubbles begin to form. Then the water starts to evaporate. You can often see the gas escape as water vapor. Water vapor also can turn back into a liquid. That happens when the vapor loses heat. The process of water vapor becoming liquid is called condensation.

C.2 Light and Objects, Grade 3, Topic area: Science

Text:

Have you ever wondered what happens when a line or path of light bumps into something in its way? Different things may happen depending on what exactly is in the light's path. If a path of light hits something that is transparent, most of the light will pass right through. Air, water, and glass are all transparent. When light hits these transparent objects, it passes through to the other side. It is almost as if the object isn't there. Most buildings have glass windows so that natural sunlight can travel from the outdoors inside. Have you ever been in a building that has a glass roof or skylight? Sometimes you can even see blue sky and clouds through the skylight! Light cannot travel through all materials. If a path of light hits something that is opaque, the light is absorbed and blocked by the object. It cannot continue in a straight line through the object. Wood, cardboard, and even a person's body are all opaque objects. Light cannot pass through to the other side. Instead, a shadow is created because the light is absorbed. Look around your classroom. Do you see transparent objects through which light is passing? Can you also find opaque objects? You will probably find that your classroom has many more opaque objects than transparent objects. Do you see any shadows?

C.3 American Government - James Madison: A Man with a Plan, Grade 4, Topic area: Social Studies

Text: After winning independence from Britain, the United States' early days were rocky ones. The national government set up by the Articles of Confederation was weak. It could make laws and rules. It could not, however, make the states follow

them. Each of the 13 states acted almost like an independent country. Each state had its own currency. Each state set its own laws. The states weren't working together for the good of the nation, so they bickered constantly. No one could agree which states would pay for the Revolutionary War with Britain. The national government could not collect taxes from the states. The state governments were simply too strong. National leaders like George Washington worried out loud. The national government was "little more than the shadow without the substance," he said. Things got so bad that the states finally agreed to take action. A man from Virginia named James Madison led the way. He called for the Constitutional Convention. Men from twelve of the thirteen states met at the convention in Philadelphia in 1787 to write the Constitution. The document would create a new national government. Washington was there. Other important leaders, like Alexander Hamilton and Benjamin Franklin, also came. Madison arrived at the convention armed with a plan. Because Madison was from Virginia, he called it the Virginia Plan. Madison's plan argued that each state's votes in Congress should be based on the number of people living in that state. Before, each state had gotten one vote in Congress no matter how big or small it was or how many people lived there. Not everyone agreed with Madison's ideas. The delegates at the convention asked for many compromises. Without them, they wouldn't sign a final draft of the United States Constitution. These compromises were difficult to reach. In the end, they helped strengthen the United States government. Still, Madison's Virginia Plan was very important. For all his work, James Madison is known today as the "Father of the Constitution."

Appendix D

SENCE code

assessStudentKnowledge.py

```
import configparser
from datetime import datetime
import pyinputplus as pyip
import pymongo
import spacy
import tabulate
from toolbelt import SentenceSimilarityMeasures


class AssessSentences:
    db_name = "sence"  # name of database
    sentences_for_passage_collection = "sentences_for_passage"
    sentence_collection = "sentences"  # name of passage collection
    passage_collection = "passage"
    keywords_collection = "keywords"  # name of keywords collection
    modified_sentences_collection = "modified_sentences"
```

```
17      student_response_collection = "student_response"
18      passages_hash = {}
19      sentences_hash = {}
20      keywords_hash = {}
21
22      def __init__(self):
23          self.client = pymongo.MongoClient("mongodb://localhost:27017/"
    )
24          self.username = "system"
25          self.sort_type = ''
26          self.chosen_passage_id = 0
27          self.kw_picked_ids = []
28          self.keywords = []
29          self.sentences = []
30          self.all_sents = {}
31          self.num_keys = 0
32          self.nlp = spacy.load('en_core_web_sm')
33          config = configparser.ConfigParser()
34          try:
35              config.read('../config.ini')
36              self.sentence_blank = config['SENTENCES']['sentence_blank'
    ]
37          except KeyError:
38              config.read('config.ini')
39              self.sentence_blank = config['SENTENCES']['sentence_blank'
    ]
40
```

```
41      # get the last index from the collection to determine the next ID
        to store
42      def get_last_index_from_collection(self, collection_name):
43          new_index = 0
44          num_docs_curs = self.client[AssessSentences.db_name][
        collection_name].find(
45              {}, {
46                  "_id": 1}).sort("_id", pymongo.DESCENDING).limit(1)
47          num_docs_curs_l = list(num_docs_curs)
48          if len(num_docs_curs_l) > 0:
49              new_index = num_docs_curs_l[0]['_id'] + 1
50          return new_index
51
52      def get_instructor_choices(self):
53          kw_ids_list = set()
54          passage_ids_list = []
55          counter = 0
56          # find unique values of passage num, sort type and keywords
        from sentences_for_passage collection.
57          # we need unique values because there are multiple rows for
        simple, complex, medium sentences for each (passage_num, sort_type
        , keywords) set
58          db_res = self.client[AssessSentences.db_name][AssessSentences.
        sentences_for_passage_collection].aggregate([
59              {"$group": {"_id": {"passage_num": "$passage_num", "sort":
         "$sort", "keywords": "$keywords"}}}
60          ])
61          db_res_list = list(db_res)
```

```
62          for item in db_res_list:
63              sent_details = item['_id']
64              passage_ids_list.append(sent_details['passage_num'])
65              kw_ids_list.update(sent_details['keywords'])
66          # go to passage table and get all the passage titles
67          passage_info = self.client[AssessSentences.db_name][
    AssessSentences.passage_collection].find(
68              {'_id': {'$in': passage_ids_list}}, {"title": 1}).sort([("
    title", pymongo.ASCENDING)])
69          for item in passage_info:
70              self.passages_hash[item['_id']] = item['title']
71          # make a list of all the keyword ids
72          # go to keywords collection and get all the keywords
73          kw_info = self.client[AssessSentences.db_name][AssessSentences
    .keywords_collection].find(
74              {'_id': {'$in': list(kw_ids_list)}}, {"keyword": 1})
75          # add all keywords to a kw_id: kw_word hash
76          for item in kw_info:
77              self.keywords_hash[item['_id']] = item['keyword']
78          # display to user
79          table = []
80          table_headers = ["Index number", "Title", "Keywords", "Sort
    type"]
81          for sent_item in db_res_list:
82              item = sent_item['_id']
83              # get the passage ID, passage title, keywords chosen and
    sort type
84              kw_str = ''
```

```
85              p_title = self.passages_hash[item['passage_num']]
86              for kw_item in item['keywords']:
87                  kw_str += " " + self.keywords_hash[kw_item]
88              sort_type = item['sort']
89              table.append([counter, p_title, kw_str, sort_type])
90              counter += 1
91          # get default and user supplied keywords for selected chapter
92          print(tabulate.tabulate(table, table_headers))
93          choice = pyip.inputNum(
94              "Enter the ID number (first column) of the chapter you
    want to run the assessment for today. \n "
95              "Please note that if you do not find the chapter you would
     like to use, you might need to run "
96              "sentence generation for them first before this step: ")
97          chosen_item = db_res_list[choice]
98          chosen_passage = chosen_item['_id']
99          self.sort_type = chosen_passage['sort']
100         self.chosen_passage_id = chosen_passage['passage_num']
101         self.kw_picked_ids = chosen_passage['keywords']
102
103     def get_sentences(self):
104         sent_ids = []
105         mod_sent_ids = []
106         simple_sents = []
107         medium_sents = []
108         complex_sents = []
109         other_sents = []
110         toolbelt = SentenceSimilarityMeasures()
```

```
111         # add all the keywords to the global variable
112         self.keywords = [self.keywords_hash[kid] for kid in self.
    kw_picked_ids]
113         # make a list of all the sentence system ids
114         # go to sentences collection and get all the system sentences
115         db_res = (self.client[AssessSentences.db_name][AssessSentences
    .sentences_for_passage_collection].find
116                   ({'passage_num': self.chosen_passage_id, 'sort':
    self.sort_type,
117                     'keywords': {'$in': self.kw_picked_ids}}))
118         db_res_list = list(db_res)
119         # create a list of simple, medium and complex sentences with
    the following attributes
120         # id, sentence, keywords in order of appearance in sentence,
    sentence with blanks, response of student
121         for item in db_res_list:
122             # check if sentence is system or user generated and put in
     appropriate bucket
123             sent_ids.append(item["sentence_id"]) if item['
    sentence_type'] == "system" else mod_sent_ids.append(
124                 item["sentence_id"])
125
126         sentences = self.client[AssessSentences.db_name][
    AssessSentences.sentence_collection].find(
127             {'_id': {'$in': sent_ids}})
128         sentences_list = list(sentences)
129         if len(mod_sent_ids) > 0:
```

```
130             mod_sentences = (self.client[AssessSentences.db_name][
    AssessSentences.modified_sentences_collection].find
131                              ({'_id': {'$in': mod_sent_ids}}))
132             for item in mod_sentences:
133                 self.sentences_hash['M' + str(item['_id'])] = item['
    modified_sentence']
134         for item in sentences_list:
135             self.sentences_hash[item['_id']] = item['sentence']
136         for item in db_res_list:
137             s = dict(s_id=item['sentence_id'],
138                      sentence=self.sentences_hash[item['sentence_id']]
     if item['sentence_type'] == "system"
139                      else self.sentences_hash['M' + str(item['
    sentence_id'])])
140             s['sent_question'] = item['sentence_with_spaces']
141             s['sent_response'] = ''
142             s['asked'] = "no"
143             s['sentence_source'] = item['source']
144             s['kw_tok_pos'] = toolbelt.find_match_return_tok_pos(s['
    sentence'], self.keywords)
145             if item['source'] == 'other':  # add the sentences from
    other passages to their own bucket
146                 other_sents.append(s)
147             elif item['difficulty_level'] == 'simple':  # identify
    simple sentences
148                 simple_sents.append(s)
149             elif item['difficulty_level'] == 'medium':  # identify
    medium sentences
```

```
150                 medium_sents.append(s)
151             else:
152                 complex_sents.append(s)  # identify complex sentences
153         self.all_sents['simple'] = simple_sents
154         self.all_sents['medium'] = medium_sents
155         self.all_sents['complex'] = complex_sents
156 
157     def check_num_inputs(self, response):
158         num_allowed = self.num_keys
159         if len(response) > 0 and len(response.split()) > num_allowed:
160             raise Exception("You entered too many words. Try again.")
161         elif len(response) > 0 and len(response.split()) < num_allowed
    :
162             raise Exception("You did not enter enough words. Try again
    .")
163 
164         return response
165 
166     def check_if_sentence_present(self, sentence):
167         all_sents = list(self.sentences)
168         lower_sents = [s.lower() for s in all_sents]
169         if sentence in lower_sents:
170             return True
171         else:
172             return False
173 
174     # this method compares the lemmatized versions of two sentences to
    make sure that they are the same in the base form
```

```
175     def compare_two_sentences(self, sent1, sent2):
176         sent1_nlp = self.nlp(sent1)
177         sent2_nlp = self.nlp(sent2)
178         sent1_toks = " ".join([tok.lemma_ for tok in sent1_nlp])
179         sent2_toks = " ".join([tok.lemma_ for tok in sent2_nlp])
180         if sent1_toks == sent2_toks:
181             return True
182         else:
183             return False
184 
185     def do_assessment(self):
186         # TODO: scaffolding
187         # show all keywords
188         print("Use one of the following words for each blank provided:
    ", ", ".join(self.keywords))
189         if "simple" in self.all_sents.keys():
190             for ind, sent in enumerate(self.all_sents['simple']):
191                 self.num_keys = len(sent['kw_tok_pos'])
192                 self.all_sents['simple'][ind]['sent_response'] = pyip.
    inputCustom(
193                     self.check_num_inputs, prompt=sent['sent_question'
    ])
194         if "medium" in self.all_sents.keys():
195             for ind, sent in enumerate(self.all_sents['medium']):
196                 self.num_keys = len(sent['kw_tok_pos'])
197                 self.all_sents['medium'][ind]['sent_response'] = pyip.
    inputCustom(
```

```
198                     customValidationFunc=self.check_num_inputs, prompt
=sent['sent_question'])
199         if "complex" in self.all_sents.keys():
200             for ind, sent in enumerate(self.all_sents['complex']):
201                 self.num_keys = len(sent['kw_tok_pos'])
202                 self.all_sents['complex'][ind]['sent_response'] = pyip
.inputCustom(
203                     customValidationFunc=self.check_num_inputs, prompt
=sent['sent_question'])
204
205     def check_each_kw(self, sents_to_check):
206         evaluated_sentences = []
207         toolbelt = SentenceSimilarityMeasures()
208         for ind, s in enumerate(sents_to_check):
209             kw_tok_list = list(s['kw_tok_pos'].values())
210             updated_sentence = s['sent_question']
211             for kw_ind, kw in enumerate(s['sent_response'].split()):
212                 updated_sentence = toolbelt.replace_words_in_sentence(
updated_sentence, kw, kw_tok_list[kw_ind])
213             if self.check_if_sentence_present(updated_sentence):
214                 # save results with og sentence, keywords, student
sentence, right / wrong?
215
216                 eval_sent = {'og_sentence': s['sentence'], '
response_sentence': updated_sentence,
217                              'passage': self.passages_hash[self.
chosen_passage_id], 'sort_type': self.sort_type,
```

```
218                             'source': s['sentence_source'], 'correct'
: True}
219             else:
220                 doc = self.nlp(updated_sentence)
221                 rep_sentence = updated_sentence
222                 kws = s['kw_tok_pos'].keys()
223                 kw_tok_pos = s['kw_tok_pos']  # get the word positions
 of the keywords in the sentence
224                 for kw_word, kw_pos in kw_tok_pos.items():
225                     typo_correct = toolbelt.typos_closest_match(doc[
kw_pos].lemma_, kws)
226                     if typo_correct.lower() == kw_word.lower():
227                         rep_sentence = toolbelt.
replace_words_in_sentence(updated_sentence, typo_correct, kw_pos)
228                         doc = self.nlp(rep_sentence)
229                 if self.compare_two_sentences(rep_sentence.lower(), s[
'sentence'].lower()):
230                     eval_sent = {'og_sentence': s['sentence'], '
response_sentence': updated_sentence,
231                                  'passage': self.passages_hash[self.
chosen_passage_id], 'sort_type': self.sort_type,
232                                  'source': s['sentence_source'], '
correct': True}
233                 else:
234                     eval_sent = {'og_sentence': s['sentence'], '
response_sentence': updated_sentence,
235                                  'passage': self.passages_hash[self.
chosen_passage_id], 'sort_type': self.sort_type,
```

```
                                    'source': s['sentence_source'], '
correct': False}
            evaluated_sentences.append(eval_sent)
        return evaluated_sentences

    # this method stores results of student responses in the student
response collection with necessary metadata
    def save_results_to_db(self, sentences_to_save, std_name,
eval_time):
        index_beg = self.get_last_index_from_collection(self.
student_response_collection)
        if len(sentences_to_save) > 0:
            for s in sentences_to_save:
                try:
                    insert_row = {"_id": index_beg,
                                  "student_name": std_name,
                                  "time_assessed": eval_time,
                                  "sentence_original": s['og_sentence'
],
                                  "sentence_response": s['
response_sentence'],
                                  "passage": s['passage'],
                                  "sort_type": s['sort_type'],
                                  "keywords": self.keywords,
                                  "source": s['source'],
                                  "correct": s['correct']

                                  }
```

```
258                     self.client[self.db_name][
259                         self.student_response_collection].insert_one(
260                         insert_row)
261                     index_beg += 1
262                 except pymongo.errors:
263                     print("Could not insert %s in database.")
264
265     def evaluate_responses(self):
266         self.sentences = self.sentences_hash.values()
267         eval_time = datetime.today().strftime('%Y-%m-%d %H:%M:%S')
268         std_name = pyip.inputStr("Please enter your (student's) name:
    ")
269         if 'simple' in self.all_sents.keys():
270             results_to_save_simp = self.check_each_kw(self.all_sents['
    simple'])
271             self.save_results_to_db(results_to_save_simp, std_name,
    eval_time)
272         if 'medium' in self.all_sents.keys():
273             results_to_save_med = self.check_each_kw(self.all_sents['
    medium'])
274             self.save_results_to_db(results_to_save_med, std_name,
    eval_time)
275         if 'complex' in self.all_sents.keys():
276             results_to_save_complex = self.check_each_kw(self.
    all_sents['complex'])
277             self.save_results_to_db(results_to_save_complex, std_name,
     eval_time)
278         print("Results have been saved.")
```

```
279         return 0
280
281
282 if __name__ == "__main__":
283     assess_sentences = AssessSentences()
284     assess_sentences.get_instructor_choices()
285     assess_sentences.get_sentences()
286     assess_sentences.do_assessment()
287
288     sents = assess_sentences.evaluate_responses()
```

config.ini

```
1 [FILE_PATHS]
2 tools = /Users/tsamuel/Dropbox/GradSchool/thesisWork/tools/stanza/
3 base_path = /Users/tsamuel/Dropbox/GradSchool/thesisWork/data/html/
4 grade4_json = grade4_vocab.json
5 grade5_json = /grade5.json
6 base_erupt_path = /Users/tsamuel/Dropbox/GradSchool/thesisWork/data/
      erupt/
7 base_erupt_chapters_1_2 = erupt_chapters_1_2.txt
8 base_erupt_passages = erupts_chapters_1_2.json
9 base_erupt_high_freq_words = erupt_high_freq_words_txtrank.json
10 erupt_ngrams_sentences = erupt_ngram_sentences.json
11 base_ngrams_sentences = base_ngrams_sentences.json
12 tier_words_path = /Users/tsamuel/Dropbox/GradSchool/thesisWork/
      tier_words/
13
```

```
[SENTENCES]
sentence_blank = ----------
num_sentences_to_return_passage = 2
num_sentences_to_return_other = 1
# data is from https://universaldependencies.org/docs/en/dep/
sentence_location_filter_simple = nsubj nsubjpass csubj csubjpass ROOT
sentence_location_filter_medium = dobj pobj obj
```

config.json

```
{
"PENN_TREEBANK": {
  "penn_treebank_orig" : {
  "CC": "Coordinating conjunction",
  "CD": "Cardinal number",
  "DT": "Determiner",
  "EX": "Existential there",
  "FW": "Foreign word",
  "IN": "Preposition or subordinating conjunction",
  "JJ": "Adjective",
  "JJR": "Adjective, comparative",
  "JJS": "Adjective, superlative",
  "LS": "List item marker",
  "MD": "Modal",
  "NN": "Noun, singular or mass",
  "NNS": "Noun, plural",
  "NNP": "Proper noun, singular",
  "NNPS": "Proper noun, plural",
```

```
"PDT": "Predeterminer",
"POS": "Possessive ending",
"PRP": "Personal pronoun",
"RB": "Adverb",
"RBR": "Adverb, comparative",
"RBS": "Adverb, superlative",
"RP": "Particle",
"SYM": "Symbol",
"TO": "to",
"UH": "Interjection",
"VB": "Verb, base form",
"VBD": "Verb, past tense",
"VBG": "Verb, gerund or present participle",
"VBN": "Verb, past participle",
"VBP": "Verb, non-3rd person singular present",
"VBZ": "Verb, 3rd person singular present",
"WDT": "Wh-determiner",
"WP": "Wh-pronoun",
"WRB": "Wh-adverb "
},
"penn_treebank_condensed" : {
"CC": "Conjunction",
"CD": "Number",
"DT": "Determiner",
"EX": "Existential there",
"FW": "Foreign word",
"IN": "Preposition",
"JJ": "Adjective",
```

```
47   "JJR": "Adjective",
48   "JJS": "Adjective",
49   "LS": "List item marker",
50   "MD": "Modal",
51   "NN": "Noun",
52   "NNS": "Noun",
53   "NNP": "Proper noun",
54   "NNPS": "Proper noun",
55   "PDT": "Predeterminer",
56   "POS": "Possessive ending",
57   "PRP": "Personal pronoun",
58   "RB": "Adverb",
59   "RBR": "Adverb",
60   "RBS": "Adverb",
61   "RP": "Particle",
62   "SYM": "Symbol",
63   "TO": "to",
64   "UH": "Interjection",
65   "VB": "Verb",
66   "VBD": "Verb",
67   "VBG": "Verb",
68   "VBN": "Verb",
69   "VBP": "Verb",
70   "VBZ": "Verb",
71   "WDT": "Wh-determiner",
72   "WP": "Wh-pronoun",
73   "WRB": "Wh-adverb "
74 }
```

```
}}
```

createSystemSentencesforMongo.py

```
# This script does the following
# i. Accepts an input text
# ii. For each sentence in the input text, identified keywords from mongo keywords_sys
# iii. Identifies parts of speech in the sentence and complexity of sentence
# iv. Adds sentence to database
import configparser
import json
import re

import pyinputplus as pyip
import pymongo
import spacy

from customStanza import Stanza

config = configparser.ConfigParser()
config.read('../config.ini')
tools_path = config['FILE_PATHS']['tools']

with open('../config.json') as fc:
    config_json = json.load(fc)

```

```
23 nlp = spacy.load("en_core_web_sm")
24 client = pymongo.MongoClient("mongodb://localhost:27017/")
25 db_name = "sence"  # name of database
26 passage_collection = "passage"  # name of passage collection
27 keywords_collection = "keywords"
28 sentences_collection = "sentences"
29 sNLP = Stanza()
30
31
32 def return_key_ids(kw_cursor_list, keyws):  # returns list of ids for
       keywords provided
33     kw_ids = []
34     for kw_item in kw_cursor_list:
35         if kw_item['keyword'] in keyws:
36             kw_ids.append(kw_item['_id'])
37     return kw_ids
38
39
40 def main():
41     # retrieve all passage titles from database
42     db_res = client[db_name][passage_collection].find({}, {"title":
       1})
43
44     print("Index number \t Title")
45     for item in db_res:
46         print("%d \t %s" % (item['_id'], item['title']))
47     print("Enter the Index number of the chapter you want to use today
       :")
```

```
48      # TODO: error checking for wrong index; use pyip smarter
49      chap_choice = pyip.inputNum()
50      # check if sentences have already been generated for this passage.
        If they have, do not re-generate
51      already_gen = client[db_name][sentences_collection].
    count_documents({'passage_id': int(chap_choice)})
52      if already_gen == 0:
53          # retrieve last index of key_words_user_collection
54          num_docs = client[db_name][sentences_collection].
    count_documents({})
55          index_st = num_docs + 1
56          # retrieve passage chosen by user from db
57          mn_passage = client[db_name][passage_collection].find({'_id':
    chap_choice})
58          passage = mn_passage[0]['text']
59          # retrieve system keywords of passage from db
60          passage_kw = client[db_name][keywords_collection].find(
61              {'passage_num': {'$in': [int(chap_choice)]}, 'inserted_by'
    : 'system'},
62              {'keyword': 1, '_id': 1})
63          passage_kw_list = list(passage_kw)
64          # check if there are keywords already generated
65          # if not, do not proceed
66          if len(passage_kw_list) > 0:
67
68              # create a | separated list of keywords to use for regex
    searching below
```

```
69          kw_string_sep = '|'.join(str(w['keyword']) for w in
    passage_kw_list)
70          print(kw_string_sep)
71
72          # retrieve condensed human-readable POS tags from config
    file
73          pos_cond = config_json['PENN_TREEBANK']['
    penn_treebank_condensed']
74
75          doc = nlp(passage)
76          pos_not_found = []
77          for sentence in doc.sents:
78              # lemmatize the sentence
79              lm_sentence = sNLP.lemmatize_sentence(sentence.text)
80
81              # retrieve all POS in sentence [output is tuple of POS
     of each word in sentence]
82              pos_str = sNLP.pos(sentence.text)
83
84              # retrieve unique value and condensed human-readable
    version of POS; if check to ignore punc.
85              pos_set = set()
86              for pos in pos_str:
87                  if pos.isalnum():
88                      try:
89                          pos_set.add(pos_cond[pos].lower())
90                      except KeyError as ke:
91                          pos_not_found.append(ke.args[0])
```

```
92
93              # retrieve all the keywords matched in the lemmatized
    sentence
94              matched_kw = re.findall(kw_string_sep, lm_sentence,
    flags=re.IGNORECASE)
95              # convert keywords to lower case and get unique values
96              matched_kw_set = set(keyw.lower() for keyw in
    matched_kw)
97              sent_length = len(sentence.text.strip().split())
98
99              # retrieve ids of identified keywords
100             keywords_ids = return_key_ids(passage_kw_list,
    matched_kw_set)
101             # insert into database
102             if len(matched_kw_set) > 0 and len(keywords_ids) > 0:
103                 insert_dict = {"_id": index_st,
104                                "sentence": sentence.text,
105                                "keywords_sys_ids": keywords_ids,
106                                "sentence_length": sent_length,
107                                "passage_id": chap_choice,
108                                "pos_present": sorted(pos_set)
109                                }
110
111                 try:
112                     client[db_name][sentences_collection].
    insert_one(insert_dict)
113                     index_st += 1
114                 except pymongo.errors.DuplicateKeyError:
```

```
115                         print("Sentence %s already in database" %
    sentence.text)
116                 else:
117                     print("Sentence %s did not have any of the
    keywords in it" % sentence.text)
118
119             print("Part of speech tags matches not identified: \n")
120             if len(pos_not_found) > 1:
121                 print(set(pos_not_found))
122
123         else:
124             print(
125                 "No key words available for the given passage. Please
    generate key words first and then run this module again.")
126     else:
127         sentences = client[db_name][sentences_collection].find(
128             {'passage_id': int(chap_choice)},
129             {'sentence': 1, '_id': 1})
130         sentences_list = list(sentences)
131         print("\n System sentences have already been generated for
    this passage. They are: ")
132         for sent in sentences_list:
133             print(sent['sentence'])
134
135
136 if __name__ == "__main__":
137     main()
```

customStanza.py

```
import configparser


from nltk import Tree
from stanza.pipeline.core import Pipeline
from collections import defaultdict


class Stanza:
    def __init__(self):
        config = configparser.ConfigParser()
        try:
            config.read('../config.ini')
            tools_path = config['FILE_PATHS']['tools']
        except KeyError:
            config.read('config.ini')
            tools_path = config['FILE_PATHS']['tools']
        self.nlp = Pipeline(lang='en', logging_level='WARN',
    processors='tokenize,pos,lemma,ner,depparse, constituency',
                            model_dir=tools_path)

    def word_tokenize(self, sentence):
        doc = self.nlp(sentence)
        return [token.text for sent in doc.sentences for token in sent
    .tokens]

    def pos(self, sentence):
        doc = self.nlp(sentence)
```

```
26          return [word.xpos for sent in doc.sentences for word in sent.
    words]
27
28      def ner(self, sentence):
29          doc = self.nlp(sentence)
30          return [ent.type for sent in doc.sentences for ent in sent.
    ents]
31
32      def parse(self, sentence):
33          doc = self.nlp(sentence)
34          return [sent.deptree for sent in doc.sentences]
35
36      def constituency_parse(self, sentence):
37          doc = self.nlp(sentence)
38          return doc[0].constituency
39
40      def lemmatize_sentence(self, sentence):
41          doc = self.nlp(sentence)
42          lm_sent = " ".join([word.lemma for sent in doc.sentences for
    word in sent.words])
43          return lm_sent
```

import_tier_words.py

```
1 # This file imports .txt files with tier word lists to the database.
2 # Files are expected to have tier words with one word per line
3 # User is asked if the words are tier 2 or tier 3 words
4 # User is asked for username inserting the words
```

```
5
6  import configparser
7  import pyinputplus as pyip
8  import pymongo
9
10
11 class ImportTierWords:
12     def __init__(self):
13         self.client = pymongo.MongoClient("mongodb://localhost:27017/"
       )
14         config = configparser.ConfigParser()
15         config.read('../config.ini')
16         self.tier_words_path = config['FILE_PATHS']['tier_words_path']
17         self.db_name = "sence"  # name of database
18         self.tier_words_collection = "tier_words"
19         self.word_tier = 3
20         self.inserted_by = "system"
21         self.grade = "4"
22
23     def insert_tier_words(self):
24         # get the file name
25         is_success = False
26         while is_success is False:
27             tier_file_name = pyip.inputStr("Please enter filename of
       file: ")
28             file_name_with_path = self.tier_words_path+"/"+
       tier_file_name
29             print(file_name_with_path)
```

```
30            try:
31                file = open(file_name_with_path, 'r')
32                lines = file.readlines()
33                is_success = True
34                self.grade = pyip.inputStr("Please enter grade level
    of words. NA if not available.: ")
35                file.close()
36            except FileNotFoundError:
37                print("File name %s does not exist. Please try again.
    " % file_name_with_path)
38
39        num_docs = self.client[self.db_name][self.
    tier_words_collection].count_documents({})
40        index_st = 1 if(num_docs == 0) else (num_docs+1)
41        for line in lines:
42            try:
43                insert_dict = {"_id": index_st,
44                               "word": line.strip().lower(),
45                               "tier": self.word_tier,
46                               "grade": self.grade,
47                               "inserted_by": self.inserted_by
48                               }
49                self.client[self.db_name][self.tier_words_collection].
    insert_one(insert_dict)
50                index_st += 1
51
52            except pymongo.errors.DuplicateKeyError:
```

```
53                 print("Word %s is already in database. Skipping." %
       line.strip())
54
55
56 if __name__ == "__main__":
57     tier_words = ImportTierWords()
58     tier_words.insert_tier_words()
```

insertDefKeywordstoMongoDB.py

```
1 # This file takes a chapter from mongoDB and does the following
2 # i. Creates default keywords
3 # inserts them in MongoDB looking for exceptions and handles them
      appropriately
4 import logging
5 from collections import Counter
6 from string import punctuation
7 import pyinputplus as pyip
8 import pymongo
9 import spacy
10
11 from customStanza import Stanza
12 nlp = spacy.load("en_core_web_sm", disable=['parser', 'ner'])  # only
      load pos tagger
13 client = pymongo.MongoClient("mongodb://localhost:27017/")
14 db_name = "sence"  # name of database
15 passage_collection = "passage"  # name of passage collection
16 keywords_collection = "keywords"  # name of keywords collection
```

```
17 sNLP = Stanza()
18
19
20 def get_hotwords(text):
21     result = []
22     pos_tag = ['PROPN', 'ADJ', 'NOUN', 'VERB', 'ADV']  # proper noun,
   adjective, noun, verb, adverb
23     lesson = nlp(text.lower())
24     for token in lesson:
25         if token.text in nlp.Defaults.stop_words or token.text in
   punctuation:
26             continue
27         if token.pos_ in pos_tag:
28             result.append(token.text)
29     return result
30
31
32 def get_spacy_keywords(text):
33     output = (get_hotwords(text))
34     most_common_list = Counter(output).most_common(10)  # prints the
   top 10 most common words with frequencies
35     keywords = (list(zip(*most_common_list))[0])  # retrieves only
   words without frequencies numbers
36     return keywords
37
38
39 def main():
40     # find title, grade and id of all chapters in db
```

```
41      db_res = client[db_name][passage_collection].find({}, {"title": 1,
        "grade": 1})
42      print("{:<12} {:<60} {:<10}".format('Index number', 'Title', '
        Grade'))
43      for item in db_res:
44          print("{:<12} {:<60} {:<10}".format(item['_id'], item['title'
        ], item['grade']))
45      print("Enter the Index number of the chapter you want to create
        default keyword list for today: ")
46      chap_choice = pyip.inputNum()
47
48      num_docs_passage = client[db_name][keywords_collection].
        count_documents({'passage_num': int(chap_choice)})
49      num_docs_collection = client[db_name][keywords_collection].
        count_documents({})
50
51      passage = client[db_name][passage_collection].find({'_id':
        chap_choice})
52      chapter = passage[0]['text']
53      if num_docs_passage > 0:
54          insertdicts = []
55          passage_kw = client[db_name][keywords_collection].find(
56              {'passage_num': {'$in': [int(chap_choice)]}, 'inserted_by'
        : 'system'},
57              {'keyword': 1, '_id': 1})
58          print("\nSystem Keywords for this passage are: ")
59          kw_str = ' '.join(item['keyword'] for item in passage_kw)
60          print("%s\n\n" % kw_str)
```

```
61          passage_others_kw = client[db_name][keywords_collection].find(
62              {'passage_num': {'$in': [int(chap_choice)]}, 'inserted_by'
    : {'$ne': 'system'}}, {'keyword': 1, '_id': 1})
63          passage_others_kw = list(passage_others_kw)
64          if len(list(passage_others_kw)) > 0:
65              print("Others have added the following keywords: ")
66              kw_others_str = ' '.join(item['keyword'] for item in
    passage_others_kw)
67              print("%s\n\n" % kw_others_str)
68          yesno = pyip.inputYesNo(
69              "Would you like to input other custom keywords? ")
70          if yesno == 'yes':
71              inputkeywords = pyip.inputStr("Please input your keywords
    separated by a space. ")
72              custkeywords = inputkeywords.split()
73              inpusername = pyip.inputStr("Please enter your username: "
    )
74              for custword in custkeywords:
75                  if custword in chapter:
76                      insertdicts.append({"keyword": custword.strip(), "
    passage_num": chap_choice,
77                                          "inserted_by": inpusername.
    strip()})
78                  else:
79                      print("Keyword %s not in text, and will therefore
    be skipped. \n" % custword)
80
81      else:
```

```
82          insertdicts = []
83
84          lemma_chapter = sNLP.lemmatize_sentence(chapter)
85          # get spacy keywords
86          spacy_keywords = list(get_spacy_keywords(lemma_chapter))
87          print("Keywords identified for this chapter are: %s" % (' '.
    join(spacy_keywords)))
88          # insert each keyword into mongodb
89          for kw in spacy_keywords:
90              insertdicts.append({"keyword": kw, "passage_num":
    chap_choice, "inserted_by": "system"})
91
92      # insert or update depending on if the keyword is already present
    in the collection
93      index_st = num_docs_collection + 1
94      for insert_dict in insertdicts:
95          try:
96              client[db_name][keywords_collection].insert_one(
97                  {"_id": index_st, "keyword": insert_dict["keyword"], "
    passage_num": [insert_dict["passage_num"]],
98                   "inserted_by": insert_dict["inserted_by"]})
99              index_st += 1  # increase id by 1 after insertion
100
101         except pymongo.errors.DuplicateKeyError:
102             client[db_name][keywords_collection].update_one({"keyword"
    : insert_dict["keyword"]},
103                                                             {'
    $addToSet': {"passage_num": insert_dict["passage_num"]}})
```

```
104
105
106
107 if __name__ == "__main__":
108     main()
```

highFrequencyWordsMethodsComparator.py

```
1  # this script does the following
2  # grabs text from the text file with title
3  # gets grade of lesson
4  # stores in mongoDB passage collection
5
6  import pyinputplus as pyip
7  import pymongo
8  import spacy
9
10 client = pymongo.MongoClient("mongodb://localhost:27017/")
11 db_name = "sence"  # name of database
12 passage_collection = "passage"  # name of passage collection
13 num_docs_cursor = client[db_name][passage_collection].find({}, {"_id":
       1}).sort("_id", pymongo.DESCENDING).limit(1)
14 # check to see if any documents are already present in the collection
15 # if none, start the index at 1
16 if client[db_name][passage_collection].count_documents({}) == 0:
17     index_st = 1
18 else:
19     # else start the index at one from the prev entry's index
```

```
20      num_docs = num_docs_cursor.next()
21      index_st = num_docs['_id'] + 1
22
23
24  # return count of number of words ignoring punctuation
25  def word_count(sentences):
26      words = 0
27      for sentence in sentences:
28          tokens = [tok.text for tok in sentence if not tok.is_punct]
29          words += len(tokens)
30      return words
31
32
33  def max_sentence_length(sentences):
34      max_l = 0
35      for sentence in sentences:
36          num_words = len([tok.text for tok in sentence if not tok.
        is_punct])
37          if num_words > max_l:
38              max_l = num_words
39      return max_l
40
41
42  yes_no_choice = pyip.inputYesNo("Would you like to import a lesson
        today? ")
43  if yes_no_choice == 'yes':
44      lesson_title = pyip.inputStr("Please enter the title of the text:
        ")
```

```
45      lesson_grade = pyip.inputNum("Please enter the lesson grade: ")
46      input_form = pyip.inputMenu(['File path', 'Enter text here'],
47                                  prompt="Please enter the number of the
        way you would like to import the lesson: \n ",
48                                  numbered=True)
49      if input_form == "File path":
50          text_blob = ''
51          file_ip_yesno = "yes"
52          while file_ip_yesno == "yes":
53              name = pyip.inputStr("Enter file name with the full path (
        eg: C:\path_to_file\documentname.extension): ")
54              filename = name.strip()
55              try:
56                  with open(filename, 'r') as f:
57                      content = f.read()
58                  for line in content:
59                      text_blob += line.replace("\n", "")
60              except FileNotFoundError as e:
61                  file_ip_yesno = pyip.inputYesNo(
62                      "We did not find a file at that location. Would
        you like to try again? Please input yes or no: ")
63
64      else:
65          text_blob = input("Please enter the lesson text here:\n")
66      if text_blob:
67          clean_text = text_blob.strip()
68          nlp = spacy.load('en_core_web_sm')
69          doc = nlp(clean_text)
```

```
70          sents = list(doc.sents)
71          word_c = word_count(sents)
72          sentence_count = len(sents)
73          average_sent_length = round(float(word_c / sentence_count), 1)
74          max_length = round(max_sentence_length(sents), 1)
75          media = pyip.inputMenu(['textbook', 'storybook', 'tvshow', ],
76                                 prompt="Please enter the number next to
    the type of media of the text you have entered: \n ",
77                                 numbered=True)
78          insert_dict = {"_id": index_st,
79                         "grade": lesson_grade,
80                         "text": clean_text,
81                         "title": lesson_title.strip(),
82                         "media": media.strip(),
83                         "total_words": word_c,
84                         "total_sentences": sentence_count,
85                         "avg_sentence_length": average_sent_length,
86                         "max_sentence_length": max_length
87                         }
88          try:
89              client[db_name][passage_collection].insert_one(
90                  insert_dict)
91              index_st += 1
92          except Exception as e:
93              print("Exception occurred while inserting: %s" % e)
```

highFrequencyWordsMethodsComparator.py

```
import textacy
from keybert import KeyBERT
import pyinputplus as pyip
from nltk.tokenize import RegexpTokenizer
from sentences.customStanza import Stanza
from rake_nltk import Rake
from nltk.corpus import stopwords
import spacy
from string import punctuation
import pymongo
from textacy.extract import keyterms as kt
from textRankforKeyterm import TextRank4Keyword
from yake import KeywordExtractor


sp_nlp = spacy.load("en_core_web_sm", disable=['parser', 'ner'])  # only load pos tagger
client = pymongo.MongoClient("mongodb://localhost:27017/")
db_name = "sence"  # name of database
passage_collection = "passage"  # name of passage collection
tokenizer = RegexpTokenizer(r'\w+')
keywords_collection = "keywords"  # name of keywords collection
rake_nltk_var = Rake()
# yake instance with max 2 n-grams, some duplication (0.1 is none, 0.9 is 'allow')  and top 10 keywords
kw_extractor_yake = KeywordExtractor(lan='en', n=1, dedupLim=0.5, top=10, features=None)
kw_extractor_bert = KeyBERT()
```

```
26 tr4w = TextRank4Keyword()
27
28 stop_words = set(stopwords.words('english'))
29
30
31 def get_hotwords(text):
32     sNLP = Stanza()
33     kw_pos_dict = {}
34     pos_tag = ['PROPN', 'ADJ', 'NOUN', 'VERB', 'ADV']  # proper noun,
    adjective, noun, verb, adverb
35
36     lesson = sNLP.nlp(text.lower())
37     for sent in lesson.sentences:
38         for token in sent.words:
39             if token.text in punctuation:
40                 continue
41             if token.upos in pos_tag:
42                 kw_pos_dict[token.text] = token.upos
43     return kw_pos_dict
44
45
46 def main():
47     # find title, grade and id of all chapters in db
48     db_res = client[db_name][passage_collection].find({}, {"title": 1,
     "grade": 1})
49     print("{:<12} {:<60} {:<10}".format('Index number', 'Title', '
    Grade'))
50     for item in db_res:
```

```
51          print("{:<12} {:<60} {:<10}".format(item['_id'], item['title'
    ], item['grade']))
52      print("Enter the Index number of the chapter you want to view
    keyword comparisons for today: ")
53
54      chap_choice = pyip.inputNum()
55      passage = client[db_name][passage_collection].find({'_id':
    chap_choice})
56      sence_words = client[db_name][keywords_collection].find({'
    passage_num':int(chap_choice)},{'_id': 1, 'keyword':1})
57      sence_words_list = list(sence_words)
58      chapter = passage[0]['text']
59      sNLP = Stanza()
60      lemma_chapter = sNLP.lemmatize_sentence(chapter)
61      kw_pos_dict = get_hotwords(lemma_chapter)
62      # get high frequency words from rake
63      rake_nltk_var.extract_keywords_from_text(lemma_chapter)
64      rake_words = rake_nltk_var.get_ranked_phrases()[:10]
65      # get high freq words using yake
66      yake_extracts = kw_extractor_yake.extract_keywords(lemma_chapter)
67      yake_keywords = [x for x, y in yake_extracts]
68      bert_extracts = kw_extractor_bert.extract_keywords(lemma_chapter,
    keyphrase_ngram_range=(1, 1),
69                                                         stop_words='
    english', top_n=10)
70      bert_keywords = [x for x, y in bert_extracts]
71
72      # textacy keyeterms
```

```
73      kt_extracts = kt.textrank(textacy.make_spacy_doc(chapter, lang="
        en_core_web_sm"), normalize='lemma', window_size=2, position_bias=
        True, topn=10)
74      kt_words = [x for x, y in kt_extracts]
75
76      # textacy sgrank
77      sgrank_extracts = kt.sgrank(textacy.make_spacy_doc(chapter, lang="
        en_core_web_sm"), normalize='lemma', ngrams=1, window_size=2, topn
        =10)
78      sgrank_words = [x for x, y in sgrank_extracts]
79
80      # textRank
81      tr4w.analyze(lemma_chapter, candidate_pos=['NOUN', 'PROPN', 'VERB'
        , 'ADV', 'ADJ'], window_size=4, lower=False)
82      textRank_words = tr4w.get_keywords(number=8)
83
84
85      counters = {'NOUN': 0, 'PROPN': 0, 'VERB': 0, 'ADV': 0, 'ADJ': 0}
86      for word in yake_keywords:
87          try:
88              counters[kw_pos_dict[word.lower()]] += 1
89          except KeyError:
90              print("Cannot find POS for ", word)
91
92      yake_keywords = sorted(yake_keywords, key=str.lower)
93      print("Yake: ", *yake_keywords, sep=", ")
94      print("Yake counters: ", counters.values())
95
```

```
    rake_keywords = sorted(rake_words, key=str.lower)
    print("Rake: ", *rake_keywords, sep=", ")

    counters = {'NOUN': 0, 'PROPN': 0, 'VERB': 0, 'ADV': 0, 'ADJ': 0}
    for word in bert_keywords:
        try:
            counters[kw_pos_dict[word.lower()]] += 1
        except KeyError:
            print("Cannot find POS for ", word)

    bert_keywords = sorted(bert_keywords, key=str.lower)
    print("Bert: ", *bert_keywords, sep=", ")
    print("Bert counters: ", counters.values())


    counters = {'NOUN': 0, 'PROPN': 0, 'VERB': 0, 'ADV': 0, 'ADJ': 0}
    for word in sgrank_words:
        try:
            counters[kw_pos_dict[word.lower()]] += 1
        except KeyError:
            print("Cannot find POS for ", word)

    sgrank_words = sorted(sgrank_words, key=str.lower)
    print("SgRank: ", *sgrank_words, sep=", ")
    print("Sgrank counters: ", counters.values())

    counters = {'NOUN': 0, 'PROPN': 0, 'VERB': 0, 'ADV': 0, 'ADJ': 0}
    for word in textRank_words:
```

```
124         try:
125             counters[kw_pos_dict[word.lower()]] += 1
126         except KeyError:
127             print("Cannot find POS for ", word)
128 
129     textRank_words = sorted(textRank_words, key=str.lower)
130     print("TextRank: ", *textRank_words, sep=", ")
131     print("TextRank counters: ", counters.values())
132 
133     counters = {'NOUN': 0, 'PROPN': 0, 'VERB': 0, 'ADV': 0, 'ADJ': 0}
134     sence_kw = []
135     for word in sence_words_list:
136         try:
137             pos = kw_pos_dict[word['keyword'].lower()]
138             counters[pos] += 1
139 
140         except KeyError:
141             print("Cannot find POS for ", word)
142         sence_kw.append(word['keyword'])
143 
144     sence_kw = sorted(sence_kw, key=str.lower)
145     print("SENCE: ", *sence_kw, sep=", ")
146     print("SENCE counters: ", counters.values())
147 
148 if __name__ == "__main__":
149     main()
```

PickSentencesOtherPassages.py

```
1  # This script is a helper script to storeSentencesForLessonsBySort.
       Given a list of keywords, it picks sentences from
2  # passages other than the one being taught. The purpose of this is to
       test the student's knowledge fof the word in a
3  # context they might not have seen before.
4
5  from SentenceSorterByFilter import *
6  from toolbelt import SentenceSimilarityMeasures
7
8
9  class PickSentencesOtherPassages:
10     db_name = "sence"  # name of database
11     sentence_collection = "sentences"  # name of passage collection
12     passage_collection = "passage"
13     keywords_collection = "keywords"  # name of keywords collection
14     modified_sentences_collection = "modified_sentences"
15     tier_words_collection = "tier_words"
16     sentences_for_passage_collection = "sentences_for_passage"
17     sentences_hash = {}
18     sentences_reverse_hash = {}
19
20     def __init__(self):
21         self.client = pymongo.MongoClient("mongodb://localhost:27017/"
       )
22         self.username = "system"
23
24     # given a sentence return the id from the sentence hash
```

```
25      @staticmethod
26      def pick_sent_given_id(sentence):
27          sentence = sentence.strip()
28          return PickSentencesOtherPassages.sentences_reverse_hash[
    sentence]
29
30
31      def pick_other_sentences(self, keyword_ids, passage_id, sort_type)
    :
32          keywords = self.client[self.db_name][self.keywords_collection
    ].find(
33              {'_id': {'$in': keyword_ids}}, {'keyword': 1})
34          kws = list(keywords)
35          keyword_str = ''
36          for item in kws:
37              keyword_str += " " + item['keyword']
38          sentence_sort_by_filter = SentenceSorterByFilter()
39          toolbelt = SentenceSimilarityMeasures()
40          # select sentences from database that have matching keywords
    from passages that are not the current lesson
41          all_sentences = self.client[self.db_name][self.
    sentence_collection].find(
42              {'keywords_sys_ids': {'$in': keyword_ids},
43               'passage_id': {'$ne': int(passage_id)}})
44          all_sentences_list = list(all_sentences)
45          for sent in all_sentences_list:
46              self.sentences_hash.update({sent['_id']: sent['sentence'].
    strip()})
```

```
47              self.sentences_reverse_hash.update({sent['sentence'].strip
    (): sent['_id']})
48          if sort_type == 'Sentence Length':
49              # display sentences by length
50              sents = sentence_sort_by_filter.
    pick_candidate_sentences_length(all_sentences_list)
51
52          elif sort_type == 'Keyword in Subject of sentence':
53              sents = sentence_sort_by_filter.
    pick_candidate_sentences_subject(keyword_ids, all_sentences_list)
54          elif sort_type == 'Number of Tier 2 and 3 words in sentence':
55              sents = sentence_sort_by_filter.
    pick_candidate_sentences_tier_words(all_sentences_list,
56              self.db_name, self.tier_words_collection, self.
    sentences_hash)
57          else:
58              sents = sentence_sort_by_filter.
    pick_candidate_sentences_num_keywords(keyword_ids, self.
    sentences_hash, all_sentences_list)
59
60          simp_med_complex_sents = {}
61          if len(sents['simple']) > 1:
62              sent = toolbelt.calculate_spacy_sim(keyword_str, list(item
    ['sentence'] for item in sents['simple']), 1)[0]
63              simp_med_complex_sents['simple'] = [{'_id': self.
    pick_sent_given_id(sent), 'sentence': sent}]
64          elif len(sents['simple']) == 1:
65              sent = sents['simple'][0]['sentence']
```

```
66              simp_med_complex_sents['simple'] = [{'_id': self.
    pick_sent_given_id(sent), 'sentence': sent}]
67          else:
68              simp_med_complex_sents['simple'] = []
69          if len(sents['medium']) > 1:
70              sent = toolbelt.calculate_spacy_sim(keyword_str, list(item
    ['sentence'] for item in sents['medium']), 1)[0]
71              simp_med_complex_sents['medium'] = [{'_id': self.
    pick_sent_given_id(sent), 'sentence': sent}]
72          elif len(sents['medium']) == 1:
73              sent = sents['medium'][0]['sentence']
74              simp_med_complex_sents['medium'] = [{'_id': self.
    pick_sent_given_id(sent), 'sentence': sent}]
75          else:
76              simp_med_complex_sents['medium']  = []
77
78          if len(sents['complex']) > 1:
79              sent = toolbelt.calculate_spacy_sim(keyword_str, list(item
    ['sentence'] for item in sents['complex']), 1)[0]
80              simp_med_complex_sents['complex'] = [{'_id': self.
    pick_sent_given_id(sent), 'sentence': sent}]
81          elif len(sents['complex']) == 1:
82              sent = sents['complex'][0]['sentence']
83              simp_med_complex_sents['complex'] = [{'_id': self.
    pick_sent_given_id(sent), 'sentence': sent}]
84          else:
85              simp_med_complex_sents['complex'] = []
86          return simp_med_complex_sents
```

SentenceSorterByFilter.py

```
# This is a helper method to storeSentencesForLessonsBySort and PickSentencesOtherPassages

import configparser
import math
import pyinputplus as pyip
import pymongo
import random
import re
import spacy


# this method returns verbs and adverbs spans from the sentence.
# It accepts a tokenized sentence and returns a list of spans
def return_verb_chunks(sentence_tokenized):
    verb_chunks = []
    for token in sentence_tokenized:
        if token.pos_ == "VERB" or token.pos_ == "ADV":
            span = sentence_tokenized[token.i:token.i + 1]
            verb_chunks.append(span)
    return verb_chunks

class SentenceSorterByFilter:
    def __init__(self):
        self.client = pymongo.MongoClient("mongodb://localhost:27017/")
        config = configparser.ConfigParser()
```

```
26          try:
27              config.read('../config.ini')
28          except KeyError:
29              config.read('config.ini')
30          self.num_sentences_to_return_passage = config['SENTENCES']['
    num_sentences_to_return_passage']
31          self.simple_sent_dependency_tags = config['SENTENCES']['
    sentence_location_filter_simple'].split()
32          self.medium_sent_dependency_tags = config['SENTENCES']['
    sentence_location_filter_medium'].split()
33
34      # this method picks 'x' number of sentences from shortest to
    longest
35      # divide number of sentences into three buckets - simple, medium,
    complex and upper bound of length for each
36      # divide sentences into each bucket and return
37      def pick_candidate_sentences_length(self, inp_list):
38          # find the max length in sentences
39          max_length = 0
40          for item in inp_list:
41              if item['sentence_length'] > max_length:
42                  max_length = item['sentence_length']
43          # find the three buckets
44          simple = math.ceil(max_length / 3) + 2
45          medium = (simple - 2) * 2
46          simp_sentences = []
47          med_sentences = []
48          complex_sentences = []
```

```
49          insufficient_flag = True
50          insufficient_tries = 0
51          while insufficient_flag:
52              for sent in inp_list:
53                  sentence = sent['sentence'].strip()
54                  if sent['sentence_length'] <= simple:
55                      simp_sentences.append({'_id': sent['_id'], '
    sentence': sentence})
56                  elif simple < sent['sentence_length'] <= medium:
57                      med_sentences.append({'_id': sent['_id'], '
    sentence': sentence})
58                  else:
59                      complex_sentences.append({'_id': sent['_id'], '
    sentence': sentence})
60              if ((len(simp_sentences) <= int(self.
    num_sentences_to_return_passage) or len(
61                      med_sentences) <= int(self.
    num_sentences_to_return_passage)) and insufficient_tries < int(
62                      self.num_sentences_to_return_passage)):
63                  simple += 1  # increase the simple and medium
    thresholds to get more sentences in each of those buckets
64                  medium += 1
65                  insufficient_tries += 1  # to avoid an endless loop in
     case there are no sentences to be found
66                  # reset the lists
67                  simp_sentences = []
68                  med_sentences = []
69                  complex_sentences = []
```

```
70              else:
71                  insufficient_flag = False
72
73          sentences = {'simple': simp_sentences, 'medium': med_sentences
    , 'complex': complex_sentences}
74          return sentences
75
76      # this method picks sentences where the keyword is in the subject
    of the sentence and bins resulting sentences into
77      # simple, med and complex based on where the word appears in the
    subject
78      def pick_candidate_sentences_subject(self, kw_list, inp_list):
79          nlp = spacy.load('en_core_web_lg')
80
81          simp_sentences = []
82          med_sentences = []
83          complex_sentences = []
84          size_simp = 0
85          size_med = 0
86          size_comp = 0
87          for item in inp_list:
88              sent = item['sentence']
89              doc = nlp(sent)
90              noun_chunks = list(doc.noun_chunks)
91              non_noun_chunks = return_verb_chunks(doc)
92              chunks = non_noun_chunks + noun_chunks
93              sent_appended = False  # reset the flag
94              # separate the sentence into noun chunks
```

```
95          for chunk in chunks:
96              if sent_appended:  # if sentence has already been added to list of sentences, skip to next sentence
97                  break
98
99              if chunk.lemma_ in kw_list or (chunk.root.dep_ != 'ROOT' and chunk.root.lemma_ in kw_list) :
100                 if (
101                         chunk.root.dep_ in self.simple_sent_dependency_tags) and size_simp <= size_med and size_simp <= size_comp:
102                     simp_sentences.append({'_id': item['_id'], 'sentence': sent})
103                     sent_appended = True
104                     size_simp += size_simp
105                 elif (
106                         chunk.root.dep_ in self.medium_sent_dependency_tags) and size_med <= size_simp and size_med <= size_comp:
107                     med_sentences.append({'_id': item['_id'], 'sentence': sent})
108                     sent_appended = True
109                     size_med += size_med
110                 elif size_comp <= size_simp and size_comp <= size_med:
111                     complex_sentences.append({'_id': item['_id'], 'sentence': sent})
112                     sent_appended = True
```

```
113                         size_comp += size_comp
114                     break
115
116         sentences = {'simple': simp_sentences, 'medium': med_sentences
    , 'complex': complex_sentences}
117         return sentences
118
119     # this method picks sentence based on number of identified
    keywords in the sentence
120     def pick_candidate_sentences_num_keywords(self, kw_ids,
    sentences_hash, all_sentences_list):
121         sentences_by_num_kw = {}
122
123         for sent in all_sentences_list:
124             # find intersection of kw_picked_ids and keyword_sys_ids
    of sentence
125             kw_in_sent = set(sent['keywords_sys_ids']) & set(kw_ids)
126             # if key (number of keywords in sentence) is not in dict,
    add it
127             if len(kw_in_sent) not in sentences_by_num_kw.keys():
128                 sentences_by_num_kw[len(kw_in_sent)] = [{'kw_id':
    kw_in_sent, 'sent_id': [sent['_id']]}]
129             else:
130                 # find the (keywords list) to append the sentence id
    to it
131                 kw_found = False
132                 for item in sentences_by_num_kw[len(kw_in_sent)]:
133                     if item['kw_id'] == kw_in_sent:
```

```
                        kw_index = sentences_by_num_kw[len(kw_in_sent)
].index(item)
                        sentences_by_num_kw[len(kw_in_sent)][kw_index
]['sent_id'].append(sent['_id'])
                        kw_found = True
                        break
                # if keywords list is not in dict, add it to the dict
                if kw_found is False:
                    sentences_by_num_kw[len(kw_in_sent)].append({'
kw_id': kw_in_sent, 'sent_id': [sent['_id']]})

    # define size of simple, medium, complex buckets
    num_kw = len(kw_ids)
    bucket_size = math.floor(num_kw / 3)
    simp_size = bucket_size
    med_size = bucket_size + 1
    # define the upper bound of the medium index; this is
sufficient to define bounds for simple and complex sentences
    med_index = simp_size + med_size

    simp_sentences = []
    med_sentences = []
    complex_sentences = []
    insufficient_flag = True
    insufficient_tries = 0
    while insufficient_flag:
        # Separate sentences into simple, complex and medium bins
depending on number of keywords in the sentence
```

```
157             for kw, kw_tuple in sentences_by_num_kw.items():
158                 if kw <= simp_size:
159                     for item in kw_tuple:
160                         sentences = item['sent_id']
161                         for sent in sentences:
162                             simp_sentences.append({'_id': sent, '
    sentence': sentences_hash[sent]})
163                 elif simp_size < kw < med_index:
164                     for item in kw_tuple:
165                         sentences = item['sent_id']
166                         for sent in sentences:
167                             med_sentences.append({'_id': sent, '
    sentence': sentences_hash[sent]})
168                 else:
169                     for item in kw_tuple:
170                         sentences = item['sent_id']
171                         for sent in sentences:
172                             complex_sentences.append({'_id': sent, '
    sentence': sentences_hash[sent]})
173             if ((len(simp_sentences) <= int(self.
    num_sentences_to_return_passage) or len(
174                     med_sentences) <= int(self.
    num_sentences_to_return_passage)) and
175                     insufficient_tries < int(self.
    num_sentences_to_return_passage)):
176                 simp_size += 1  # increase the simple and medium
    thresholds to get more sentences in each of those buckets
177                 med_size += 1
```

```
178                 insufficient_tries += 1  # to avoid an endless loop in
 case there are no sentences to be found
179                 # reset the lists
180                 simp_sentences = []
181                 med_sentences = []
182                 complex_sentences = []
183             else:
184                 insufficient_flag = False
185         all_sentences = {'simple': simp_sentences, 'medium':
med_sentences, 'complex': complex_sentences}
186         return all_sentences
187 
188     # this method generates tuples of sentences with (1...n) keywords
of sentences chosen at random
189     @staticmethod
190     def pick_candidate_sentences_random(kw_ids, inp_list, num_tuple):
191         list_sents = []
192         if num_tuple == 1:
193             for item in inp_list:
194                 rand_sent = random.choice(item['sent_id'])
195                 list_sents.append({'kw_id': item['kw_id'], 'sent_id':
rand_sent})
196         else:
197             # get all available tuples of keywords
198             all_kw_combos = [tuple(item['kw_id']) for item in inp_list
]
199             picked_kws = []
200             for kw_id in kw_ids:
```

```
201                 # for each kw_id pick a tuple that has that id in it
202                 kw_picked_tuple = list(filter(lambda x: kw_id in x,
    all_kw_combos))
203                 # now pick one of those by random and add to list
204                 picked_kws.append(random.choice(kw_picked_tuple))
205             unique_tuples = list(set(picked_kws))
206             # now that we have a list of random keyword tuples, we
    need to pick  sentences based on those keywords
207             for item in unique_tuples:
208                 for pair in inp_list:
209                     if sorted(list(pair['kw_id'])) == sorted(list(item
    )):
210                         rand_sent = random.choice(pair['sent_id'])
211                         list_sents.append({'kw_id': pair['kw_id'], '
    sent_id': rand_sent})
212                         break
213
214         return list_sents
215
216     # this method returns sentences based on number of tier 2 and 3
    words present in the sentences
217     def pick_candidate_sentences_tier_words(self, inp_list, db_name,
    tier_words_collection, sentences_hash):
218         tier_two_words_set = set()
219         tier_three_words_set = set()
220         tier_words_other_set = set()
221         tier_word_sent_dict = {}
222         most_t_ids = 0
```

```
223         simp_sentences = []
224         med_sentences = []
225         complex_sentences = []
226         # remove stop words from sentence
227         nlp = spacy.load('en_core_web_sm')
228         for sent_item in inp_list:
229             sentence = sent_item['sentence']
230             nlp_sentence = nlp(sentence)
231             # remove all the stop words from the sentence (to reduce
    number of lookups) and lemmatize the word
232             filtered_tokens = [token.lemma_ for token in nlp_sentence
    if not token.is_stop]
233             filtered_tokens = [token.lower() for token in
    filtered_tokens]
234             # identify tier 2 and 3 words from condensed sentence in
    the database
235             tier_words = self.client[db_name][tier_words_collection].
    find(
236                 {'word': {'$in': filtered_tokens}})
237
238             tier_words = list(tier_words)
239             if tier_words and len(tier_words) > 0:
240                 for w in tier_words:
241                     if w['inserted_by'] == 'system':
242                         if w['tier'] == 2:
243                             tier_two_words_set.add(w['word'])
244                         elif w['tier'] == 3:
245                             tier_three_words_set.add(w['word'])
```

```
246                     else:
247                         tier_words_other_set.add(w['word'])
248                 t_ids = [x['_id'] for x in tier_words]
249                 # tier_words_set.update(t_words)
250                 tier_word_sent_dict[sent_item['_id']] = t_ids
251 
252         if tier_two_words_set:
253             t_words = " ".join(tier_two_words_set)
254             print("Tier 2 words in this passage are: ", t_words)
255         if tier_three_words_set:
256             t_words = " ".join(tier_three_words_set)
257             print("Tier 3 words in this passage are: ", t_words)
258         if tier_words_other_set:
259             t_words = " ".join(tier_words_other_set)
260             print("Others have added the following tier words: ",
    t_words)
261         addwords = pyip.inputYesNo("Would you like to add any more
    tier words based on this passage? ")
262         if addwords == 'yes':
263             username = pyip.inputStr("Please enter your username: ")
264             tier2addedwords = pyip.inputStr("Please add additional
    Tier 2 words separated by a space. ")
265             t2addedwords = tier2addedwords.split(" ") if (len(
    tier2addedwords) > 0) else None
266             tier3addedwords = pyip.inputStr("Please add additional
    Tier 3 words separated by a space. ", blank=True)
267             t3addedwords = tier3addedwords.split(" ") if (len(
    tier3addedwords) > 0) else None
```

```
268             if (t2addedwords and len(t2addedwords) > 0) or (
    t3addedwords and len(t3addedwords) > 0):
269                 all_added_words = []
270                 tw_add_hash = {}
271                 num_docs_cursor = self.client[db_name][
    tier_words_collection].find(
272                     {}, {"_id": 1}).sort("_id", pymongo.DESCENDING).
    limit(1)
273                 num_docs = num_docs_cursor.next()
274                 index_st = num_docs['_id'] + 1
275                 if t2addedwords and len(t2addedwords) > 0:
276                     for tw in t2addedwords:
277                         doc = nlp(tw)
278                         insert_w = doc[0].lemma_
279                         insert_dict = {"_id": index_st,
280                                        "word": insert_w.lower(),
281                                        "tier": 2,
282                                        "grade": "NA",
283                                        "inserted_by": username
284                                        }
285                         try:
286                             insert_id = self.client[db_name][
    tier_words_collection].insert_one(insert_dict)
287                             all_added_words.append(insert_w.lower())
288                             tw_add_hash[insert_w.lower()] = insert_id.
    inserted_id
289                             index_st = index_st + 1
290                         except pymongo.errors.DuplicateKeyError:
```

```
291                             print("Word %s is already in the database.
    " % tw)
292                 if t3addedwords and len(t3addedwords) > 0:
293                     for tw in t3addedwords:
294                         doc = nlp(tw)
295                         insert_w = doc[0].lemma_
296                         insert_dict = {"_id": index_st,
297                                        "word": insert_w.lower(),
298                                        "tier": 3,
299                                        "grade": "NA",
300                                        "inserted_by": username
301                                        }
302                         try:
303                             insert_id = self.client[db_name][
    tier_words_collection].insert_one(insert_dict)
304                             all_added_words.append(insert_w.lower())
305                             tw_add_hash[insert_w.lower()] = insert_id.
    inserted_id
306                             index_st = index_st + 1
307                         except pymongo.errors.DuplicateKeyError:
308                             print("Word %s is already in the database.
    " % tw)
309                 # recheck sentences to see if newly added words are
    present
310                 add_words_str = "|".join(all_added_words)
311                 for item in inp_list:
312                     sent = item['sentence']
313                     nlp_sentence = nlp(sent)
```

```
                    sent = " ".join([token.lemma_ for token in
nlp_sentence])
                    sent = sent.lower()
                    matches = re.findall(add_words_str, sent)
                    if matches and len(matches) > 0:
                        # add sentence to tier_word_sent_dict
                        for match in matches:
                            tw_id = tw_add_hash[match]
                            # if sent_id already exists, append tier
word id to existing dict
                            if item['_id'] in tier_word_sent_dict.keys
():
                                tier_word_sent_dict[item['_id']].
append(tw_id)
                            # else add new dict item
                            else:
                                tier_word_sent_dict[item['_id']] = [
tw_id]

    # check the max number of tier words in a sentence to define
bounds for simple, med and complex sentences
    for s_id, tw_ids in tier_word_sent_dict.items():
        if len(tw_ids) > most_t_ids:
            most_t_ids = len(tw_ids)
    # return sentences grouped as simple, medium and complex based
 on number of tier words in sentences
    # find the three buckets
    simple = math.floor(most_t_ids / 3)
```

```
335         medium = simple * 2
336         insufficient_flag = True
337         insufficient_tries = 0
338         while insufficient_flag:
339             for s_id, tw_ids in tier_word_sent_dict.items():
340                 tw_num = len(tw_ids)
341                 if tw_num <= simple:
342                     simp_sentences.append({'_id': s_id, 'sentence':
    sentences_hash[s_id]})
343                 elif simple < tw_num <= medium:
344                     med_sentences.append({'_id': s_id, 'sentence':
    sentences_hash[s_id]})
345                 else:
346                     complex_sentences.append({'_id': s_id, 'sentence':
     sentences_hash[s_id]})
347             if ((len(simp_sentences) <= int(self.
    num_sentences_to_return_passage) or len(
348                     med_sentences) <= int(self.
    num_sentences_to_return_passage)) and
349                     insufficient_tries < int(self.
    num_sentences_to_return_passage)):
350                 simple += 1  # increase the simple and medium
    thresholds to get more sentences in each of those buckets
351                 medium += 1
352                 insufficient_tries += 1  # to avoid an endless loop in
     case there are no sentences to be found
353                 # reset the lists
354                 simp_sentences = []
```

```
355                 med_sentences = []
356                 complex_sentences = []
357             else:
358                 insufficient_flag = False
359 
360         # show users tier words and see if they want to add more
361         # if yes, add to db
362         all_sentences = {'simple': simp_sentences, 'medium':
    med_sentences, 'complex': complex_sentences}
363         return all_sentences
```

showStudentResults.py

```
1 import pyinputplus as pyip
2 import pymongo
3 import tabulate
4 
5 
6 class ShowStudentResults:
7     db_name = "sence"  # name of database
8     student_response_collection = "student_response"
9 
10    def __init__(self):
11        self.client = pymongo.MongoClient("mongodb://localhost:27017/"
    )
12 
13    def show_results(self):
```

```
14          # find unique values of passage num, sort type and keywords
    from sentences_for_passage collection.
15          # we need unique values because there are multiple rows for
    simple, complex, medium sentences for each (passage_num, sort_type
    , keywords) set
16          db_res = self.client[ShowStudentResults.db_name][
    ShowStudentResults.student_response_collection].aggregate([
17              {"$group": {"_id": {"passage": "$passage", "student_name":
     "$student_name", "keywords": "$keywords",
18                                  "time_assessed": "$time_assessed", "
    sort_type": "$sort_type"}}}
19          ])
20          db_res_list = list(db_res)
21          table = []
22          counter = 1
23          table_headers = ['Index \n Number', 'Passage title', 'Keywords
    ', 'Filter type', 'Student name',
24                           'Time of assessment']
25          for item in db_res_list:
26              result_details = item['_id']
27              table.append([counter, result_details['passage'], ", ".
    join(result_details['keywords']),
28                            result_details['sort_type'], result_details[
    'student_name'], result_details['time_assessed']])
29              counter += 1
30          print(tabulate.tabulate(table, table_headers))
31          input_choice = pyip.inputNum("Please enter the index number of
     the results you will like to see: ")
```

```
32          db_filter = db_res_list[input_choice - 1]['_id']
33          student_res_db = self.client[ShowStudentResults.db_name][
    ShowStudentResults.student_response_collection].find(
34              {'passage': db_filter['passage'], 'keywords': {'$in':
    db_filter['keywords']},
35               'sort_type': db_filter['sort_type'],
36               'student_name': db_filter['student_name'], 'time_assessed
    ': db_filter['time_assessed']})
37          student_results_list = list(student_res_db)
38          print("Filter criteria used:")
39          print("Passage: ", db_filter['passage'])
40          print("Keywords: ", ", ".join(db_filter['keywords']))
41          print("Student name: ", db_filter['student_name'])
42          print("Time of assessment: ", db_filter['time_assessed'])
43          num_correct = 0
44          for item in student_results_list:
45              print("\nOriginal sentence: ", item['sentence_original'])
46              print("Student response: ", item['sentence_response'])
47              if item['correct']:
48                  correct = "CORRECT."
49                  num_correct += 1
50              else:
51                  correct = "WRONG."
52              print("Student response is ", correct)
53          print("\n Student answered %d of %d sentences correctly." % (
    num_correct, len(student_results_list)))
54          print("The percentage of sentences answered correctly is: %.2f
    %%" % (num_correct / len(student_results_list)*100))
```

```
55
56
57 if __name__ == "__main__":
58     student_results = ShowStudentResults()
59     student_results.show_results()
```

storeSentencesForLessonBySort.py

```
1 # this script does the following
2 # based on keywords, picks simple, medium and complex sentences based
      on the following criteria
3 # a. length of sentences
4 # b. number of keywords in sentence
5 # c. parts of speech in the sentence
6 import numpy as np
7 from PickSentencesOtherPassages import *
8
9
10 class PickSentences:
11     db_name = "sence"  # name of database
12     sentence_collection = "sentences"  # name of passage collection
13     passage_collection = "passage"
14     keywords_collection = "keywords"  # name of keywords collection
15     modified_sentences_collection = "modified_sentences"
16     tier_words_collection = "tier_words"
17     sentences_for_passage_collection = "sentences_for_passage"
18     sentences_hash = {}
19     keywords_hash = {}
```

```
20
21      def __init__(self):
22          self.client = pymongo.MongoClient("mongodb://localhost:27017/"
        )
23          self.username = "system"
24          self.chap_choice = ''
25          self.kw_picked_ids = []
26          self.kw_list = []
27          self.filter_choice = 0
28
29      # this method returns a keywords ids given a sentence and a subset
         of keyword and corresponding sentences
30      @staticmethod
31      def get_keywords_given_sentence(sentence, kw_sentence_corpus):
32          kw_id = 0
33          sent_id = 0
34          # get the sentence id from sentence hash
35          for i in PickSentences.sentences_hash:
36              if PickSentences.sentences_hash[i] == sentence:
37                  sent_id = i
38                  break
39          # get the keyword id from the corpus
40          for item in kw_sentence_corpus:
41              if sent_id in item['sent_id']:
42                  kw_id = item['kw_id']
43                  break
44          return kw_id, sent_id
45
```

```
46      # get sentence from hash
47      @staticmethod
48      def get_sentence_from_hash(sent_id):
49          try:
50              sentence = PickSentences.sentences_hash[int(sent_id)]
51          except ValueError:
52              sentence = PickSentences.sentences_hash[str(sent_id).strip
    ()]
53          except KeyError:
54              sentence = PickSentences.sentences_hash[str(sent_id).strip
    ()]
55          return sentence
56
57      # this method generates tuples of sentences with (1...n) keywords
    of sentences chosen at random
58      @staticmethod
59      def pick_candidate_sentences_random(kw_ids, inp_list, num_tuple):
60          list_sents = []
61          if num_tuple == 1:
62              for item in inp_list:
63                  rand_sent = random.choice(item['sent_id'])
64                  list_sents.append({'kw_id': item['kw_id'], 'sent_id':
    rand_sent})
65          else:
66              # get all available tuples of keywords
67              all_kw_combos = [tuple(item['kw_id']) for item in inp_list
    ]
68              picked_kws = []
```

```
69            for kw_id in kw_ids:
70                # for each kw_id pick a tuple that has that id in it
71                kw_picked_tuple = list(filter(lambda x: kw_id in x,
    all_kw_combos))
72                # now pick one of those by random and add to list
73                picked_kws.append(random.choice(kw_picked_tuple))
74            unique_tuples = list(set(picked_kws))
75            # now that we have a list of random keyword tuples, we
    need to pick  sentences based on those keywords
76            for item in unique_tuples:
77                for pair in inp_list:
78                    if sorted(list(pair['kw_id'])) == sorted(list(item
    )):
79                        rand_sent = random.choice(pair['sent_id'])
80                        list_sents.append({'kw_id': pair['kw_id'], '
    sent_id': rand_sent})
81                        break
82
83        return list_sents
84
85     # pick sentences based on tf-idf and cosine similarity for each
    set of num keywords
86     # for each number of keywords set, rank the candidate sentences by
     highest cosine sim score
87     # pick the one with the highest that has not already been picked.
88     # pick the next one making sure that a. new keywords in the pick
    and b. sentence has not already been picked
89     # do this until max number of sentences has been reached
```

```
90      # inputs are number of keywords in each sentence, list of combos
    of kw and sentences, and number of sentences to return
91      @staticmethod
92      def pick_sent_cosine_bert_sim(cos_bert, sentences, num_pick=3):
93          all_sents = []
94          uniq_kw_id = set()
95          chosen_sents = []
96          i = 1
97          tb = SentenceSimilarityMeasures()
98          for item in sentences:
99              uniq_kw_id.update(item['kw_id'])
100             for it in item['sent_id']:
101                 all_sents.append(PickSentences.sentences_hash[it])
102
103         if cos_bert == 'cos':
104             sim_matrix = tb.tfidf_cos_sim(all_sents)
105         else:
106             sim_matrix = tb.sentence_sim_bert(all_sents)
107         sorted_index = np.argsort(sim_matrix)
108         len_sorted_index = len(sorted_index)
109         print("All sentences")
110         print(all_sents)
111
112         while (i <= num_pick) and ((len_sorted_index - i) >= 0):
113             # pick the sentence that has the highest cosine sim value
114             try:
115                 current_index = sorted_index[len_sorted_index - i]
116                 chosen_sent = all_sents[current_index]
```

```
117                 print("Chosen sentence: %s" % chosen_sent)
118                 # get the kw_id[s] tuple and sentence_id of the
    sentence
119
120                 kw_id, sent_id = PickSentences.
    get_keywords_given_sentence(chosen_sent, sentences)
121                 # add it to the chosen sents
122                 chosen_sents.append({'kw_id': kw_id, 'sent_id':
    sent_id})
123                 i = i + 1
124             except IndexError:
125                 break
126
127         return chosen_sents
128
129     def pick_default_sentences(self):
130         sents = []
131         sentence_sort_by_filter = SentenceSorterByFilter()
132         # select sentences from database that have matching keywords
133         all_sentences = self.client[PickSentences.db_name][
    PickSentences.sentence_collection].find(
134             {'keywords_sys_ids': {'$in': self.kw_picked_ids},
135              'passage_id': int(self.chap_choice)})
136         all_sentences_list = list(all_sentences)
137         if len(PickSentences.sentences_hash) == 0:
138             for sent in all_sentences_list:
139                 PickSentences.sentences_hash.update({sent['_id']: sent
    ['sentence']})
```

```
140         mod_sents = pick_sentences.get_modified_sents_from_db()
141         # if there is version of modified sentence available, replace
    original sentence with modified sentence
142         mod_sents_og_ids = [mod_sent[2] for mod_sent in mod_sents]  #
    get all ids of original sentences
143         # iterate through both lists to find matches and then
    substitute with modified sentence if available
144         for ind, item in enumerate(all_sentences_list):
145             for ind_mod, item_mod in enumerate(mod_sents_og_ids):
146                 if item['_id'] == item_mod:
147                     #  replace original sentence id and sentence with
    mod sent id and sentence
148                     all_sentences_list[ind]['_id'] = 'M' + str(
    mod_sents[ind_mod][0])
149                     all_sentences_list[ind]['sentence'] = str(
    mod_sents[ind_mod][3])
150         if self.filter_choice == 'Sentence Length':
151             # display sentences by length
152             sents = sentence_sort_by_filter.
    pick_candidate_sentences_length(all_sentences_list)
153
154         elif self.filter_choice == 'Keyword in structure of sentence':
155             sents = sentence_sort_by_filter.
    pick_candidate_sentences_subject(self.kw_list, all_sentences_list)
156             # sents = self.pick_candidate_sentences_subject(kw_picked,
     all_sentences_list)
157         elif self.filter_choice == 'Number of Tier 2 and 3 words in
    sentence':
```

```
158             sents = sentence_sort_by_filter.
    pick_candidate_sentences_tier_words(all_sentences_list,
159
                PickSentences.db_name,
160
                PickSentences.tier_words_collection,
161
                PickSentences.sentences_hash)
162
163         else:
164             if self.filter_choice == 'Number of keywords':
165                 sents = sentence_sort_by_filter.
    pick_candidate_sentences_num_keywords(self.kw_picked_ids,
166
                    PickSentences.sentences_hash,
167
                    all_sentences_list)
168             else:
169                 sentences_by_num_kw = {}
170
171                 for sent in all_sentences_list:
172                     # find intersection of kw_picked_ids and
    keyword_sys_ids of sentence
173                     kw_in_sent = set(sent['keywords_sys_ids']) & set(
    self.kw_picked_ids)
174                     # if key (number of keywords in sentence) is not
    in dict, add it
```

```
175                     if len(kw_in_sent) not in sentences_by_num_kw.keys
    ():
176                         sentences_by_num_kw[len(kw_in_sent)] = [{'
    kw_id': kw_in_sent, 'sent_id': [sent['_id']]}]
177                     else:
178                         # find the (keywords list) to append the
    sentence id to it
179                         kw_found = False
180                         for item in sentences_by_num_kw[len(kw_in_sent
    )]:
181                             if item['kw_id'] == kw_in_sent:
182                                 kw_index = sentences_by_num_kw[len(
    kw_in_sent)].index(item)
183                                 sentences_by_num_kw[len(kw_in_sent)][
    kw_index]['sent_id'].append(sent['_id'])
184                                 kw_found = True
185                                 break
186                         # if keywords list is not in dict, add it to
    the dict
187                         if kw_found is False:
188                             sentences_by_num_kw[len(kw_in_sent)].
    append({'kw_id': kw_in_sent, 'sent_id': [sent['_id']]})
189
190                 if self.filter_choice == 'TD-IDF cosine sim':
191                     # pick sentences based on highest cosine
    similarity
192                     for item, value in sentences_by_num_kw.items():
```

```
193                         sents = self.pick_sent_cosine_bert_sim("cos",
    value, 5)
194
195                 elif self.filter_choice == 'BERT':
196                     # pick sentences based on highest bert similarity
197                     for item, value in sentences_by_num_kw.items():
198                         sents = self.pick_sent_cosine_bert_sim("bert",
     value, 5)
199
200                 elif self.filter_choice == 'Random':
201                     # pick sentences for each kw tuple
202                     for count, i in enumerate(self.kw_picked_ids, 1):
203                         try:
204                             picked_sentences = self.
    pick_candidate_sentences_random(self.kw_picked_ids,
205
                     sentences_by_num_kw[count],
206
                     count)
207                             for item in picked_sentences:
208                                 print(
209                                     "keyword(s): %s sentence: %s" % (
210                                         item['kw_id'], PickSentences.
    get_sentence_from_hash(['sent_id'])))
211                         except KeyError:
212                             print("No sentences found with %d keywords
    " % count)
213         return sents
```

```
214
215     def get_chapter_keywords_and_filter(self):
216         kw_str = ""
217         toolbelt = SentenceSimilarityMeasures()
218         # find title and id of all chapters in db
219         db_res = self.client[PickSentences.db_name][PickSentences.
    passage_collection].find({}, {"title": 1})
220         print("Index number \t Title")
221         for item in db_res:
222             print("%d \t %s" % (item['_id'], item['title']))
223
224         # get default and user supplied keywords for selected chapter
225         self.chap_choice = pyip.inputNum("Enter the Index number of
    the chapter you want to pick sentences for today: ")
226         passage_kw = self.client[PickSentences.db_name][PickSentences.
    keywords_collection].find(
227             {'passage_num': {'$in': [int(self.chap_choice)]}, '
    inserted_by': 'system'},
228             {'keyword': 1, '_id': 1})
229         passage_kw = list(passage_kw)
230
231         for item in passage_kw:
232             kw_str = kw_str + " " + str(item['keyword'])
233             PickSentences.keywords_hash.update({item['_id']: item['
    keyword']})
234         # kw_str = ' '.join(item['keyword'] for item in passage_kw)
235         print("\nSystem Keywords for this passage are: ", kw_str)
```

```
236         passage_others_kw = self.client[PickSentences.db_name][
    PickSentences.keywords_collection].find(
237             {'passage_num': {'$in': [int(self.chap_choice)]}, '
    inserted_by': {'$ne': 'system'}},
238             {'keyword': 1, '_id': 1})
239         passage_others_kw = list(passage_others_kw)
240         if len(passage_others_kw) > 0:
241             kw_others_str = ""
242             for item in passage_others_kw:
243                 kw_others_str = kw_others_str + " " + str(item['
    keyword'])
244                 PickSentences.keywords_hash.update({item['_id']: item[
    'keyword']})
245             print("Others have added the following keywords: ",
    kw_others_str)
246             kw_all = list(kw_str.split(" ")) + list(kw_others_str.
    split(" "))
247             # kw_others_str = ' '.join(item['keyword'] for item in
    passage_others_kw)
248             # print("%s\n" % )
249         else:
250             kw_all = list(kw_str.split(" "))
251         kw_picked = pyip.inputStr("\n Please enter the keywords you
    want to use in today's class: ")
252         # combine default and user keyword lists
253         kw_picked_str = list(kw_picked.split(" "))
254         for index, item in enumerate(kw_picked_str):
255             if item not in kw_all:
```

```
256                 correct_kw = toolbelt.typos_closest_match(item, kw_all
    )
257                 kw_picked_str[index] = correct_kw
258                 print("\n %s has been corrected to %s" % (item,
    correct_kw))
259         # validate the user entered list against combined list to
    ensure that only words from the database are chosen
260         self.kw_list = list(set(kw_all) & set(kw_picked_str))
261         # print(kw_picked)
262         # get ids for the lesson keywords
263         # combine both default and user keyword lists and create a
    dict of kw_id and keyword
264         kw_all_dict = {}
265         for item in passage_kw:
266             kw_all_dict[item['keyword']] = item['_id']
267         for item in passage_others_kw:
268             kw_all_dict[item['keyword']] = item['_id']
269
270         for item in self.kw_list:
271             self.kw_picked_ids.append(kw_all_dict[item])
272         # get the user choice on filter to use for ranking sentences
273         self.filter_choice = pyip.inputMenu(
274             ['Sentence Length', 'Number of keywords', 'Keyword in
    structure of sentence',
275              'Number of Tier 2 and 3 words in sentence'],
276             prompt="Please enter the number of the sort you would like
     applied today.\n",
277             numbered=True)
```

```
278         return 0
279
280     def modify_sentences(self):
281         sent_mod_choice = pyip.inputYesNo("Would you like to modify
    any of these sentences?")
282         inserted_sentences = []
283         if sent_mod_choice == 'yes':
284             # get the id of last sentence in the collection
285             num_docs_cursor = self.client[PickSentences.db_name][
    PickSentences.modified_sentences_collection].find({}, {
286                 "_id": 1}).sort("_id", pymongo.DESCENDING).limit(1)
287             num_docs = num_docs_cursor.next()
288             index_st = num_docs['_id'] + 1
289             self.username = pyip.inputStr("Please enter your username:
     ")
290             sent_choice = pyip.inputStr(
291                 "Which sentences would you like to modify? Please
    enter the sentence IDs separated by a space.")
292             sent_mod_ids = sent_choice.split(" ")
293             for s_id in sent_mod_ids:
294                 try:
295                     s_id = int(s_id)
296                     mod_exists = self.client[PickSentences.db_name][
297                         PickSentences.modified_sentences_collection].
    find_one(
298                         {"orig_sent_id": s_id, "modified_by": self.
    username}, limit=1)
299                     if mod_exists:
```

```
300                         rep_mod = pyip.inputYesNo(
301                             "Sentence # %d has already been modified
    by you. The modified sentence is : %s Would you like to replace
    this modified sentence? " % (
302                                 s_id, mod_exists['modified_sentence'])
    )
303                         # check if user wants to replace existing
    modified sentence
304                         if rep_mod == "yes":
305                             sentence = self.get_sentence_from_hash(
    s_id)
306                             sent_mod = pyip.inputStr("Please enter the
     alternate version of sentence: %s" % sentence)
307                             sent_length = len(sent_mod.strip().split()
    )
308                             filter_sent = {"orig_sent_id": int(s_id),
    "modified_by": self.username}
309
310                             update_dict = {"$set": {"sentence_length":
     sent_length,
311                             "modified_by": self.username,
312                             "modified_sentence": sent_mod
313                              }}
314                             insertResult = self.client[PickSentences.
    db_name][
315                                 PickSentences.
    modified_sentences_collection].find_one_and_update(
316                                 filter_sent, update_dict)
```

```
317                             inserted_sentences.append((int(s_id), self
    .username, insertResult['_id'], sent_mod))
318                             index_st += 1
319                             print("Sentence has successfully been
    modified.")
320 
321                     else:
322                         sentence = PickSentences.
    get_sentence_from_hash(s_id)
323                         sent_mod = pyip.inputStr("Please enter the
    alternate version of sentence: %s" % sentence)
324                         sent_length = len(sent_mod.strip().split())
325 
326                         if sent_length > 0:
327                             insert_dict = {"_id": index_st,
328                             "modified_sentence": sent_mod,
329                             "orig_sent_id": int(s_id),
330                             "sentence_length": sent_length,
331                             "modified_by": self.username
332                             }
333 
334                             try:
335                                 insertResult = self.client[
    PickSentences.db_name][
336                                     PickSentences.
    modified_sentences_collection].insert_one(
337                                     insert_dict)
338                                 inserted_sentences.append(
```

```
339                                     (int(s_id), self.username,
insertResult.inserted_id, sent_mod))
340                                 index_st += 1
341                                 print("Sentence has successfully been
modified.")
342                             except pymongo.errors.DuplicateKeyError:
343                                 print("Sentence %s already in database
" % sentence.text)
344
345                 except KeyError:
346                     print("Sentence with id %d does not exist." % s_id
)
347         return inserted_sentences
348
349     # this method retrieves modified sentences for the system picked
sentences from mongoDB
350     def get_modified_sents_from_db(self):
351         m_sents = []
352         sent_ids = PickSentences.sentences_hash.keys()
353         l_sent_ids = list(sent_ids)
354         modded_sents = self.client[PickSentences.db_name][
PickSentences.modified_sentences_collection].find(
355             {'orig_sent_id': {'$in': l_sent_ids}})
356         modded_sents = list(modded_sents)
357         for m_sent in modded_sents:
358             # create a list of ({orig_sent_id:(mod_sent_id,
mod_sentence, modified_by)})
359             m_sents.append(
```

```
360                 (m_sent['_id'], m_sent['modified_by'], m_sent['
orig_sent_id'], m_sent['modified_sentence']))
361             m_id = 'M' + str(m_sent['_id'])
362             PickSentences.sentences_hash.update({m_id: m_sent['
modified_sentence']})
363         return m_sents
364
365     @staticmethod
366     # return the id of the keyword or sentence using the keywords_hash
     or sentence_hash structures
367     def return_id_from_hash(str_to_match, kw_sent='kw'):
368         if kw_sent == 'kw':
369             for key, item in PickSentences.keywords_hash.items():
370                 if item == str_to_match:
371                     return key
372         else:
373             for key, item in PickSentences.sentences_hash.items():
374                 if item.strip() == str_to_match.strip():
375                     return key
376
377     def pick_n_most_similar_sentences(self, sys_sents, n=2):
378         kw_str = ''
379         toolbelt = SentenceSimilarityMeasures()
380         for kw_id in self.kw_picked_ids:
381             kw_str += self.keywords_hash[kw_id] + ' '
382         kw_str = kw_str.rstrip()
383         sim_sentences = {}
384         # get only sentences to send to the similarity method
```

```
385         simp_sents = [s['sentence'] for s in sys_sents['simple']]
386         sents = toolbelt.calculate_spacy_sim(kw_str, simp_sents, n)
387         sim_sentences['simple'] = [{'_id': PickSentences.
    return_id_from_hash(item, 'sent'), 'sentence': item} for item
388                                    in sents]
389
390         med_sents = [s['sentence'] for s in sys_sents['medium']]
391         sents = toolbelt.calculate_spacy_sim(kw_str, med_sents, n)
392         sim_sentences['medium'] = [{'_id': PickSentences.
    return_id_from_hash(item, 'sent'), 'sentence': item} for item
393                                    in
394                                    sents]
395
396         complex_sents = [s['sentence'] for s in sys_sents['complex']]
397         sents = toolbelt.calculate_spacy_sim(kw_str, complex_sents, n)
398         sim_sentences['complex'] = [{'_id': PickSentences.
    return_id_from_hash(item, 'sent'), 'sentence': item} for item
399                                     in sents]
400         return sim_sentences
401
402     @staticmethod
403     def print_sentences(sys_sents):
404         if sys_sents['simple'] and len(sys_sents['simple']) > 0:
405             print("Simple sentences")
406             for item in sys_sents['simple']:
407                 print("%s: %s " % (item['_id'], item['sentence']))
408         if sys_sents['medium'] and len(sys_sents['medium']) > 0:
409             print("Medium sentences")
```

```
410             for item in sys_sents['medium']:
411                 print("%s: %s " % (item['_id'], item['sentence']))
412         if sys_sents['complex'] and len(sys_sents['complex']) > 0:
413             print("Complex sentences")
414             for item in sys_sents['complex']:
415                 print("%s: %s " % (item['_id'], item['sentence']))
416         return 0
417 
418     def get_last_index_from_collection(self, collection_name):
419         new_index = 0
420         num_docs_curs = self.client[PickSentences.db_name][
    collection_name].find(
421             {}, {
422                 "_id": 1}).sort("_id", pymongo.DESCENDING).limit(1)
423         num_docs_curs_l = list(num_docs_curs)
424         if len(num_docs_curs_l) > 0:
425             new_index = num_docs_curs_l[0]['_id'] + 1
426         return new_index
427 
428     @staticmethod
429     def save_sents_to_db(pick_sents, auto_sentence_ids):
430         toolbelt = SentenceSimilarityMeasures()
431         if auto_sentence_ids is not None and len(auto_sentence_ids) >
    0:
432             simple_sents_to_save = auto_sent_ids['simple']
433             med_sents_to_save = auto_sentence_ids['medium']
434             complex_sents_to_save = auto_sent_ids['complex']
435         else:
```

```
436             # ask instructor which sentences want to be saved to db in
     simple complex and medium
437             simple_sents_to_save = pyip.inputStr(
438                 "Please select SIMPLE sentences to be saved for this
    passage from this list. Please type the index number of the
    sentences separated by a space. Simply hit enter if you do not
    want to save any simple sentences.",
439                 blank=True)
440             med_sents_to_save = pyip.inputStr(
441                 "Next, please select MEDIUM complexity sentences to be
     saved for this passage from this list. Please type the index
    number of the sentences separated by a space. Hit enter if you do
    not want to save any medium complexity sentences.",
442                 blank=True)
443             complex_sents_to_save = pyip.inputStr(
444                 "Next, please select COMPLEX sentences to be saved for
     this passage from this list. Please type the index number of the
    sentences separated by a space. Hit enter if you do not want to
    save any complex sentences.",
445                 blank=True)
446
447         # save to db collection
448         # get the id‘ of last sentence in the collection
449         index_start = pick_sents.get_last_index_from_collection(
    PickSentences.sentences_for_passage_collection)
450         simp_ids_to_save = simple_sents_to_save.strip().split(" ") if
    len(simple_sents_to_save) > 0 else []
```

```
451         med_ids_to_save = med_sents_to_save.strip().split(" ") if len(
    med_sents_to_save) > 0 else []
452         comp_ids_to_save = complex_sents_to_save.strip().split(" ") if
     len(complex_sents_to_save) > 0 else []
453 
454         if len(simp_ids_to_save) > 0 or len(med_ids_to_save) > 0 or
    len(comp_ids_to_save) > 0:
455             username = pyip.inputStr("Please enter your username: ")
456             for id_ins in simp_ids_to_save:
457                 id_ins = id_ins.strip()
458                 # identify if sentence is system or user generated
459                 if id_ins.upper().startswith("M"):
460                     sent_type = "user"
461                 else:
462                     sent_type = "system"
463                 # get the ID number of the sentence (strip the M if
    necessary)
464                 match = re.match(r"([a-z]?(\d+))", str(id_ins), re.I)
465                 sentence_id = match.groups(0)[1]
466                 # insert dashed version of sentence
467                 kw_matches_in_sent = toolbelt.get_keywords_in_sentence
    (pick_sents.get_sentence_from_hash(id_ins.upper()),
468 
     pick_sents.kw_list)
469 
470                 insert_tuple = {"_id": index_start,
471                                 "username": username,
472                                 "passage_num": pick_sents.chap_choice,
```

```
473                                 "sort": pick_sents.filter_choice,
474                                 "keywords": pick_sents.kw_picked_ids,
475                                 "sentence_id": int(sentence_id),
476                                 "sentence_type": sent_type,
477                                 "difficulty_level": "simple",
478                                 "sentence_with_spaces": toolbelt.
    replace_kw_with_blanks(
479                                     pick_sents.get_sentence_from_hash(
    id_ins.upper()), kw_matches_in_sent),
480                                 "source": "passage"
481                                 }
482                 try:
483                     pick_sents.client[PickSentences.db_name][
    PickSentences.sentences_for_passage_collection].insert_one(
484                         insert_tuple)
485                     index_start = index_start + 1
486                 except Exception as exp:
487                     print("Error occurred:", exp)
488
489             for id_ins in med_ids_to_save:
490                 # identify if sentence is system or user generated
491                 if id_ins.upper().startswith("M"):
492                     sent_type = "user"
493                 else:
494                     sent_type = "system"
495                 # get the ID number of the sentence (strip the M if
    necessary)
496                 match = re.match(r"([a-z]?(\d+))", str(id_ins), re.I)
```

```
497                 sentence_id = match.groups(0)[1]
498                 kw_matches_in_sent = toolbelt.get_keywords_in_sentence
    (pick_sents.get_sentence_from_hash(id_ins.upper()),
499
     pick_sents.kw_list)
500                 insert_tuple = {"_id": index_start,
501                                 "username": username,
502                                 "passage_num": pick_sents.chap_choice,
503                                 "keywords": pick_sents.kw_picked_ids,
504                                 "sort": pick_sents.filter_choice,
505                                 "sentence_id": int(sentence_id),
506                                 "sentence_type": sent_type,
507                                 "difficulty_level": "medium",
508                                 "sentence_with_spaces": toolbelt.
    replace_kw_with_blanks(
509                                     pick_sents.get_sentence_from_hash(
    id_ins.upper()), kw_matches_in_sent),
510                                 "source": "passage"
511                                 }
512                 try:
513                     pick_sents.client[PickSentences.db_name][
    PickSentences.sentences_for_passage_collection].insert_one(
514                         insert_tuple)
515                     index_start = index_start + 1
516                 except Exception as exp:
517                     print("Error occurred:", exp)
518             for id_ins in comp_ids_to_save:
519                 # identify if sentence is system or user generated
```

```
520                 if id_ins.upper().startswith("M"):
521                     sent_type = "user"
522                 else:
523                     sent_type = "system"
524                 # get the ID number of the sentence (strip the M if
    necessary)
525                 match = re.match(r"([a-z]?(\d+))", str(id_ins), re.I)
526                 sentence_id = match.groups(0)[1]
527                 kw_matches_in_sent = toolbelt.get_keywords_in_sentence
    (pick_sents.get_sentence_from_hash(id_ins.upper()),
528
     pick_sents.kw_list)
529                 insert_tuple = {"_id": index_start,
530                                 "username": username,
531                                 "passage_num": pick_sents.chap_choice,
532                                 "keywords": pick_sents.kw_picked_ids,
533                                 "sort": pick_sents.filter_choice,
534                                 "sentence_id": int(sentence_id),
535                                 "sentence_type": sent_type,
536                                 "difficulty_level": "complex",
537                                 "sentence_with_spaces": toolbelt.
    replace_kw_with_blanks(
538                                     pick_sents.get_sentence_from_hash(
    id_ins.upper()), kw_matches_in_sent),
539                                 "source": "passage"
540                                 }
541                 try:
```

```
542                     pick_sents.client[PickSentences.db_name][
    PickSentences.sentences_for_passage_collection].insert_one(
543                         insert_tuple)
544                     index_start = index_start + 1
545                 except pymongo.errors.DuplicateKeyError:
546                     print("You have previously entered this sentence
    for this sort type. Skipping. ")
547                 except Exception as exp:
548                     print("Error occurred:", exp)
549                     return 1
550         return 0
551 
552     def check_if_sents_already_present(self):
553         # select sentences from database that have matching chapter
    choice, filter and have all the requested keywords present
554         all_sentences_filter = self.client[PickSentences.db_name][
    PickSentences.sentences_for_passage_collection].find(
555             {'keywords': {'$all': self.kw_picked_ids},  # check if
    this has accurate values
556              'sort': self.filter_choice,
557              "source": "passage",
558              'passage_num': int(self.chap_choice)})
559         all_sentences_filter_list = list(all_sentences_filter)
560         # get the sentence matching the id
561         if len(all_sentences_filter_list) > 0:
562             final_sents = {}
563             simp_sents = []
564             med_sents = []
```

```
565             comp_sents = []
566             for item in all_sentences_filter_list:
567                 if item['difficulty_level'] == 'simple':
568                     if item['sentence_type'] == 'user':
569                         s = self.client[PickSentences.db_name][
    PickSentences.modified_sentences_collection].find_one(
570                             {'_id': int(item['sentence_id'])},
571                             {'sentence': 1, '_id': 0})
572 
573                         simp_sents.append({'_id': 'M' + item['
    sentence_id'], 'sentence': s['sentence']})
574                         PickSentences.sentences_hash.update({'M' +
    item['sentence_id']: s['sentence']})
575                     else:
576                         s = self.client[PickSentences.db_name][
    PickSentences.sentence_collection].find_one(
577                             {'_id': int(item['sentence_id'])},
578                             {'sentence': 1, '_id': 0})
579                         simp_sents.append({'_id': item['sentence_id'],
     'sentence': s['sentence']})
580                         PickSentences.sentences_hash.update({item['
    sentence_id']: s['sentence']})
581                 elif item['difficulty_level'] == 'medium':
582                     if item['sentence_type'] == 'user':
583                         s = self.client[PickSentences.db_name][
    PickSentences.modified_sentences_collection].find_one(
584                             {'_id': int(item['sentence_id'])},
585                             {'sentence': 1, '_id': 0})
```

```
586
587                         med_sents.append({'_id': 'M' + item['
    sentence_id'], 'sentence': s['sentence']})
588                         PickSentences.sentences_hash.update({'M' +
    item['sentence_id']: s['sentence']})
589                     else:
590                         s = self.client[PickSentences.db_name][
    PickSentences.sentence_collection].find_one(
591                             {'_id': int(item['sentence_id'])},
592                             {'sentence': 1, '_id': 0})
593                         med_sents.append({'_id': item['sentence_id'],
    'sentence': s['sentence']})
594                         PickSentences.sentences_hash.update({item['
    sentence_id']: s['sentence']})
595                 elif item['difficulty_level'] == 'complex':
596                     if item['sentence_type'] == 'user':
597                         s = self.client[PickSentences.db_name][
    PickSentences.modified_sentences_collection].find_one(
598                             {'_id': int(item['sentence_id'])},
599                             {'sentence': 1, '_id': 0})
600
601                         comp_sents.append({'_id': 'M' + item['
    sentence_id'], 'sentence': s['sentence']})
602                         PickSentences.sentences_hash.update({'M' +
    item['sentence_id']: s['sentence']})
603                     else:
604                         s = self.client[PickSentences.db_name][
    PickSentences.sentence_collection].find_one(
```

```
                            {'_id': int(item['sentence_id'])},
                            {'sentence': 1, '_id': 0})
                        comp_sents.append({'_id': item['sentence_id'],
 'sentence': s['sentence']})
                        PickSentences.sentences_hash.update({item['
sentence_id']: s['sentence']})

            final_sents['simple'] = simp_sents
            final_sents['medium'] = med_sents
            final_sents['complex'] = comp_sents
            return final_sents

    def get_other_sents_passage(self):
        all_sentences_filter = self.client[PickSentences.db_name][
PickSentences.sentences_for_passage_collection].find(
            {'keywords': {'$all': self.kw_picked_ids},  # check if
this has accurate values
             'sort': self.filter_choice,
             "source": "other",
             'passage_num': int(self.chap_choice)})
        all_sentences_filter_list = list(all_sentences_filter)
        if len(all_sentences_filter_list) > 0:
            final_sents = {}
            simp_sents = []
            med_sents = []
            comp_sents = []
            for item in all_sentences_filter_list:
                if item['difficulty_level'] == 'simple':
```

```
629                     if item['sentence_type'] == 'user':
630                         s = self.client[PickSentences.db_name][
    PickSentences.modified_sentences_collection].find_one(
631                             {'_id': int(item['sentence_id'])},
632                             {'sentence': 1, '_id': 0})
633 
634                         simp_sents.append({'_id': 'M' + item['
    sentence_id'], 'sentence': s['sentence']})
635                         PickSentences.sentences_hash.update({'M' +
    item['sentence_id']: s['sentence']})
636                     else:
637                         s = self.client[PickSentences.db_name][
    PickSentences.sentence_collection].find_one(
638                             {'_id': int(item['sentence_id'])},
639                             {'sentence': 1, '_id': 0})
640                         simp_sents.append({'_id': item['sentence_id'],
     'sentence': s['sentence']})
641                         PickSentences.sentences_hash.update({item['
    sentence_id']: s['sentence']})
642                 elif item['difficulty_level'] == 'medium':
643                     if item['sentence_type'] == 'user':
644                         s = self.client[PickSentences.db_name][
    PickSentences.modified_sentences_collection].find_one(
645                             {'_id': int(item['sentence_id'])},
646                             {'sentence': 1, '_id': 0})
647 
648                         med_sents.append({'_id': 'M' + item['
    sentence_id'], 'sentence': s['sentence']})
```

```
649                         PickSentences.sentences_hash.update({'M' +
    item['sentence_id']: s['sentence']})
650                     else:
651                         s = self.client[PickSentences.db_name][
    PickSentences.sentence_collection].find_one(
652                             {'_id': int(item['sentence_id'])},
653                             {'sentence': 1, '_id': 0})
654                         med_sents.append({'_id': item['sentence_id'],
    'sentence': s['sentence']})
655                         PickSentences.sentences_hash.update({item['
    sentence_id']: s['sentence']})
656                 elif item['difficulty_level'] == 'complex':
657                     if item['sentence_type'] == 'user':
658                         s = self.client[PickSentences.db_name][
    PickSentences.modified_sentences_collection].find_one(
659                             {'_id': int(item['sentence_id'])},
660                             {'sentence': 1, '_id': 0})
661
662                         comp_sents.append({'_id': 'M' + item['
    sentence_id'], 'sentence': s['sentence']})
663                         PickSentences.sentences_hash.update({'M' +
    item['sentence_id']: s['sentence']})
664                     else:
665                         s = self.client[PickSentences.db_name][
    PickSentences.sentence_collection].find_one(
666                             {'_id': int(item['sentence_id'])},
667                             {'sentence': 1, '_id': 0})
```

```
668                         comp_sents.append({'_id': item['sentence_id'],
     'sentence': s['sentence']})
669                         PickSentences.sentences_hash.update({item['
    sentence_id']: s['sentence']})
670
671             final_sents['simple'] = simp_sents
672             final_sents['medium'] = med_sents
673             final_sents['complex'] = comp_sents
674             return final_sents
675
676
677 if __name__ == "__main__":
678     pick_sentences = PickSentences()
679     other_sentences = PickSentencesOtherPassages()
680     toolbelt_sents = SentenceSimilarityMeasures()
681     # get chapter choice, keywords and filter choice from user
682     get_chap = pick_sentences.get_chapter_keywords_and_filter()
683     present_sents = pick_sentences.check_if_sents_already_present()
684     if present_sents is None or len(present_sents) == 0:
685         # pick the default sentences
686         def_sents = pick_sentences.pick_default_sentences()
687         # pick n most similar sentences to keywords
688         most_sim_sentences = pick_sentences.
    pick_n_most_similar_sentences(def_sents)
689         print("Here are the simple, medium and complex sentences most
    representative of the passage: ")
690         PickSentences.print_sentences(most_sim_sentences)
691         auto_sents_yes_no = pyip.inputMenu(
```

```
692             ['Choose 1 if you are happy with the current list of
    sentences.',
693              'Choose 2 if you would like to pick your own sentences.'
    ],
694             prompt="Please enter the number of the choice you would
    like to proceed with: \n",
695             numbered=True)
696         if auto_sents_yes_no == 'Choose 1 if you are happy with the
    current list of sentences.':
697             # get all the ids in str format for the save_sents_to_db
    method
698             str_ids = ''
699             auto_sent_ids = {}
700             for xy in most_sim_sentences['simple']:
701                 str_ids = str_ids + ' ' + str(xy['_id'])
702             auto_sent_ids['simple'] = str_ids
703             str_ids = ''
704             for xy in most_sim_sentences['medium']:
705                 str_ids = str_ids + ' ' + str(xy['_id'])
706             auto_sent_ids['medium'] = str_ids
707             str_ids = ''
708             for xy in most_sim_sentences['complex']:
709                 str_ids = str_ids + ' ' + str(xy['_id'])
710             auto_sent_ids['complex'] = str_ids
711             saved_sents = PickSentences.save_sents_to_db(
    pick_sentences, auto_sent_ids)
712             print("These sentences have been saved for future use.")
713         else:
```

```
714             # print system and previously modified sentences
715             PickSentences.print_sentences(def_sents)
716             # check if sentences need to be modified in this session
717             inserted_sents = pick_sentences.modify_sentences()
718             # if yes, append mod_sent list with inserted_sent list,
    and reprint all sentences
719             if inserted_sents and len(inserted_sents) > 0:
720                 print("Here are the final list of system generated and
     modified sentences: ")
721                 PickSentences.print_sentences(def_sents)
722             saved_sents = PickSentences.save_sents_to_db(
    pick_sentences, [])
723         other_sents = other_sentences.pick_other_sentences(
    pick_sentences.kw_picked_ids, pick_sentences.chap_choice,
724
    pick_sentences.filter_choice)
725         print("Here are three sentences with the keywords you
    requested that have been selected from other passages:")
726         PickSentences.print_sentences(other_sents)
727         # Save other sents in the db too.
728         index_beg = PickSentences.get_last_index_from_collection(
    pick_sentences,
729
    PickSentences.sentences_for_passage_collection)
730
731         for sent_t, sent_other in other_sents.items():
732             if len(sent_other) > 0:
```

```
733                 full_sentence = pick_sentences.client[PickSentences.
    db_name][PickSentences.sentence_collection].find_one(
734                     {'_id': int(sent_other[0]["_id"])},
735                     {'sentence': 1, '_id': 0})
736                 pick_sentences.sentences_hash.update({int(sent_other
    [0]["_id"]): full_sentence['sentence']})
737         for sent_t, sent_other in other_sents.items():  # sent_t =
    simple|medium|complex sent_other: sentence
738             # with other details
739             if len(sent_other) > 0:
740                 kws_in_sent = toolbelt_sents.get_keywords_in_sentence(
741                     pick_sentences.get_sentence_from_hash(int(
    sent_other[0]["_id"])),
742                     pick_sentences.kw_list)
743                 insert_row = {"_id": index_beg,
744                               "username": pick_sentences.username,
745                               "passage_num": pick_sentences.
    chap_choice,
746                               "sort": pick_sentences.filter_choice,
747                               "keywords": pick_sentences.kw_picked_ids
    ,
748                               "sentence_id": int(sent_other[0]["_id"])
    ,
749                               "sentence_type": "system",
750                               "difficulty_level": sent_t,
751                               "sentence_with_spaces": toolbelt_sents.
    replace_kw_with_blanks(
```

```
752                                     pick_sentences.
    get_sentence_from_hash(int(sent_other[0]["_id"])), kws_in_sent),
753                                 "source": "other"
754                                 }
755                 try:
756
757                     insertOtherResult = pick_sentences.client[
    PickSentences.db_name][
758                         PickSentences.sentences_for_passage_collection
    ].insert_one(
759                         insert_row)
760                     index_beg += 1
761                 except pymongo.errors.DuplicateKeyError:
762                     print("Sentence %s already in database" %
    sent_other[0]["_id"])
763             else:
764                 print(
765                     "Unfortunately SENCE could not find any %s
    sentences from other lessons for this criteria. " % sent_t)
766
767      else:
768          # print system and previously modified sentences
769          print("Sentences have already been picked for this filter and
    set of keywords. They are: ")
770          PickSentences.print_sentences(present_sents)
771          inserted_sents = pick_sentences.modify_sentences()
772          # print sentences from other passages
```

```
773         print("Here are three sentences with the keywords you
    requested that have been selected from other passages:")
774         other_sents = pick_sentences.get_other_sents_passage()
775         PickSentences.print_sentences(other_sents)
```

toolbelt.py

```
1  import configparser
2  from sklearn.metrics.pairwise import cosine_similarity
3  from sklearn.feature_extraction.text import TfidfVectorizer
4  import numpy as np
5  from sentence_transformers import SentenceTransformer
6  from nltk.metrics import edit_distance
7  import spacy
8  from spacy.matcher import Matcher
9
10
11 class SentenceSimilarityMeasures:
12     def __init__(self):
13         self.tfidf_vectorizer = TfidfVectorizer()
14         self.model = SentenceTransformer('bert-base-nli-mean-tokens')
15         self.nlp = spacy.load('en_core_web_sm')
16
17     def tfidf_cos_sim(self, corpus):
18         tfidf_matrix = self.tfidf_vectorizer.fit_transform(corpus)
19         cosine_sim = cosine_similarity(tfidf_matrix, tfidf_matrix)
20         avg_cos_sim = np.average(cosine_sim, axis=1)
21         return avg_cos_sim
```

```
22
23      def tfidf_cos_sim_compare_single_sentence_to_corpus(self, sentence
        , corpus):
24          cosine_sim = []
25          for sent in corpus:
26              tfidf = self.tfidf_vectorizer.fit_transform(sentence, sent
        )
27              cosine_sim.append(cosine_similarity(tfidf[0], tfidf[1])
        [0][0])
28          return min(cosine_sim)
29
30      def sentence_sim_bert(self, corpus):
31          sentence_embeddings = self.model.encode(corpus)
32          cosine_sim = cosine_similarity(sentence_embeddings,
        sentence_embeddings)
33          avg_cos_sim = np.average(cosine_sim, axis=1)
34          return avg_cos_sim
35
36      # this method uses spacy to identify the sentence (from a list of
        sentences) closest to a group of words
37      def calculate_spacy_sim(self, words_to_compare, sents,
        num_to_return=1):
38          sims = []
39          sents_return = []
40          doc1 = self.nlp(words_to_compare)
41          # find the similarity measure between the list of words and
        the sentence
42          for sent in sents:
```

```
43              doc2 = self.nlp(sent)
44              sims.append(doc1.similarity(doc2))
45          npcs = np.array(sims)
46          # sort the indices in descending order
47          ascending_order_index = npcs.argsort()
48          npcs_index_array = ascending_order_index[::-1]  # we need
    reverse order since we want the top 'x' values
49          # retrieve the num_to_return most similar sentences
50          for ind in range(0, num_to_return):
51              try:
52                  sents_return.append(sents[npcs_index_array[ind]])
53              except IndexError:
54                  break
55          return sents_return
56
57      # this method inputs the typo word and a list of keywords. It uses
        NLTK edit distance to find the closest matched keyword
58      @staticmethod
59      def typos_closest_match(typo, keywords, limit=2):
60          dist = 99
61          match = typo
62          typo = typo.lower()
63          kw_lower = [k.lower() for k in keywords]
64          for kw in kw_lower:
65              edit_dist = edit_distance(typo, kw)
66              if edit_dist < dist and edit_dist <= limit:
67                  dist = edit_dist
68                  match = kw
```

```
69
70          return match
71
72      # this method accepts a sentence and a list of keywords for the
    lesson and identifies
73      # which keywords are present in the sentence
74      def get_keywords_in_sentence(self, sentence, kw_list):
75          # kw_list = [AssessSentences.keywords_hash[k_id] for k_id in
    kw_ids]
76          match_list = []
77          doc = self.nlp(sentence.lower())
78
79          matcher = Matcher(self.nlp.vocab)
80          for item in kw_list:
81              matcher.add("matched kw", [[{'LEMMA': item}]])
82          matches = matcher(doc)
83          for match_id, start, end in matches:
84              span = doc[start:end]
85              # store the matched keyword, and the starting and ending
    token index
86              match_list.append({'keyword': span, 'start': start, 'end':
     end})
87          return match_list
88
89      # this method checks to see if an honorific such as Mr. Mrs., Miss
     etc. is present in a phrase
90      def check_for_honorific(self, phrase):
91          doc = self.nlp(phrase)
```

```
92          for tok in doc:
93              if tok.pos_ == 'PROPN' and tok.text[-1] == '.':  # check
    if token is a proper noun
94                  # and the final character is a dot.
95                  return True
96              else:
97                  return False
98
99      def replace_kw_with_blanks(self, sentence, matched_keywords):
100         config = configparser.ConfigParser()
101         try:
102             config.read('../config.ini')
103             sentence_blank = config['SENTENCES']['sentence_blank']
104         except KeyError:
105             config.read('config.ini')
106             sentence_blank = config['SENTENCES']['sentence_blank']
107         sent = self.nlp(sentence)
108         new_sent = ''
109         used_kw = []
110         for kw_item in matched_keywords:
111             if kw_item['keyword'].text not in used_kw:
112                 # check if sent[0:kw_item['start'].text has an
    honorific such as Mr. Mrs. etc. to account for the bug
113                 # in Matcher. Matcher looks at Mr. as one token, spacy
     tokenizer sees them as two.
114                 if self.check_for_honorific(sent[0: kw_item['start']].
    text):
115                     start = kw_item['start'] - 1
```

```
116                     end = kw_item['end'] - 1
117
118                 else:
119                     start = kw_item['start']
120                     end = kw_item['end']
121                 new_sent = sent[0: start].text + sent[
122                     start - 1].whitespace_  # If tokens have been
123                 # skipped, add them in (with trailing whitespace if
    available)
124                 new_sent += sentence_blank + sent[start].whitespace_
    # Replace token, with trailing
125                 # whitespace if available
126                 new_sent += sent[end:].text  # add remainder of the
    sentence
127                 sent = self.nlp(
128                     new_sent)  # for more than one keyword in the
    sentence, regenerate the tokenized form of
129                 # the sentence to repeat this process
130                 used_kw.append(
131                     kw_item['keyword'].text)  # this is so that we don
    't add the same keyword as multiple dashes
132         return new_sent
133
134     def replace_words_in_sentence(self, sentence, word, pos_to_replace
    ):
135         sent = self.nlp(sentence)
136         word = word.strip()
137         new_sent = sent[0: pos_to_replace].text + sent[
```

```
138             pos_to_replace - 1].whitespace_  # If tokens have been
    skipped, add them in (with trailing whitespace
139         # if available)
140         new_sent += word + sent[pos_to_replace].whitespace_  # Replace
     token, with trailing
141         # whitespace if available
142         new_sent += sent[pos_to_replace + 1:].text  # add remainder of
     the sentence
143         return new_sent
144
145     def find_match_return_tok_pos(self, sentence, keywords):
146         doc = self.nlp(sentence.lower())
147         used_kw = []
148         tok_pos = {}
149         for token in doc:
150             if token.lemma_ in keywords and token.lemma_ not in
    used_kw:
151                 tok_pos[token.text] = token.i  # return a dict of
    matched keyword to index of word in sentence
152                 used_kw.append(token.lemma_)
153         return tok_pos
```

Vita

Born in Chennai, India, Tabitha K. Samuel completed her Bachelor's in Computer Science and Engineering from Easwari Engineering College in 2005. She worked for two years as a Software Engineer at Infosys, Limited, a top IT and software services company in India. This is where she learned much about software development methodologies that have served her well for decades. She moved to Knoxville, Tennessee, in 2007 to join the Master's in Computer Science program. With Dr. Berry as her advisor, she completed her Master's project on a Performance Evaluation of the MATLAB Parallel Computing Toolbox for Parallel Implementations of Nonnegative Tensor Factorization. On graduation in 2009, she joined the National Institute for Computational Sciences (NICS) at the University of Tennessee, Knoxville as a User Support Specialist for (at that time's) world's fastest academic supercomputer - Kraken. There, she discovered the remarkable and diverse world of academic research computing and has been a part of its ecosystem since. Tabitha has worked at NICS for over 15 years and currently serves as its interim Director. An active Research Computing and Data (RCD) community member, Tabitha is co-PI of the Building Research Innovation at Community Colleges (BRICCs)-Pathways (NSF #2346751) and BRICCs-Research Data Management (NSF #2437898) NSF projects, focusing on collaborative cyberinfrastructure advancement and data management. She is also the co-founder of Tennessee-RCD, a platform for collaboration and regional advancement in cyberinfrastructure for RCD professionals in Tennessee. More information can be found at: https://www.linkedin.com/in/tabithasamuel/.