Allegheny College
Department of Computer Science

Senior Thesis

Natural Language Processing and Student Reflective Writing

by

Michael Abraham

ALLEGHENY COLLEGE

DEPARTMENT OF COMPUTER AND INFORMATION SCIENCE

Project Supervisor: **Douglas Luman**
Co-Supervisor: **Alexis Hart**

08 December 2023

Abstract

The goal of this thesis is to improve technical writing through the development of a data-trained model that is able to score student reflective writing (SRW) based on an established standard. Student reflection most closely reflects technical writing in the Computer Information Science department, so the model focuses on the improvement of SRW. Technical writing is an aspect of Computer Science that is fairly absent from the department, so instead of creating an entirely new target of research, it would be useful to utilize the closest form of technical writing in the department. Defining what good technical writing consists of meant that I had to use literature to come up with a definition/standard that I would then implement with my model. Defining good technical writing also meant acknowledging the positives and negatives that come with technical writing. Once exploring technical writing and its intricacies, I developed the model that would use my research as a basis for scoring responses based on a crafted rubric. The model was built by utilizing Natural Language Processing (NLP), specifically the Python library Gensim and its method for word embedding known as word2vec. By exploring the capabilities of different topic models and word embeddings I assert that word2vec modeling would be the best method for utilizing information retrieval in this context. Through this research, a system for improving technical writing was revealed; this system utilizes word2vec processing, proselint, and syntax tree. However, implementation of this system was unable to be entirely carried out due to time issues that arose in the development of the thesis. Future steps in developing the project further and implementing systems that would allow the project to be fully automated are described in the thesis.

Table of Contents

Introduction **4**
Motivation 4
Current State of the Art 9
Goals of the Project 12

Related work **17**

Methods **30**
Evaluation 41

Experiments **43**
Threats to Validity 50

Conclusion **55**
Future Work 55
Ethical Implications 61

List of Figures

1 NLP_Explanation 28

2 Synax-Tree 58

List of Tables

Introduction

Motivation

The goal of this thesis is to improve technical writing in Allegheny College's Computer Information Science (CIS) Department by developing a data-trained model that will analyze student writing in order to show students specific changes they can make to their writing to improve its specificity and clarity. The model will prove that it is possible to automatically grade for good technical writing based upon a targeted rubric. Technical writing is best described by Suzan Last in her book Technical Writing Essentials; she defines technical writing as so"Technical Writing is a genre of non-fiction writing that encompasses not only technical materials such as manuals, instructions, specifications, and software documentation, but it also includes writing produced in day-to-day business operations such as correspondence, proposals, internal communications, media releases, and many kinds of reports" (Last and BCcampus). According to Technical Writer HQ's online guide "What does a Technical Software Writer Do?" In the field of computer science, specifically focusing on software engineering, technical writers have the responsibility to "make user guides, setup instructions, instruction manuals, online training material, release notes, new feature documents, how-to guides, and quick reference articles" (*What Does a Software Technical Writer Do?*). Overall, technical writing requires the ability to be able to communicate a wide range of ideas and think about the different processes that occur in and out of programming. Technical writing may not come easy, as communication itself is a skill, so improving communication would be valuable in order to increase the capability of software engineers and foster an interest into a career in technical writing. Last and Technical Writer HQ make a great point from their quotes; for such an integral part of professional technical development, technical writing and communication practices aren't very

prevalent in the CIS department.

In the department, we heavily engage in technical communication through software development teams and practices such as scrum and agile software development, while the practice of technical writing is something we engage with in certain capacities. From personal experience, the most prevalent form of technical writing in the department mainly involves documentation of code. The department mainly uses the programming language Python, and documentation in python involves commenting. Programming languages have a syntax, and through this syntax is how programmers communicate with their systems and ask their systems to complete a certain task. Comments are ignored by the system, and so they are a form of note taking in a block of code. Comments are prefaced with a hashtag (#) and are associated with a specific function or set of lines that are meant for users to read and thus understand the program. In the department, there is the standard to practice commenting as often as possible in order to make code and programs more readable in the event that they are referenced in the future. Markdown files are another form of documentation that builds user understanding. Markdown files completely ignore Python syntax, only reading syntax when an allocated space is made for it to do so. Space is allocated for Python syntax by surrounding lines with three backticks at both the start and end of the space. Markdown files can be used to go into further depth on the referenced program. Lastly, students are also asked to engage in readings from textbooks in order to deepen understanding. Textbooks offer students a perspective into what technical writing looks like and textbooks are important for teaching; however, anecdotes from students of the department lead me to believe that they aren't often useful.

Technical communication/writing is essential to software engineering when we think of DevIQ's article "Bus Factor." The bus factor; "A project's bus factor (or truck factor) is a number equal to the number of team members who, if run over by a bus, would put the project in jeopardy" (*Bus Factor*). Depending on a single individual, or a group of individuals to carry a project in a software engineering team is not realistic, because as the bus factor explains without them the project is put in jeopardy. This is where the importance of technical writing comes into effect. If an individual is able to document their understanding and know-how of a project they were working on, when new software developers come into the project they will be able to start where the previous developer left off. To put this into perspective, companies that have software infrastructure most likely have an

extremely large number of programs that they have to run; this means that if someone tried to simply look at these programs to build understanding they'd be going through hundreds of thousands, possibly millions of lines of code. Trying to understand these programs line by line wastes time so instead of spending possible weeks or months trying to read through such a large amount of code, they could refer to the documentation to get a better idea and get started much faster. The CIS department practices documentation based on course assignment, as CIS professors don't teach with the same focuses as others. This means that each computer science course doesn't guarantee a concrete practice of technical writing; it is most likely that they will engage in technical writing of some sort, but this isn't guaranteed. From my experience in the CIS department I have had classes where even basic technical writing wasn't engaged with; this may be different from future CIS department classes. In order to improve/set a standard of technical writing in the department, there needs to be greater practice/engagement in technical writing.

While commenting is an important aspect of technical writing, its only purpose in the department is to practice writing efficient and effective code. In order to create a more consistent standard for technical writing in the department, I propose strengthening a form of writing that is closest to technical writing, this being student reflective writing (SRW). This form of writing will be used as a basis for my research.

SRW is the term that I will use to reference student reflections. Students create SRW from completing computational projects and assignments and answering a set of questions meant to gauge their understanding having completed the work assigned. From personal experience, these questions usually ask the student to recall what it is they excelled at while completing an assignment and what it is they struggled with. There may be multiple other questions asked in addition to the two, but these two questions are the crux of what reflection in the department is asking students to do. While SRW is the best form of documentation to engage with, an issue arises from focusing on SRW; SRW isn't necessarily technical writing.

The two disciplines of writing have distinct differences and objectives; SRW recounts a student's experience solving a problem, what they struggled with, and how they possibly overcame that struggle. Technical writing focusing on simplifying complex ideas and making systems and processes more understandable. Despite the differences, the two writing styles aim to inform the person reading them of a complex system/process or help them gain a

better understanding of this system/process. These similarities between the two forms of writing, as well as discussions of the validity of SRW as technical writing allow me to continue development of the thesis with the same scope I am using. Making SRW my focus for my goal of improving technical writing in the department allowed me to get started on looking deeper into how SRW truly works in the department to gauge what I should and shouldn't be doing going forward with the thesis.

The reason that I focus on technical writing in the department is due to my own research in technical writing practices in professional settings. Suzan Last says it best "engineers who are more advanced in their careers spend only 5-10% of their time engaged in problem solving of some kind and 90-95% of their time engaging in related communications tasks: researching, writing and reading reports, proposals, emails, letters, memos; giving or attending presentations; discussing and meeting with colleagues, team mates, managers, clients, and so forth" (Last and BCcampus). Spending so much time communicating ideas shows how much communication is important in professional settings. Someone can have the technical ability to work on projects and complete tasks, but being able to properly and effectively communicate is just as important. From personal experience, verbal communication is something that naturally happens in the CIS department through the software development projects, as well as events such as ITea, a social gathering of CIS majors that happens every Friday. While verbal communication is up to standard, it would be beneficial to focus more on digital communication.

Relevant information about CIS is something that can be easily found. When searching the phrase "news about tech," large companies such as CNN pop up, as tech is currently a closely observed current event. Even though tech is being closely observed, the main focus is on what is interesting and revolutionary, not much information about nuanced tech will usually be included. Nuanced aspects of tech are just as important as the more relevant aspects, because the potential of these nuanced aspects can't be explored without someone taking an interest in them; this is what draws me to technical writing. My interest in technical writing has led me to try and connect it to the CIS department and its practices.

The connection I am trying to make with technical writing and SRW has its issues as SRW has a lot more connection to cognitive learning theory than it does technical writing. This is due to the type of writing these two genres are; I will further explain cognitive learning theory in the relevant work section, but the basic idea is the method in which people learn. SRW

can never really be a learning tool because of how it is executed. When thinking about the reason for reflection at all, we come to the conclusion that reflections are meant to think of possible ways of improving, the focus here being that it is up to the student. After the assignment is completed, there is no real initiative to try and use the student's reflection to help them improve, even though SRW is extremely valuable. Instead of simply having reflections that just exist, I wish to give them a use that could possibly benefit a career path some may have not considered.

When it comes to technical writing, it wasn't something that I had ever heard before from neither students nor faculty. After finding the career and doing further research, I became quite interested in it and I noticed that our department seems to be lacking in terms of technical writing practices. The Society of Technical Communication's article called "Definition of Technical Communication" goes into detail on the services technical communication provides:

Technical procedures and illustrations that clarify steps and identify parts of a product allow users to focus on getting tasks done efficiently, accurately, and safely. Training and instructional materials that teach people new skills make them more employable and productive in their organizations. Medical instructions that are informative and comprehensive ensure regulatory compliance and help patients and care providers manage treatment, improve health, and reduce the costs and risks associated with care. Well-designed websites that are user-focused make it easier to find information and increase user traffic and satisfaction. (*Defining Technical Communication*)

Technical communication has a very big role in conveying information, and as a department it would be beneficial to students to explore this aspect. There is no real initiative or focus on technical writing in the department overall. This is to say that there are professors who do good work in getting students to engage in SRW and fully flesh out what it is that they learned and experienced while solving a computational problem; however, there are also professors who don't really engage that much in SRW. The need for an overall focus reveals itself here as not all students end up taking a course with some form of SRW. As a representative of the class of 24', out of the 5 professors that had been instructing in the department over the course of 2020 to 2023, I will have been taught by only 4 of them. This is significant, because it goes to show that while our department is small, it is also divided based on courses. A student pursuing a specific path in computer science will likely not have a professor that teaches a course that is outside of their

focus. This thesis will allow students who may never experience technical writing to be guaranteed an introduction.

Current State of the Art

The model should work in a way that allows all professors to be able to utilize it, thus establishing a standard of SRW across the department. There will not be a need for professors to change their course if they are able to utilize a tool that can act almost as a grading system. The grading system in our department is called gatorgrader and it is used for automatically grading code. It does this by asking the user to fulfill specific requirements in their programs. Once a student finishes their work, they are able to know their grade immediately, and make changes before the deadline of the assignment in order to try and achieve a higher grade. This is the same system I would want to apply to the model I plan on creating; however, it may not be able to account for the quality of the writing itself.

I am referring to the creation of cohesive sentences in writing. Grammerly's article "Sentence Structure: Learn the Rules for Every Sentence Type," says that sentences are defined by a specific structure, and the most basic structure "requires at least a verb and a subject" (Ellis). If the system asks the user to provide a simple sentence, they can create a sentence that encapsulates the instruction of having a verb and subject, but that doesn't mean that the sentence has to make sense. If a group of words has a verb and subject it is counted as a sentence, this becomes a loop hole in the system. While this is an example of one loop hole, there are many that follow the same trend. While I am wary of those, I am confident that there is a way to allow for loop holes to be dealt with accordingly, and limit the possibilities of loopholes over all.

Writing in computation is a problem that has been challenged for a while and falls under the realm of Natural Language Processing (NLP). Jalaj Thanaki breaks down NLP in her book Python Natural Language Processing by defining natural language first; "As a human being, we express our thoughts or feelings via a language Whatever you speak, read, write, or listen to is mostly in the form of natural language, so it is commonly expressed as natural language" (Thanaki). Once we are given this definition she further defines NLP:

Natural language processing is the ability of computational technologies

and/or computational linguistics to process human natural language Natural language processing is a field of computer science, artificial intelligence, and computational linguistics concerned with the interactions between computers and human (natural) languages Natural language processing can be defined as the automatic (or semiautomatic) processing of human natural language (Thanaki)

Thanaki's definition showcases my reason for using NLP to help with the improvement of SRW; the purpose of NLP is using computation to try and understand and define how humans interpret language. When Thanaki says "computational technologies" this refers to software or software tools that do the work of processing language. NLP can be interpreted as a theoretical concept in the sense that the goal of NLP is commonly not attained. When thinking about NLP tools we use in our everyday lives, we can see why this is the case; tools like auto-fill, suggested words, and suggested responses encompass this. These tools tend to not be able to automatically generate words that perfectly fit the context of a conversation, or train of thought. This is because context is extremely important in reading and writing, and interpreting a sentence in the wrong context completely changes the meaning of the sentence. A NLP tool doesn't always pick up the meaning of a sentence with 100% accuracy because a lot goes into how a natural sentence operates.

Talking and human conversation can be visualized as a continuous conversation that happens over the span of years. A conversation that a person has can be called upon and referenced years down the road. Referencing a conversation from the past means that whoever wasn't in the initial conversation will not understand what is being referenced, and even if they were there is the possibility that they missed the true meaning of the conversation because it could've been referencing a conversation that happened even further back. Part of human communication and language is changing the definition of words or phrases based on one's interpretation and meanings.

While I say NLP can be interpreted as a theoretical concept, there are applications of NLP that establish its usefulness, one of these ways being word vectorization. Word vectorization is the process of making word vectors, word vectors being "an attempt to mathematically represent the meaning of a word," according to Jayesh Bapu Ahire's online article "Introduction to Word Vectors" (*Introduction to Word Vectors*). These word vectors become unique and can be used with great accuracy. Applications that utilize word vectors tend to do things like finding words that have similar meanings. An application would do this by finding words that have similar word vector

values. While these types of applications yield accurate information, it is important to understand that we are trying to use word vectors for information retrieval, which is different from their common use.

Information retrieval is defined by Simplilearn's article "What is Information Retrieval (IR) in Machine Learning?" as "a completely automated procedure that answers to a user query by reviewing a group of documents and producing a sorted document list that ought to be relevant to the user's query criteria" (*What Is Information Retrieval (IR) in Machine Learning? | Simplilearn*). The model would be a form of information retrieval, but it wouldn't be using the methods as defined. The information retrieval would be determining whether something is or isn't present in a presented document. This is to say that we won't be using information retrieval at such a high-level. There is more to stress about the importance of what the tool is meant to provide, and another aspect of is quality.

The thesis I am proposing is important because it would fulfill the aspect of grading in the department that isn't as focused upon, and become a way to check for quality. In the professional setting, whether it is technology related or not, companies are successful when they are able to produce something that is of utmost quality. The adverse effects of work not being produced with utmost quality is loss in "time, work, money, and ultimately, professional recognition" (Last and BCcampus) The losses incurred from a lack of quality is very real; "No one knows exactly how much poor communication costs business, industry and government each year, but estimates suggest billions. In fact, a recent estimate claims that the cost in the U.S. alone is close to $4 billion annually" (Last and BCcampus). Quality is unlike any other aspect of work output; anyone can produce code or manufacture products, what makes code or products worthwhile is quality. Checking for quality is an important aspect in both coding and writing, because while requirements can be filled, where professional standards come into play is through quality.

Code can be produced that fulfills a certain requirement, but from a professional standpoint, it has to fulfill that requirement as efficiently as possible. When coding, there are procedures that check for quality linting such as Github actions and linters, but they aren't enough to stand on their own. The reason for this is because code and writing are parallel in the sense that they are both languages. When dealing with language the possibility of not using language efficiently is simply inevitable. For coding there may be methods to achieve your goal that are faster or more efficient, just as in writing there may be ways to be more specific and clear through specific

vocabulary use. When it comes to coding, there is more development being done in order to automatically make code more efficient, but just like the thesis I am proposing it is nearly impossible to do this perfectly. What I am proposing is the documentation side of this effort; a model that ascertains the quality of a document. By now, the importance of SRW is clear, another example of technical writing in the department would be Senior Comprehensive Projects themselves.

For a student that has avoided writing in the department for their entire undergraduate career, having to write a five chapter summary of their work will be a rude awakening. There are valid and necessary reasons for being able to write about coding and computation, and this can not be argued. Computer science is a high-level thinking discipline that requires a lot of studying and effort in order to excel at. There are thousands of textbooks and documents that highlight the process of learning to code or use functions in programming, and these exist for a reason. Before coding on PCs became the standard, code was more often thought out on pen and paper. The need to write out ideas has always been an essential part of coding and programming, so cutting that aspect out is actually detrimental to the development of students in the department. This thesis serves as the foundation for advocating the importance of writing in the department.

Goals of the Project

The objective of this thesis is to create a model that can arguably be considered for grading SRW in the department. The hypothesis I am working with is that good technical writing has certain specifications that make it good. In order to understand what I mean by this I will frame how this idea should be understood. Writing that is meant to fulfill a certain objective will do so at different levels of specificity. Explaining ideas requires writers to detail their process to the best of their ability, but writers don't always use the best words, phrases, or structure to detail their process. This is due to the level of writing that an individual may possess, as well as a level of explanation that an individual may possess. Some writers can explain a process in great detail, while others may struggle in specificity, or proper explanation. In terms of the thesis this may seem like a bad thing, but it is more realistic to have SRW that isn't as efficient as it could possibly be. In order to work with this data, I would have to produce it, as I either have to find SRW that

happens to be an open resource, or utilize my own, which doesn't allow for much variation of ideas.

In order to collect data, I will utilize a study group who will participate in a coding exercise and describe their experience in detail in order to analyze that data and pinpoint specific words or phrases that may lead to ambiguity, or that may not be as specific as possible. I believe that highlighting these words and/or phrases and then revising them can allow writers to create better reflective writing, and thus help them voice their understanding. This tool completes the objective I have for the project because it gives students a method of identifying areas in their writing that could use improvements. The value of this is that students will be able to better understand what exactly they are struggling with, also allowing them to voice that misunderstanding to someone else. The objective will be mainly focused towards students in the department as this thesis is being written; framing the time that I hope to achieve this objective is important because of the technology that is currently available. It is important to acknowledge the upsurge in generative AI as this thesis is being written due to its various uses, one of these uses encompassing the basis for my project.

The most notable resource that is utilized currently, ChatGPT, has functionalities such as improving writing by offering more ideas and context. While this is convenient it essentially goes against my objective. ChatGPT is able to generate responses to computational problems, but it doesn't provide great insight for individual problems and solutions. ChatGPT can tell a student what they did wrong and how they are able to fix it, but in doing so the student doesn't have to think for themselves about the project they are working on. A common issue with ChatGPT is that it conflicts with college curriculum, especially with coding and programming. When a student tries to use ChatGPT to generate code, the AI does so by focusing on what it is being asked to do, and not the context in which it should be done in. For example, if you tell the AI to create a program that creates a calculator, it will create a program that does so, but that program may be problematic. At times the program may not be able to run, but probably one of the most problematic issues is that the code may just be completely incorrect. ChatGPT can't do context without needing to be queried to do so, and because of the need to be so specific, a user would end up needing to spend a lot more time trying to tailor the AI's answers to a certain goal that they are attempting to reach. In order to avoid this, it would be best to work with a model that doesn't need to be queried or handled that extensively.

In order to achieve the goal that I have for this project, it is important to understand how exactly I am going to enact my project. My project can be broken up into different parts that I need to fulfill for the final end result. The project is a double thesis meant to fall under the CIS department as well as the English department. The process for the CIS department is as follows: I will create a coding exercise, using Google Forms, where I will utilize an esoteric programming language, that I will distribute to students who have taken CMPSC 100 or have engaged with some form of Computer Science prior to coming to Allegheny College. I will disclose the specific esoteric programming language that I am using after the survey is distributed to students and closed for the sake of hiding it. I don't want students knowing what the programming language is, as they will be able to look it up and gain more knowledge. I want students to learn about the program from a reference sheet that I will provide them. The reason for this is to simulate the learning experience, as students study and are required to take tests based on what they were able to pick up from studying. I will utilize the esoteric programming language by asking users to learn the language and solve two problems that require users to have some understanding of the language. The problems will not be difficult; they will be quite basic as the objective isn't to confuse the participants, but to gauge how well they understood the material they learned and if they can reflect that understanding. The goal of the survey isn't to have users solve the problem correctly, but for users to reflect on the process of learning the language in order to solve the problem. That data is what I will analyze by utilizing Gensim's Word2vec, in order to find descriptive words that participants of the dataset used. I will report this data in order to show how SRW looks like and what students pick up from the problem.

On the English side of the project, I will do research into technical writing and what makes good technical writing in order to make a rubric. While this is the goal, there is more that needs to be done in the event that aspects of the method don't play out as I want them to. I am collecting SRW for my project in order to better represent what I am trying to convey; by actually using SRW I am making a better case for the necessity and usefulness of the model. However, there is the possibility of not enough data being collected due to students not taking my survey. In addition, there is the possibility that the collected data is unusable due to students simply putting N/A in the given margin. In order to account for this, I also plan to use ChatGPT, an open AI-system, in order to produce synthetic data that I know will be

able to be analyzed. To analyze my data I chose to use Word2vec instead of other NLP software like Spacy because of the accuracy of vectorization. On the English side, I have been doing research into SRW and have found a few sources that can help me in developing the rubric and a paper that will detail the different aspects of SRW that I may not go as in depth on in this thesis. There is a lot more that can be talked about when it comes to the method in which I am implementing this project such as the specific esoteric language I'm using, and how I will be storing and handling the data, but that will be featured in the methodology section of this thesis.

The project hopes to contribute to the field of CIS and English; the project would be a contribution to the capabilities of NLP through the development and usefulness of the tool.. As a contributor to NLP, the project is exploring how useful NLP can be in giving writers useful feedback and determining the conveyance of their ideas. to the specific form of writing and conveying of ideas. I derived my idea from researching work from Allison Parrish and Jalaj Thanaki. Parrish is a software engineer more known for her creativity in coding and design, and Thanaki is a data scientist and an author. I first discovered Thanaki's textbook in which she introduced NLP and how it is used when pertaining to Python. From reading through her textbook trying to find a way to use NLP for my project I discovered her introduction to Word2vec. She explained in great detail what Word2vec was and how it works and I thought that it would be the best method to implement what it was that I was thinking of doing. After discovering Word2vec, I decided to do more research beyond her textbook and that was when I discovered Allison Parrish's Github repository in which she explains Word2vec and shows how she uses it for poetry. Both of these contributions are what steered me in the right direction, and I want to take what I learned from their explanations and create something that can contribute to them. After discovering these sources and fleshing out my project more, I started looking into literature related to SRW. At first, I wasn't sure that there would be much research done into SRW, but I was able to find some sources that began leading me in the right direction. The first paper that I found, "Reactions of Students to Reflective Learning in a Technical Report-Writing Course" by Belinda Ho, showed me that SRW is a valid study that has information. While both sides of the project are geared towards their own respective studies, the project as a whole contributes to the idea of cognitive learning theory. This idea asks us to think about thinking such as how our thinking becomes influenced by internal and external factors. Cognitive learning theory is an idea that is

most closely related to psychology, but instead of focusing on that side of it, the project is mainly focusing on the editing/changing of ideas, making it more English centered. Understanding the scope of the project has allowed me to properly research and come up with sources to base my research on.

The thesis will be organized based on the five different sections; the introduction, the related work, the method, the experimental results, and the discussion and future work section. The introduction section has featured what the project is that I'm working on and some overview about motivation, validity, etc. For the related work section, I plan to do a deep dive into the sources that I have mentioned. Because I am doing a double thesis, this section will be the best to start talking about the information I get from research and readings. I am trying to find good SRW standards from these readings, and so far I have found a few sources that are good to start with. I have also been recommended by a lot of other sources that should help me writing this section. I also want to explore cognitive learning theory in this section in order to further flesh out what this project is working with. This section will mainly be detailed with the specifics of systems I am using, meaning that there won't be a lot about the software I am trying to develop, but the surrounding information about that software. I will talk more about the software in my methodology section. In my methodology section I plan to further explain how I am going to collect data, analyze data, and present data to the reader. I talked a bit about what it is that I want to do, but I haven't gone into much detail in order to leave more information for that section. The experimental results section will feature the results of the model, and the final discussion section will feature what was learned from the thesis and possible future research. When thinking about future research, I would want to look more into AI and Machine Learning so that I can make the model more flexible to use.

Related work

The term SRW is one that I came up with to focus on the objective of this project; however, when looking for sources related to SRW, the topic diverges into various other focuses. SRW examines reflective writing and learning, cognitive learning theory, communication skills at professional levels, etc. and because of this, research tends to be divided based on what is specifically being addressed. There is also the importance of understanding the aspects of good technical writing in order to incorporate the tool to reflect those aspects. There are some cases that examine student reflection that conduct in experiments that are similar to my proposed experiment; quizzing students based on a concept and examining how they go about solving the problem. These works of literature provide significant information in cultivating the importance of SRW.

Before getting into SRW and what it asks of users, it would be helpful to highlight the importance of literacy in code and programming. When thinking about programming, writing for human readers isn't something that people usually think about; programming is mainly associated with development of code and software tools. However, the overall goal of this project is to bring writing and thinking into what is generally considered by students a field that doesn't involve those aspects. The importance of literacy in programming can be further explored from Annette Vee's Coding Literacy: How Computer Programming is Changing Writing.

Vee argues that a computer science education and computer literacy is something that is essential to current societal education based on its relevance and prevalence in society today: "Arguments about education are always ideological: they reflect the values of a particular society and moment. Arguments about literacy education may serve as uniquely effective ciphers of their time because of literacy's perceived central role in nation-building and individual success" (Vee). When thinking about our society at

the moment, technological innovation would be a reasonable reflection of our current society. Technical advancements have been occurring for the past few years, and access to the internet is increasing tremendously. According to Statista, a website that displays numerical data for a number of topics, there are around "5.35 billion internet users worldwide which amounted to 66.2 percent of the global population. Of this total, 5.04 billion, or 62.3 percent of the world's population, were social media users" (Petrosyan). More than half of the world's population engages in technology, which shows the importance of possessing knowledge for using computers. Vee advocates for the importance of literacy and its flexibility, and criminalizes illiteracy:

Literacy's rhetorical malleability—the fact that it can be effectively pinned to any number of skills and practices, and the fact that people are motivated to do so—stems partly from its rich metaphorical resources [...] Illiteracy is a disease, a link to criminality, a drain on the economy, and a cause for joblessness and individuals being held back from reaching their potential (Vee).

Vee's perspective of literacy comes from a well-researched delve into the importance of literacy in society and history. Literacy is something that is used to bring people together, whether it be scholarly knowledge, religion, or other factors. Literacy was something that wasn't previously available to everyone; it is something that even in today's society isn't available to everyone, so we see how literacy becomes an essential tool for those who are able to utilize it. Programming followed a similar pattern as from its initial conceptualization to advancements, programming became more and more embedded into individual lives. Just as literacy is a tool we should be utilizing, programming is just as important to utilize in our everyday lives for its many benefits.

The essence of modern technology is programming; all technological tools run on some form of program that is written. While programming doesn't appear in everyday life, it is there whether you know it or not. With how ingrained technology and the internet is in our current society, the benefits of knowing about programming can be helpful. Every industry deals with technology in some way, and if there is an industry that doesn't it could easily be integrated to involve technology. Overall, it is something that we now rely on quite heavily, making it an important tool that can be applied to almost any field of study.

Literacy and programming are connected because, as Vee puts it "Programming is writing because it is symbols inscribed on a surface and designed

to be read, but programming is also designed to be executed by the computer. Because it represents procedures to be executed by a computer, programming is a type of action as well as a type of writing" (Vee). The emphasis on programming being writing is what drives the project and that is to further deepen this relationship between programming and writing. Programming is only one form of writing, but with it comes actual writing itself, and so the two need to work in tandem in order for a truly deep understanding to be made. As programming and writing are in the same family of knowledge, bridging the gap between the two will only further enhance our understanding of both writing and programming.

The bridge between writing and programming is one that should be acknowledged, and within the project how this bridge looks like in a real world example will be revealed. The experiment that I plan to carry out is one that has been done before, so it would be beneficial to understand what this experiment looks like as conducted in larger and more respected environments. A significant example of this is Belinda Ho's article "Reactions of Students to Reflective Learning in a Technical Report-Writing Course" published in the journal English for Specific Purposes. The article recounts a similar experiment with the hope of finding the difference between how two groups of students engage with reflective learning; The general group will learn a concept and only engage in low-level reflection, while another group will learn in the same concept while engaging in much extensive and higher-level reflection. This study asked the question of "how students reacted to reflective learning" as they are looking towards a concrete answer as to which is more effective, more or less reflective learning (Ho).

The study's significance came about based on what was observed about the students and the learning process they went through. The study was quite descriptive as the students were closely examined in order to understand how exactly they were conceptualizing and practicing reflective learning and understanding. The study brought up how reflective learning is carried out at two levels "The metacognitive level and the cognitive level" which shows that a lot more is being done in reflective learning than what it seems (Ho). Cognition is the process of gaining knowledge and learning, and metacognition is how we think about this process, specifically how we think. When looking at SRW the goal of its existence is to help students improve by acknowledging their own work, but doing this also invites a much deeper line of thinking and acknowledgement that is important to the process of learning. To acknowledge one's own work is to try and look at their work from the perspective

of someone else; people generally aren't able to see the flaws or patterns in their own work and so another perspective is needed.

It is best to think of this process of learning from an experience as something that is much larger to the student than just the individual experience. Ho describes it as so, "The new experience will become part of the 'overall experience,' which is a source from which they can draw when they have to go through another existing experience, such as being given another task to do in the future" (Ho). While Bo is acknowledging the importance of reflection by specifying that the experience of greater reflection can be something students draw back to when experiencing another task, Bo isn't acknowledging this can go even past simply completing tasks. Bo essentially opened the door to the fact that asking students to reflect on their work is something that is much larger than it seems.

School can be considered a pressuring environment due to what is expected of students. Students are expected to produce a level of work that is of a certain quality, and when they are not able to do this they are penalized in ways that can be demoralizing to the overall goal of learning. According to Kathryn Watson's article "School Anxiety; Causes, Symptoms, and Treatments" the most prominent causes of anxiety in school are "Social anxiety, bullying or cliques, academic anxiety or testing, past or ongoing trauma, generalized anxiety disorders, chronic health conditions." (Watson) Without getting into the structures of school that attribute to anxiety, it is important to acknowledge that academia, testing, and trauma are factors that cause students anxiety. It must be acknowledged that school is an environment in which a student's knowledge and intelligence is closely examined and criticized. While school and universities differ, the fact remains that the same sort of methods are being employed, and for students that attend universities with the hope of pursuing a career, that same pressure can be just demoralizing as attending high school or elementary school. The significance of this is reflection; students feel pressure being examined by others, so asking students to reflect upon themselves may offer new, unique feelings of pressure that are important to acknowledge when conducting experimentation. This is to say that Ho's experiment, as well as similar experiments are causing thoughts and changes to arise in students past the study.

Thinking about the study led me to have a direction in which I could research more relevant information to note when examining students behavior and language, but before moving on to that it is important to know about the results of the study. Students in the study reported not liking "reflec-

tive tasks at the metacognitive level," this is the level of thinking that I am hoping to examine in my own experiment (Ho). The study also states that" reflective tasks at the cognitive level are not unwelcome by either group" (Ho). Why the case may be that students dislike this type of learning can be explained by the next quote in the article: "Reflective groups that involve peer participation, such as group discussions, are especially popular" (Ho). This is significant to the contribution of what is known about cognitive learning theory in the article "Bandura's Social Learning Theory & Social Cognitive Learning Theory" published by Razieh Tadayon Nabavi and Mohammad Sadegh Bijand.

Nabavi and Bijand explore the history of cognitive learning theory and credit its discovery and development to the man known as Albert Bandura, a psychologist with a Ph. D from the University of Iowa. Albert's study into cognitive learning theory is broken down into two categories; Social learning theory and and social cognitive learning theory. It is important to explore both because these theories relate to the classroom learning environments and how students learn best. In social learning theory, Bandura states that:

Social learning theory is increasingly cited as an essential component of sustainable natural resource management and the promotion of desirable behavioral change. (Muro & Jeffrey 2008). This theory is based on the idea that we learn from our interactions with others in a social context. Separately, by observing the behaviors of others, people develop similar behaviors. After observing the behavior of others, people assimilate and imitate that behavior, especially if their observational experiences are positive ones or include rewards related to the observed behavior. According to Bandura, imitation involves the actual reproduction of observed motor activities. (Tadayon Nabavi and Bijandi)

Social experience influences behavior and development, so from the quote we can see why the group enjoyed doing group activities more than working on their own. The learning process that is most common in schooling, from elementary schools to colleges and universities, is teaching a large number of students at a time. This is the typical setting of a classroom, and it turns out that doing so actually helps students learn better. This is something that can be directly observed in a classroom with discussions. Discussions are meant to allow different students to offer facts and/or opinions about a matter in order to build a better understanding overall. The 'overall' refers to the overall knowledge of the classroom/environment; schools and learning institutions strive to ensure that a large number of students can reach an

agreed upon understanding and now we see that classroom discussions aim to satisfy this desire.

Knowing that students tend to favor group activity and discussion can help in determining possible results from gathering information from the study; the possibility of a lot of information being gathered being either incomplete or inexhaustive. Despite this fact, collecting data is important because the goal of the project is to make reflective writing better by acknowledging these issues and addressing them directly; individual students won't be addressed about what they submit, but overall data will be addressed. The data is meant to help learn about students' learning; this is done based on what they write down after completing the exercise.

Social learning theory leads into the other idea of social cognitive learning theory as "Moreover, Social Cognitive Learning Theory places a heavy focus on cognitive concepts. It is also focused on how children and adults operate cognitively on their social experiences and how these cognitions then influence behavior and development" [article]. This quote ties together Bandura's idea of how students learn, which is from social experiences.

The learning experience is directly related to the process of learning and how it made someone feel while they were doing so. Children and adults operate in the same way, that way being that they themselves have to want to learn. In colleges and universities, this want may persist a bit more being that college is something that a student chooses to enroll in, but the reluctance can be just as prevalent. Learning from social experiences would be important in order to actually get students to improve, so instead of complicating the learning process and experience, I feel that it would be more beneficial to offer a proven method to enhance student engagement; a rubric.

Rubrics are a set of guidelines that allow the student to have an idea of what they need to encompass when completing an assignment. Rubrics are mainly present in writing assignments, and they truly help students shape their writing and understanding of what they should and shouldn't include. Writing styles such as MLA, and APA are examples of guidelines that help students with writing when thinking about research. They can loosely be used as examples in creating an effective rubric, but more would need to be done to cater the rubric to the intended audience of university students. The thesis will explore an experiment in order to gauge student's writing and utilize a tool to offer improvements, and what could also be helpful would be a rubric that could be used to help students write more and be more descriptive.

When tackling this task, it was useful to refer to another experiment that utilized a rubric for their automatic score grader to see how development would take place. Leonora Kaldara's, Nicholas R. Yoshida's and Kevin C. Haudek's article "Rubric development of AI-enabled scoring of three-dimensional constructed-response assessment aligned to NGSS learning progression" describes that "the key to successful implementation of NGSS in practice lies in designing high-quality assessments'' (Kaldara). From Haudek I decided to spend more time on the creation of the exercise. For the exercise, I wanted to have just a few guidelines that would ask students to write up to a certain word count, but try and answer the questions as best as they possibly could. The hypothesized dislike for SRW was the reason for this and so I wanted to understand how much a student will write without being prompted to, and while the study hasn't been conducted yet, it can be assumed that students who don't have a good time with the exercise will not write a lot. There are also no restrictions that are keeping them from writing outside of the prescribed word limits, meaning they can write less than expected or even just stop the exercise altogether. These results are useful nonetheless because it shows that when completing an assignment without much direction and that can be frustrating, students may opt out of doing it. Examining students' actions while administering this experiment is important because it shows that a tool that can offer feedback on enhancing reflection through further description and explanation to students on reflection would be worthwhile, as negative results were collected. Moving on to the rubric itself, we first need to understand the context that the rubric is going to account for.

SRW is quite simple, in the sense that it is mainly asking two questions; what did you excel at, and what did you struggle with. These two questions can be asked in numerous ways, but they serve the overall purpose of asking students to think about what was easy and what was hard. This is only the first level of thinking that students are asked to engage in with SRW; identifying what one excelled at and underperformed at are good, but in order to reach the overall goal of improvement more thinking needs to be done.

We've identified that group learning is most enjoyable when learning, but there comes times when a group discussion or activity causes everyone to become silent. This can either be due to boredom, fully exploring the topic, and worst of all, not knowing what they are talking about. Group discussions are enjoyable because students feel as though what they have

to bring to the table will further the discussion/activity and cause critical thinking to happen, but when they have nothing to bring critical thinking can not happen. More often than not, students will feel less inclined to say anything as they are worried that their guess will be incorrect, and this is when group learning becomes a hindrance. Students are more comfortable expressing their lack of knowledge in a private, enclosed space where their ignorance will not become known to other students; this is usually through a conversation/communication with a professor. It's much easier to string together the bits and pieces of knowledge in one's head on paper, than it is in a public setting when all eyes are one one person.

Once a student understands what they once didn't, they can work towards improving their skill in discipline, which is usually the last stage of student reflective writing; what the next steps are. This last step isn't as important in the context of this experiment, so the rubric will mainly be focused on asking a student to be specific in describing what it is they did well on, what it is they need to improve and how they plan to do so. With these in mind, this is the rubric I developed that would encompass this.

Criteria		Grade
	Have a word count of 50 - 100 words	
Response	Be told in first person perspcetive including details about the learning process itself Should answer the question of what was understood and what was a struggle in complete sentences.	
Understanding	Recount which concepts	

Criteria		Grade
	were understood	
	Recount which concepts were understood	
Process	Describe a timeline on how you went about the exercise. Include first steps to last steps.	
	Ensure that the steps are easy to follow.	
Readability	Ensure that your response is using clear and cohesive words and sentence structure.	
Specificity	Use vocabulary and keywords that appears in the reference document.	
	Ensure that the	

Criteria	Grade
response is using standard English punctuation, sentence structure, usage, and grammar.	

This rubric should allow for students to have a better idea on what it should look like for them to further detail their learning experience. The students will not have the rubric available to them when they complete the experiment, but their responses can help in developing the rubric to be more useful. Moving forward, the experiment and its impact would be important to begin discussing.

While we know how the experiment will be carried out, what is most important has to be the tool that will be used for retrieving information from the data. Information retrieval is a study of its own, and to better understand how information retrieval will be used in the context of this project, it would be beneficial to look at the article "Using Word Embeddings for Information Retrieval: How Collection and Term Normalization Choices Affect Performance" published in Conference on Information and Knowledge Management '18: Proceedings of the 27th ACM International Conference on Information and Knowledge Management by authors Dwaipayan Roy, Debasis Ganguly, Sumit Bhatia, Srikanta Bedathur, and Mandar Mitra. Roy, Ganguly and co.'s article delves into information retrieval using word embedding, specifically looking at fastText "a word embedding model that utilizes sub-word information for learning term representations" and word2vec, a commonly used word embedding model (Roy). For the project I will be utilizing word2vec for word embedding, so it would be helpful to first further explain word2vec and what it has to offer. I will do this by referencing Thanaki's book.

Thanaki goes into depth on different aspects of natural language processing (NLP) such as gathering corpus, syntactic and lexical analysis, sentence tokenization, lemmatization, etc.. What I want to focus on is her introduction to word2vec and how exactly it works; "Word2vec is developed by using two-layer neural networks. It takes a large amount of text data or text corpus as input and generates a set of vectors from the given text" [thanaki_2017].

Neural networks are an idea that is mainly present in psychology that involve the brain, but its presence here changes its original definition. Neural networks are connected and signal to one another; there are three layers in total. There is the input layer, the hidden layer, and the output layer; the input layer would be where words are input, the hidden layer would be where words are manipulated, while the output layer is where words would be output. It is important to know that in all of these layers there should be the same amount of words present in all layers as a data would go through them.

Thanaki gives a good introduction to word vectors and neural networks, and from Dimitrios Alikanoitis's article "Automatic Text Scoring Using Neural Networks" I am able to learn more about how neural networks will be used in this context. Alikanoitis talks about Long-Short Term Memory Network (LSTM), which is mainly used "in order to embed these sequences in a vector of fixed size" (Alikniotis). LSTM models can be trained as is, or if they are made into bi-directional LSTMs (BLSTM). According to Alikanoitis "The two BLSTMs trained with word2vec embeddings are among the most competitive models in terms of correlation and outperform all the models, except the ones using pre-trained embeddings" (Alikniotis). Development of the rubric requires the need to correlate concepts to responses, so understanding that word2vec is able to train models for correlation the best solidified my decision of using it going forward. Moving back to Roy and Ganguly's article would be better in building understanding of how exactly I will utilize this. As I've stated before we are utilizing word2vec and its use of word vectors for information retrieval. Information retrieval involves the types of analysis I mentioned above; syntactic analysis, lexical, and much more. The main idea of information retrieval is looking at large data sets and extrapolating a specific piece of information from it. In this case, I plan to utilize information retrieval in order to retrieve words, specifically words that aren't descriptive.

What I mean by specificity is that they should be using more descriptive words that are unique and that are able to have meaning easily extrapolated from them. Reflection is the focus of my study, and because of this I would hope that the prospects of reflection can be improved from the study. Reflection is a word that defines itself in its name, and that is to reflect; in order for a student to improve, or anyone to improve in general, they must reflect on what exactly they need to do better. Reflection isn't just a singular process though, the most efficient way to reflect is to be able to voice one's concerns and essentially have a dialogue. Going back to Belinda Ho's

experiment, we see that people enjoy reflecting in a larger group than they do on their own. In order to be able to do the better of those options when it comes to reflecting would be to have the ability to properly voice what exactly it is that students are reflecting on, and descriptive words is the way to do this. Generating synthetic data is an aspect of the project, and one thing I noticed about generating no descriptive reflections is that nothing can really be said or done to ask the user to improve. If the user doesn't know what exactly it is they are struggling with, the best one could do is ask the user to explain or do the process over again and try to build a better understanding so that they may describe what the issue is. What would be best for a student's reflection would be for it to be understood immediately in order to begin the process of working to improve. While one may think that this isn't something that can be done by a software application, they would be surprised to know that this isn't the case.

Natural language processing isn't a new concept, it is something that has been studied and worked on for years and it is responsible for a few systems that we have in our society today. Grammar checks, spelling checks, word suggestions, and now generative AI are all a part of NLP, and depending on who you ask they can be pretty useful. The best way to explain how NLP words would be to reference Allison Parrish's graph from her github repository "Understand Word Vectors":

	START ___ was	it ___ the	was ___ best	the ___ of	best ___ times	of ___ it	times ___ was	was ___ worst	worst ___ times	of ___ END
it	1	0	0	0	0	0	1	0	0	0
was	0	2	0	0	0	0	0	0	0	0
the	0	0	1	0	0	0	0	1	0	0
best	0	0	0	1	0	0	0	0	0	0
of	0	0	0	0	1	0	0	0	1	0
times	0	0	0	1	0	0	0	0	0	1
worst	0	0	0	1	0	0	0	0	0	0

Figure 1: NLP_Explanation

This graph showcases how word vectors are created, but just as important, the graph also showcases how sentences are understood by software applications. Software applications break down sentences to examine each word in the sentence. Applications would then look at the word and see what is surrounding the word in order to get an understanding of what the sentence is trying to say; context clues is another way to think of it. The

0s and 1s that are present in the graph are a visualization of vectorization. This visualization represents the word "it," and these vectors are used in different mathematical equations in order to produce desired results. Roy and Ganguly uses these formulas for finding words that have similarities between them:

The results of the experiment were that "word2vec embeddings, in general, helps in achieving significantly better performance than using embeddings learned from an external, un-normalized corpus" (Roy). This observation is beneficial because my project plans to use a normalized corpus, meaning I will be using the better of the two. For definitions sake, normalized corpus is essentially data that is being compared is the same size in terms of characters. For example, if I am working with two data sets, it would be a normalized corpus if both those data sets contained 600 characters. It is important to note that "this trend of getting better IR effectiveness with word2vec embeddings learned from normalized, target collections does not held when fastText is used for learning embeddings" (Roy). This is important for future research and use of the tool itself. It would be best to use normalized data when using the software application, as it would not be as efficient to use word2vec for non-normalized corpus. Roy and Ganguly confirm my use of word2vec, so it is a process that I am justified in utilizing.

The evidence I presented helped solidify my reasoning and method of experimentation, and through my research I was able to push on with the thesis.

Methods

Developing the thesis required combining the aspects of a typical CIS thesis and and English thesis. While the methods for developing the double thesis seem to have been different, a lot of what was done was quite similar in terms of research and explanation. There are concepts in both disciplines that would be important to acknowledge before getting into the combined methodology. Starting with the CIS side, development of the thesis included a lot of research into how topic modeling is used with NLP processing, as well as the validity and extensive reach that NLP has in its current deployment.

Exploring Natural Language Processing and its capabilities was the first step in the development of my project. The project utilizes NLP as the main method of development, as it is the core software concept that I am utilizing, and for most effective usage I had to be sure that it was a viable tool that could provide results. My exploration of NLP stemmed from reading a previous senior's Comprehensive Project; that senior's project tackled the capabilities of NLP at the time the thesis was written. Due to the fact that I am unable to find that same thesis again, I am only able to use anecdotal evidence to review the goal of the thesis and its content. The student was answering the question of whether or not NLP can be used for enhancing writing and sentence structure. The student initially didn't believe this to be the case, but after doing research was proven wrong. When initially reading the project I also disagreed with the student's assumption that NLP couldn't complete the task; my reason for disagreeing was because of how developed NLP is currently. My hypothesis and the senior's conclusion showcases how NLP has developed since the conceptualization of that senior's project.

Discovering the senior's thesis was beneficial for developing my own thesis, as the idea I was hoping to explore couldn't be carried out without the confirmation that NLP has the ability to carry out my idea. Reading through the senior's thesis also showed me that NLP had capabilities that weren't fully

realized at the time based on the senior's hypothesis. From this discovery, I knew that my thesis was a lot more feasible than I had initially realized. After discovering this thesis, and doing further research into the discipline, I discovered more about NLP and realized that I had experienced NLP in ways that I didn't even realize.

Discovering NLP and Thanaki's textbook that explored NLP was the catalyst for discovering how exactly I was going to develop my project and that was through the use of word2vec. I discovered word2vec and began looking into how useful word2vec could be, by reading about how it works and its possible uses. Thanaki described word2vec with great detail which went over its capabilities, uses , limits, etc. Knowing more about word2vec persuaded me to try and use the software in the development of my project. Thanaki's description, while being helpful, was more centered towards mathematical applications of word2vec; I wanted to use word2vec in a different way, which is what led me to Allison Parrish's use of the system. Allison Parrish and her explanation on how word2vec was more focused on extrapolating words for essays/poems and manipulating them in different ways. Her main focus was using word2vec artistically, and this is where I knew that I could use it in the realm of literature.

What interested me and led me to the overall idea for the project was the use of a function she developed called the `closest()` function. This function was written in Python and was used to find the closest word in a corpus to other words that she specified. She inputted the word red and got returned a bunch of different objects that were red. For example, from this function the program returned words like "fire engine" and "tomato" (**?**). Seeing this made me realize that this process should be possible for more than just colors; I wanted to explore how word2vec could be used to retrieve more useful information.

At first I wasn't sure how exactly word2vec could be translated into my ideas, so I had to look more into how word2vec was used in the context of writing and sentence structure. This is what led me to discovering the Stanford Core NLP. The CoreNLP is a website that breaks down a NLP system that was developed at Stanford. This system is able to break down a sentence and extrapolate information such as parts-of-speech, named entities, dependency parsers, and co-references. Named entities refer to nouns, dependency parses refer to adjectives and prepositions, and co-references refer to relationship between pronouns and nouns (Making the connection to a name and a pronoun) (Jurafsky). This was the most concrete example of

how word2vec could be used for writing and in discovering this I was able to ascertain the feasibility of the project. Discovering these different aspects allowed me to get a clear vision of how I would begin development of my project, and with this information I was able to get started.

The program I developed would be in the Python programming language, as it is the language that I have worked with the most, and I found a lot of information about word2vec in relation to Python, mainly Thanaki's textbook and other previously mentioned articles that go in depth on word2vec. I had considered using the R programming language as the project would go on in development, as a lot of what word2vec is used for is data manipulation. Whether or not I actually use this will depend on the future development of the project.

In order to start programming using word2vec, I first had to do an install of Natural Language Toolkit (NLTK). NLTK is "a leading platform for building Python programs to work with human language data. It provides easy-to-use interfaces to over 50 corpora and lexical resources such as WordNet, along with a suite of text processing libraries for classification, tokenization, stemming, tagging, parsing, and semantic reasoning, wrappers for industrial-strength NLP libraries" (NLTK). Python libraries are developed by fellow programmers and make the application of topics much easier to do by providing the framework for other programmers to work with. Without Python libraries, programmers would not be able to develop the numerous technical systems that we utilize today. Installing NLTK allowed me to have access to word2vec modeling libraries, as well as libraries that would do other kinds of topic modeling. The NLTK library is extremely large; however, I don't need to utilize everything in that library, so I specified what exactly I needed through the import statement:

```
# Code for importing NLTK
import gensim
from gensim.models import LsiModel, KeyedVectors
import gensim.corpora as corpora
from nltk.tokenize import sent_tokenize
from nltk.corpus import stopwords
```

The NLTK comes from Gensim, "Gensim is a free open-source Python library for representing documents as semantic vectors, as efficiently (computer-wise) and painlessly (human-wise) as possible" (Gensim). Gensim includes word2vec and other algorithms that use vectorization, so

we need to import it in order to get started with programming. More installation needs to be done in order to fully be able to work with the programming language and dependencies, and that is through installing the software on our terminal. As I've previously stated, installation is necessary for working with whatever content isn't readily available on one's device. These commands are meant to be run in the terminal in order to be able to fully utilize the libraries:

```
# Terminal commands for installing package
pip install gensim
pip install nltk
```

Discovering what is needed for installation is important, so that others who wish to run the code that is developed knows how to get started in using and developing the program. After installation came figuring out how exactly I wanted to utilize word2vec. The process for this began with looking into a beginner code example of how word2vec is used. I was able to find the basic necessities of using word2vec and what results would be yielded. The beginner code helped me understand how tokenization of data worked, which is the main step to beginning processing with word2vec. Word2vec creates vectors to be used for topic modeling, but in order to begin that step, you must tokenize the data that is being input. Tokenization is "a process by which a piece of sensitive data, such as a credit card number, is replaced by a surrogate value known as a token" (Gartner). Once the data has been tokenized it is able to be used for topic modeling, and how word2vec does topic modeling is through.

After understanding tokenization came the crux of the project, which was topic modeling and using an efficient topic model. When first developing the program, I only utilized basic semantic analysis with word2vec in order to show how I would try to complete the project. This basic example essentially just compared two words and found the similarity between the words based on their vectors. This basic display of analysis served as my prototype, as I hoped to develop the model more and make it be able to include more relevant information than simply comparing two words. My focus is SRW, meaning that I would need to be able to handle multiple sentences and even paragraphs being input. From my basic development to my current development there was a lot more included in order to increase complexity and ability of the model.

After my prototype, I began thinking about how I would develop the more advanced version, and in doing so I discovered the need to work with data and the need to find topic models that use word2vec as infrastructure while doing more advanced modeling. Starting with the discovery of data, I realized that because I would be focusing on SRW I should probably collect SRW in order to be used for testing of the topic model. This began my method of data collection, encompassing both real data and synthetic data

In order to gather data I would need to complete the process of applying to the Institutional Review Board (IRB) in order to have my collection of data approved. Going through this process made me acknowledge how exactly I was going to collect data. I wanted SRW, but I could collect that without having students answer some sort of question and make responses that I would then be able to utilize. This led to developing my coding exercise that would utilize the confusing nature of an esoteric programming language. An esoteric programming language can be described as a programming language that doesn't use conventional methods of identifying characters/processes. In programming languages like Python, someone who doesn't use Python can read the code and understand, or at least have some idea, of what the program may do. With esoteric programming languages, this is not possible as there are no words to describe what exactly is happening.

While this process would yield me positive results, they were results that I would need to wait in order to start working with; the process of going through the IRB is quite extensive, and even after going through the process, there is also the need to advertise the exercise to students and wait for them to complete it. The students taking the exercise consist of students who have taken at least an introductory CIS course. Due to the fact that collecting real data would take time, as a medium I also generated synthetic data through the use of generative AI, specifically ChatGPT.

ChatGPT required users to ask questions as to what the AI should do; these questions are called queries. I started querying ChatGPT in a way that can generate responses that I could then feed into my model. Initially, my queries would look like this:

Pretend that you are a student with a novice level understanding of Brainfuck. As this novice-level student, answer this question (Part of answering the question as a novice includes the possibility of being incorrect): Using your understanding of BrainFuck, provide a program that assembles and prints the word "Hello."

I would end up getting this result:

This Brainfuck program initializes the memory with values corresponding to the ASCII codes of the characters in the word "Hello" and then uses the . command to output each character. The +, -, >, and < commands are used to manipulate the memory cells and build the desired ASCII codes.

This result was too literal and did not act as a university student. Trying to query this response to be more exhaustive/less exhaustive only led to similar results that weren't desirable. In order to effectively use ChatGPT, I had to actually structure my queries in a way that would allow for better and more realistic responses.

This led me to begin specifying what exactly I was looking for with my queries. I would ask ChatGPT to act as a university student with a focus in Computer Science in order to narrow down what kind of student response should be expected. I would also ask ChatGPT to either limit word choice, or try to expand word choice more, as well as limiting or expanding the word count for each response. Another significant query distinction I made was separating levels of responses into three categories; novice, intermediate, and advanced. Doing this meant that GPT had to follow the standard that I also included to generate a response that went along with exactly what I was asking it. The exact query I fed it looked like this:

As if you were a university student majoring in Computer Science, generate 25 responses that answer the question of how to generate the word "Hello" that are divided based on novice understanding, intermediate understanding, and advanced understanding of the programming language BFK. I will give you a reference sheet for the BFK programming language; refer to the reference sheet in your responses as understanding of the language should strictly come from the reference sheet. The response should be from 50 to 100 words. I will now define novice, intermediate, and advanced levels of understanding:

Novice level responses shouldn't be able to reflect what was learned from the reference sheet in great detail. Novice level responses shouldn't be too specific and should generally be vague in order to reflect a lack of understanding. Novice level responses should be less exhaustive than intermediate level responses. In order to include variation, novice level responses can also have a chance at being incorrect answers to the question

Intermediate level responses should be able to partially reflect what was learned from the reference sheet. Intermediate level responses should be fairly specific and fairly exhaustive. In comparison, they should be more specific and exhaustive than a novice level response, but shouldn't be more complex

than an advanced level response. They should have a lesser chance of being incorrect.

Advanced level responses should be able to perfectly reflect what was understood from the reference sheet. Advanced level responses should be very specific and exhaustive. In comparison, they should be more complex than both a novice level response and an intermediate level response. They should almost always be correct.

From this process of querying the AI in order to generate enhanced data, I was able to get 25 responses for each category. These responses were what I would base the usefulness of my topic model on and because I queried the AI in such a way, I could expect that once I gather more results from users, their results will not be too different from results gathered by the AI. Here is some responses that were generated and what they looked like:

Noice Level Responses I might try inputting various commands randomly into the BFK code to see if any combination produces the desired output. If I see any semblance of "Hello" appearing on the output, I'd consider that a successful attempt. Initially, I'd experiment with moving the pointer to different positions within the BFK array to observe any changes in the output. Then, I'd try different commands to manipulate the cell values and see if the output reflects the word "Hello." Experimenting with various combinations of commands while running the BFK program might yield unexpected results. By observing the output generated from random key presses, I could potentially stumble upon the desired output. Intermediate Level Responses Utilizing arrays to store the ASCII values of "Hello," I'd iterate through the array and adjust the cell values accordingly for each character. I would write a loop to iterate through each letter of "Hello," incrementing the cell values accordingly to match their ASCII equivalents. Employing techniques from object-oriented programming, I'd encapsulate the logic for generating "Hello" into a class, enabling better organization and abstraction of code. Advanced Level Responses Generating "Hello" in BFK involves a systematic approach to manipulate cell values corresponding to ASCII codes of each letter. Utilizing pointer movements, incrementing or decrementing cell values precisely to match ASCII representations, and strategically placing output commands ensure the accurate display of the word "Hello". To achieve "Hello" in BFK, one must meticulously manage pointer movements and cell values, aligning them with ASCII representations of each letter. Advanced techniques such as optimization of loops and conditional statements enhance program efficiency and accuracy in displaying the desired word. In the realm of BFK,

crafting “Hello” demands a sophisticated understanding of pointer manipulation and ASCII encoding. By employing efficient algorithms and leveraging the full spectrum of available commands, one can seamlessly orchestrate the generation of the word “Hello” with precision and elegance.

These responses gave me a good idea of how ChatGPT was addressing the question, and how ChatGPT is approaching the question as I did more querying. Even in the novice response ChatGPT was trying to answer the question in a way that highlights the steps taken to get the output; the output being the word “hello.” Essentially, it is redacting the information necessary to understanding the steps taken in solving the problem and acting as though it doesn’t have the knowledge to solve the problem. As the complexity of responses increased, the responses began including all the steps necessary in actually solving the problem.

Once I believed that the data would be able to reflect a student’s response, I began using the Latent Dirichlet Allocation (LDA) model. LDA is “generative probabilistic model that assumes documents are mixtures of topics, and topics are mixtures of words” (Mudadla). When I first discovered this model and realized that it utilized the word2vec infrastructure, I decided that I would try it out and see whether or not it could yield me some positive results. In discovering the LDA model I also discovered Latent Semantic Analysis (LSA). LSA is “a theory and method for extracting and representing the contextual-usage meaning of words by statistical computations applied to a large corpus of text” [latentsemanticanalysis&sentimentclassificationwithpython_2018]. The LDA model that I developed yielded results, but it wasn’t what I was looking for. Here is how I implemented the LDA model:

```
def create_dictionary_and_ldamodel(texts):
    id2word = corpora.Dictionary(texts)
    data = texts
    corpus = [id2word.doc2bow(text) for text in data]
    lda model  = gensim.models.ldamodel.Ldamodel(corpus = corpus,
                              id2word = id2word,
                              num_topics = 5,
                              randome_state = 100,
                              update_every = 1,
                              chunksize = 100,
                              passes = 10,
```

```
                            alpha = 'auto',
                            per_word_topics = True
    model = lda_model.print_topics(num_topics = 5, num_words = 5)
    return model
```

This model is a function that I created and works by first making a dictionary to reference which will be the dataset I input. It then uses tools and algorithms available in the gensim library to have the words in the dataset be parsed through. After that comes the creation of the model itself with the specification, here you can see that I specified the model to only return 5 topics, and for each topic include 10 words. This aspect of the model is something I can change, either by increasing the amount of topics that are available, increasing the amount of words to include in each topic and so on.

The LDA model, as explained in its definition, returns a specified amount of topics that the model understood to be different and central to understanding the data that was used. This means that LDA models would work better for data that includes a lot of different topics; for example a news article that talks about theft, inflation, politics, environmental activism would be a good data source to draw from because there are a lot of topics to pick up on. WHile testing this model I also found that it simply returns words in the dataset that seem to carry a large amount of weight in understanding the topic. This means that if the news article's section on theft were examined by the model it would probably return words like stealing, thief, robbery; words that really define the topic. This kind of model wouldn't be beneficial for the development of my project and so I tried the LSA model next. The LSA model is similar to the LDA model, but differs largely based on the inclusion of different topics. Here is the application of the LSA model:

```
def create_dictionary_and_lsamodel(texts):
    id2word = corpora.Dictionary(texts)
    data = texts
    corpus = [id2word.doc2bow(text) for text in data]
    lsa_model = LsiModel(corpus,
                num_topics = 5,
                id2word = id2word
    model = lsa_model.print_topics(num_topics = 5, num_words = 5)
    return model
```

Just like the LDA model, a lot of the set up is the same, the main difference is specifying what I want the model to find. In this case I had the mode look at 5 topics and 5 words per topic. The specifications can always be changed based on if I wanted to include more topics or more words.

As LDA models prosper when different topics are present in the dataset, LSA models prosper a bit more when a dataset is focusing on one or only a few topics. Testing this model did yield me better results than the LDA model, but still not enough to conclusively prove that it could be a viable way to provide feedback. Having explored these methods I ended up looking deeper into the possibility and found the constructed response calcifier.

Discovering the CRC led me to finding an application of NLP that has been explored before; implementing rubrics into programming. This was exactly what I needed in order to be able to return a value that would be significant and that could be trained to use accurately. The results of using this tool will be in the next section.

While this tool had its purpose and uses, I realized that the tool wasn't enough to ensure that the project was meeting the standard I meant to provide for it. In order to ensure that I was getting the most out of the model, I went back and started looking into word2vec modeling and other ways that I could possibly implement it to be effective. Going back and trying to find this ended up being the best result.

The LDA and LSA model had limitations, these limitations being that they weren't able to effectively describe the data simply based on the goal of each model. These models had value, but the way I was using them wouldn't have led me to any good results, so switching to word2vec modeling for semantic analysis was the best step going forward. In developing the word2vec model, I discovered that semantic analysis could be done in a way that was similar to the development of my prototype. My prototype showcased how word vectors were created, and it turns out that there was a way to manipulate word vectors in order to do calculations that could yield me the results I was looking for.

My prototype introduced me to word vectors, but I wasn't really sure where that was going to take me, but after doubling back and exploring I saw that word vectors can be manipulated as if they were simply numbers in an equation. I would be able to use them to make calculations, and I was able to calculate the average of word vectors, thus making a word vector that represented a sentence. The sentence could then be compared with other sentences that had been averaged and by utilizing cosine would be able to

yield the comparison between the two. The function for this ends up looking like this:

```
average_sentence_vector(sentence, model, num_features):
"""Find average of sentence vectors in model"""
words = gensim.utils.simple_preprocess(sentence)
feature_vec = np.zeros((num_features,)) dtype="float32")
nwords = 0


index2word_set = set(model.wv.index_to_key)
for word in words:
if word in index2word_set:
nwords += 1
feature_vec = np.add(feature_vec, model.wv[word])

if nwords > 0:
    feature_vec = np.divide(feature_vec, nwords)
return feature_vec

compare_sentence_to_reference(document, model,
reference_sentence, num_features=100):
sentences = sent_tokenize(document)
reference_vector = average_sentence_vector(reference_sentence,
model, num_features)

print(f"\nComparing sentences to: \"{reference_sentence}\"")
for i, sentence in enumerate(sentences):
    sentence_vector = average_sentence_vector(sentence,
    model, num_features)
    if len(sentence) > 50:
        similarity = 1 - cosine(reference_vector,
        sentence_vector)
        print(f"Similarity to sentence {i+1}: {similarity:.4f}")
    else:
        print("This response doesn't meet the rubric's word count.")
        pass
```

This function was able to yield me better results that I could work with.

This excerpt is only part of the function and showcases how the averaging of sentence vectors is done and how these vectors are then compared to a reference sentence. The reference sentence is meant to be a sentence that satisfies the requirements of the rubric. By comparing the responses to these requirements and seeing how similar the scores are between the two, I am able to define a sentence as being either similar, thus fulfilling the requirements on the rubric, or not very similar, thus trickling down the grade to something more appropriate. The results of the function will be in the next section.

While this discovery process was occurring, completing the English requirements came next, as I had to ensure that I had a literature review that argued my point, which was supported by evidence.

Evaluation

Starting this process was doing extensive research into whether or not this idea was possible. The previous sources led me to believe that this was the case, but next came proving that good technical writing was possible. Articles such as Alikaniotis's, Roy's and Ganguly's, and Kaldaras's, demonstrated that scoring utilizing NLP was possible and that more can be done in order to ensure that these scores are accurate. One of the reasons I go into depth in their articles and their explanations of automatic text scoring and assessment is because they help build the idea that an automated system such as this is possible.

While the development of my project, and the study they presented are different in terms of approach, the methods that they are utilizing in order to develop results and come to a conclusion are rubrics. They are reporting the results of running their systems through rubrics and getting results that are graded similarly. In order to prove that my tool is feasible and has the ability to do what I described it to be able to is by utilizing their examples and showcases of how to actually do it.

The combination of these two disciplines aren't presented in a way that is atypical of either subject, as CIS would be more computational and less about the process in discovering the tools and how I used them; English would also be more wordy with less use of code and data. The combination of the disciplines is the process in which I came to the conclusion that I will get to and explain, and realizing that was part of the process of this double thesis. Both disciplines work in tandem in order to prove the point that I am

making, and as I move to experimental results, I will explain more on how I used my program to prove my thesis.

Experiments

The results of experimentation caused me to change the development of my tool and how it would be used. Development and research for the tool led me down the path of exploring LDA and LSA modeling in order to satisfy the goal of my project. However, the issue with LDA and LSA lies in the results they yield and its effectiveness. LDA and LSA are forms of topic modeling and while they utilize word2vec they are utilizing it mainly to find topics and topic context from sentences. This seemed to be useful upon initial research, but as I worked with the models I realized that topic modeling would soon hit a dead end in how I could possibly utilize it.

LDA and LSA can display topics in a sentence, which does have value, but loses value for the sake of the project when the end goal of the two models is graphing. NLP is mathematical in nature, and the results from working with topic modeling will almost always be some number that needs to be defined in order to ascertain its value. For topic modeling we are given the weight of words in a document; the weight of these words define what the document is about, and it is up to us to determine what these topics even are. Graphing the results is how data scientists give meaning to something like the weight of these words, and while that is something I could possibly do, it wouldn't be as effective.

The project is looking for the possibility of a system that actually is able to determine whether someone's writing is meeting the criteria that is being displayed by the rubric. Going down the route that I was being taken on would yield me a mathematical result, but this result wouldn't be extremely accurate. The LDA and LSA models would find topics from the data I was feeding it, but these topics didn't accurately represent the data because of the fact that there was a lack of diversity in the kind of data it was working with. In addition to the lack of diversity, comes the fact that the results of finding the topic doesn't really offer enough results that I can work with.

Knowing the weight of the words in a topic doesn't have anything to do with individual responses; I can't use that information to grade someone's response. Here are the results that I would yield having experimented with LDA and LSA models:

```
LDA Model: [(0,
'0.006*"bfk" + 0.005*"hello" + 0.005*"might" + 0.004*"code" + 0.004*"output"'), (1
'0.002*"bfk" + 0.002*"hello" + 0.002*"output" + 0.002*"generating" + 0.002*"code"
'0.042*"bfk" + 0.041*"hello" + 0.026*"code" + 0.022*"output" + 0.016*"program"'),
'0.003*"bfk" + 0.003*"hello" + 0.002*"code" + 0.002*"output" + 0.002*"values"'), (
'0.003*"bfk" + 0.003*"hello" + 0.003*"code" + 0.002*"output" + 0.002*"might"')]

LSA Model: [(0,
'0.464*"bfk" + 0.457*"hello" + 0.292*"code" + 0.240*"output" + 0.180*"program"')]
```

In the LDA model, the topics are identified by the first number in each list starting from 0 (0, 1, 2, 3, 4). In each topic you can see that words start to repeat due to the lack of topics that are present in the document. The LDA model is meant to have more diverse information so this makes sense, however the LSA model shows that this doesn't really change. The LSA model is supposed to be able to handle less diverse information, so in order to encapsulate this I made the LSA model only display one topic. Understanding that the information being worked with is responses to solving the coding exercise that deal with the esoteric programming language BFK makes the words make more sense, but this doesn't really tell us anything about the individual responses. It is too all encompassing, and thus becomes ineffective.

I still include details about the LDA and LSA models because while they are ineffective in the goal of the project, they are not ineffective as a whole. These models do serve the purpose of returning topics that are seemingly prevalent and that is something that can be utilized. The returning of topics is affected by the weight of the words that the models extrapolate, meaning that the words make up the meaning. These models can be used to determine how to better utilize words/synonyms that can enhance the understanding of sentences. If the model focuses on words that aren't related to one another it could be helpful to then change those words to be more specific. This is a possibility and there is also the option of getting into R programming concepts such as linear regression and graphing.

Overall LDA and LSA modeling was not a good method for trying to find useful results; however, detailing the process of working with the model would be valuable for future reference. The LDA and LSA functions are included in the previous section; these are the essential functions of how the program works. Before using that function the user would just need to tokenize the file/data they wish to work with; this process is fairly simple. Tokenization looks like this:

```
def open_file_and_tokenize():
"""Opens file input by user and tokenizes the text."""
try:
with open("/content/sample_data/synthetic.txt", "r") as sample:
s = sample.read()
except FileNotFoundError:
print("File not found, please check again.")
sys.exit()
f = s.replace("\n", " ")
tokens = []
for i in sent_tokenize(f):
# Tokenize and remove punctuation
tokens.extend(gensim.utils.simple_preprocess(str(i),deacc=True))
return tokens
```

In order to work with this function, one must change a few things about the program. The program currently takes a file in my Google Drive called synthetic.txt and it is doing this because I specified the text file as so. The user would then have to change that file path within the parentheses into something that they wish to utilize. Other than that the user will have tokenized data produced indicated by the return statement

The next step for producing results from this model is by actually using the functions that create the LDA and LSA models. These models don't need to be manipulated to work, but can be changed depending on what the user of the program wants. The specifications all do different things such as specifying the number of documents to consider at once, specifying the amount of memory allocation going to the model, saving the state in which the model is currently in in order to test the same state over and over etc. I mainly used the model to try and change the number of topics that could be produced. I mainly did this in order to see if the increasing number of topics

could possibly lead to the inclusion of more words; this was not always the case for the LDA model at least. The LSA model, with an increased size of topics, would include more words that weren't previously being represented in the LDA model. The only issue with the words produced is that the words that seem to be encompassed within the topics don't always have such good correlation; the correlation tends to be better than the LDA model, but still nonsensical. As previously mentioned, these model types are mainly used for dealing with diverse amounts of data, so their purpose was heavily underserved in the scope of the project. Finally running the program is done by using the functions created in a main function as so:

```
def main():

    tokens = open_file_and_tokenize()
    processed_tokens = remove_stopwords(tokens)
    bigrams = make_bigrams(processed_tokens)

    lda_model = create_dictionary_and_ldamodel(bigrams)
    lsa_model = create_dictionary_and_lsamodel(bigrams

    print(f"LDA Model: {lda_model} \n\nLSA Model: {lsa_model}")

if __name__ == "__main__":
    main()
```

The main function goes through the process of tokenizing, processing tokenization, utilizing bigrams and feeding them through the LDA and LSA model. Seeing a few of the functions may raise questions on the purpose of them, and so I can explain their purpose. The remove_stopwords(tokens) function that is being assigned to the processed_tokens variables is simply a function that removes stop words from the data. The make_bigrams function being assigned to the bigrams variable takes the data and creates bigrams with the data. Daniel Jurafsky & James H. Martin's book Speech and Language Processing define bigram as so"a 2-gram (which we'll call bigram) is a two-word sequence of words like "please turn", "turn your", or "your homework", and a 3-gram (a trigram) is a three-word sequence of words like 'please turn your", or 'turn your homework' " (Jurafsky). The significance of using bigrams is its ability to increase predictability. By making two-word

programs they are able to use them to gain a better understanding of their possible meanings.

When I first discovered bigrams I believed that I would be using LDA and LSA models for prediction, but as development went on and I realized I wouldn't be I still included them. I believed that by including the models I would be able to more accurately determine which words would have greater weight in the output of the model. The bigram and stop words function look like this:

```
def remove_stopwords(texts):
    """Remove stopwords from the texts."""
    return [word for word in gensim.utils.simple_preprocess(str(texts))
    if word not in stop_words]

def make_bigrams(texts):
"""Creates bigrams from the provided texts."""
bigram = gensim.models.Phrases([texts], min_count = 5, threshold = 100)
bigram_mod = gensim.models.phrases.Phraser(bigram)
return bigram_mod[texts]
```

Having explored the LDA and LSA model and the concepts surrounding the two, I back tracked in order to find where I went off the rails with the idea and picked up on what would be more useful. The goal of my project and even the development of my prototype involved simply using the word2vec library that mainly focused on semantic analysis, so this is where I picked the project back up. My goal was semantic analysis, and so I started that process once again by building a model that mainly used only word2vec and finding how to use that model for testing.

I was initially able to create a program that compared each of the sentences in the synthetic data I gathered to each other in order to find semantic analysis. I did this by finding the average word vectors for each of the given sentences. This average word vector would serve as the determining factor for how similar two sentences are semantically when compared to one another. Because the data was synthetically generated I found that a lot of the sentences had similar structure to one another. The results of this experiment were as so:

```
Similarity between sentence 1 and sentence 31: 0.9833
```

```
Similarity between sentence 1 and sentence 32: 0.9874
Similarity between sentence 1 and sentence 33: 0.9802
Similarity between sentence 2 and sentence 18: 0.9808
Similarity between sentence 2 and sentence 19: 0.9909
Similarity between sentence 2 and sentence 20: 0.9873
Similarity between sentence 5 and sentence 9: 0.9872
Similarity between sentence 5 and sentence 10: 0.9860
Similarity between sentence 5 and sentence 11: 0.9813
```

The result shows that semantic similarity is very high with these sentences, as if the similarity is 1 then it shows that the sentences are identical. This is valuable and can be used when further development is made for comparing sentences. Currently the sentences are being compared with one another, so it makes sense that the similarity is so high; but what needs to happen is that sentences that satisfy the rubric perfectly need to be compared to the sentences that were generated. Doing this will yield better results as to what sentences correctly answer the question, and what sentences do not.

After getting this result, I realized that I would need to compare an example sentence that satisfies the rubric to the sentences I got from the data in order to get the results I am looking for, and so I created the example sentence that fulfilled all the requirements of the rubric: "I solved the coding exercise by learning how BFK works and how it stores values. Each value is stored in a cell, and you must utilize cells to both store values and do loopings. After discovering this I utilized the first two cells for constant looping and the other cells to store values. I understood the need to loop and increment values, but I didn't understand the ASCII table and getting numbers to match values." With this sentence satisfying the requirement I edited the program to compare the words against the response and these were the results:

```
Similarity to sentence 1: 0.9680
Similarity to sentence 2: 0.9741
Similarity to sentence 3: 0.9761
Similarity to sentence 4: 0.9795
Similarity to sentence 5: 0.9702
```

The trend is consistent with all the other sentences; the similarity between the synthetic sentences and the created sentences is very high. The high

similarity is due to the fact that the model is being trained off of the synthetic data. The responses in the synthetic data tend to have the same patterns of including a lot of the words and phrases that are present in the concept of BFK, so the response is matching the use of these words and phrases and thus getting a positive result.

At this stage the work is valid, but could be improved in order to account for more nuanced/unexpected results. The model itself is one that can be changed and trained to look for different patterns, and training the model however you want can yield positive results in the long run. The model works and the results are valuable, but the project hopes to go further beyond this point, so more can be done in order to improve the quality of grading that the model can do. In order to improve the quality of the model the first step would be to use better data in order to train the model.

While real data collected from users would be ideal, at the time of the project the coding exercise that I had provided users hasn't generated an extensive amount of data and so it is not a viable source to use. Because of the lack of real data, what would be the best possible substitute would be to query the generative AI in order to create specific outputs that differ a lot more, but are generated infrequently. In addition to specifying output, I was able to use a more exclusive version of the generative AI that would do more work with how responses would be generated, leading to more diverse and specific responses. Utilizing the same prompt as in the method section I generated, I was able to generate data that had more variation in complexity. Comparing these sentences to the example sentence I previously mentioned, I was able to get a more diverse set of example extrapolations:

```
Similarity to sentence 28: 0.1824
Similarity to sentence 29: 0.4458
Similarity to sentence 30: 0.3369
Similarity to sentence 31: 0.4067
Similarity to sentence 32: 0.5098
This response doesn't meet the rubric's word count.
Similarity to sentence 34: 0.3770
This response doesn't meet the rubric's word count.
Similarity to sentence 36: 0.3863
Similarity to sentence 37: 0.5052
```

This result shows a lot more variation, as certain responses either have a very high similarity score to the example response or a very low score;

there are also the cases in which some responses just aren't long enough to be considered. This variation replicates what real responses to the questions asked would look like which demonstrates validity. In order to reproduce the results of the experiment a lot of the features have already been included in the previous sections. Despite being able to replicate how human responses would look like, the inability to use real world data for the model doesn't allow the effectiveness of the model to be shown off. Synthetic data can only showcase so much without responses becoming repetitive, yet accurate. Evaluating the model is limited due to limitations and its effectiveness is not completely clear. This in itself seems to be a threat to its validity, but that isn't the case.

The validity of the experiment would be apparent had the results of the experiment, with the intended data, turned out to be invalid or un-useful, which is not the case here. The data that is being worked with does demonstrate a level of writing that could be apparent in a student, so to write it off completely wouldn't be fair. In the event that more students participated in the exercise and yielded more results the validity of the experiment could've been further examined, but this is not the case in this instance.

Threats to Validity

Despite the inability of not being able to use real world data, the tool itself is fairly effective for completing the task it is assigned. The tool finds similarity between sentences in a way that seems to be quite effective and while there could be more added to the tool its overall functionality can be quite useful. When thinking about how the tool can be evaluated it would be useful to consider other tools and how exactly they are developed. Looking at the development of other projects will help in evaluating my own, and to do that I took a look at Kaveh Taghipour and Hwee Tou Ng's article " A Neural Approach to Automated Essay Scoring." They focus on scoring essays, but the concept is the same. Here is how they evaluate essays: "The output of an Automated Essay Scoring (AES) system can be compared to the ratings assigned by human annotators using various measures of correlation or agreement" (Taghipour and Ng). As Taghipour continues he talks about different mathematical equations that essentially tries to find weight through referencing data that is input to data that is considered ideal. This method is similar to how I went about developing the program and lets me know that

the method in which I went about creating the system was something that was previously already thought about and done.

To reproduce the results of the experiment the user would need to utilize the code blocks I listed in the program in a code editor. I will put the parts together of the code blocks previously listed in order to allow users to simply take the entire code block instead of putting it together piece by piece.

```
import sys
import os
import proselint
from gensim.models import Word2Vec
from nltk.tokenize import word_tokenize, sent_tokenize
from nltk.corpus import stopwords
import gensim.utils
import numpy as np
from scipy.spatial.distance import cosine
from nltk import pos_tag, from nltk import pos_tag, RegexpParser
import nltk
nltk.download('punkt')
nltk.download('stopwords')


# Stop words
stop_words = stopwords.words('english')
stop_words.extend(['from', 'subject', 're', 'edu', 'use'])

def open_file_and_tokenize():
    """Opens file, lints content with proselint, then tokenizes words."""
    try:
        with open("", "r") as sample:  # Inputting path due to presence in Colab
            content = sample.read()
    except FileNotFoundError:
        print("File not found, please check again.")
        sys.exit()

    suggestions = proselint.tools.lint(content)
```

```
    # Tokenize
    words = word_tokenize(content)
    words = [w.lower() for w in words if w.isalpha()]
    # isalpha ensures that words are in lower case
    words = [w for w in words if w not in stop_words]
    return words

def train_model(data):
    """ Create and train word2vec model"""
    model = Word2Vec(sentences=[data],
    vector_size=100, window=5, min_count=1, workers=4)
    model.train([data], total_examples=1, epochs=10)
    # The model is trained based on the data that
    I am using so try to find a better source to train the data with
    return model # Trains model to be used

def average_sentence_vector(sentence, model, num_features):
    """ Find average of sentence vectors in model"""
    words = gensim.utils.simple_preprocess(sentence)
    # Goes beyond tokenization to ensure modeling works efficiently
    feature_vec = np.zeros((num_features,), dtype="float32")
    nwords = 0

    index2word_set = set(model.wv.index_to_key)
    for word in words:
        if word in index2word_set:
            nwords += 1
            feature_vec = np.add(feature_vec, model.wv[word])

    if nwords > 0:
        feature_vec = np.divide(feature_vec, nwords)
    return feature_vec

def compare_sentence_to_reference(document,
model, reference_sentence, num_features=100):
    sentences = sent_tokenize(document)
    reference_vector = average_sentence_vector(reference_sentence,
    model, num_features)
```

```
    # Comparison calculation
    print(f"\nComparing sentences to: \"{reference_sentence}\"")
    for i, sentence in enumerate(sentences):
        sentence_vector = average_sentence_vector(sentence, model, num_features)
        if len(sentence) > 50:
        similarity = 1 - cosine(reference_vector, sentence_vector)
        print(f"Similarity to sentence {i+1}: {similarity:.4f}")
        else:
        print("This response doesn't meet the rubric's word count.")
        pass

def main():
    processed_text = open_file_and_tokenize()
    word2vec_model = train_model(processed_text)

    reference_sentence = ""

    # Read the document again for sentence-level processing
    try:
        with open("insert_path", "r") as sample:
            document_content = sample.read()
    except FileNotFoundError:
        print("Document file not found for sentence comparison.
        Please check again.")
        sys.exit()

    # Perform and display sentence similarity comparisons
    print("\nCalculating sentence similarities:")
    compare_sentence_to_reference(document_content,
    word2vec_model,
    reference_sentence)

if __name__ == "__main__":
    main()
```

Where the program differs is the reference sentence and open_file_and_tokenize function. The reference sentence can be whatever sentence you want it

to be, you would just need to input that sentence into the program. The open_file_and_tokenize would also need a new path to specify in order to run the program. Running the program with a new reference sentence and new path will yield different results than the results currently reported. To go even further, in the train model function, the user can change the data that is used to train the model. The model is being trained by the data that is also being compared to the reference sentence, the user would just need to refer to the main function and change the argument for the train_model function assigned to the word2vec_model variable. Changing the argument will yield even more diverse results. These features can be changed and tested as much as the user likes and hopefully they are able to find different or more interesting results.

Conclusion

Having finalized the project there is a lot to discuss when thinking about the quality of the project, the output and its usefulness, future steps and planning, and the possibilities of polishing what has already been established. Personally, I feel that I got a lot done with what I was able to work with; however, I wish I had more time to try and polish what I have into something that is greater and more detailed.

Future Work

Starting with the quality of the project, I feel as though the project describes what exactly I wanted to prove in delving into the subject. The goal of my project was to make a program that would be able to grade essays, but also provide feedback, and while I wasn't able to implement the aspect of the project that implemented feedback, I knew that it was possible. The proseline function would've been one of the last things I'd have implemented had the time been available to me, and it would include the suggestions that I had been advertising.

The proselint function would've utilized the proselint computer program that does exactly what I detail. It is a linter, and what linters do is essentially look through either programs, or markdown files, and scans for specified checks. In terms of programming, these checks can have to do with formatting, efficiency, etc; the whole purpose of linting is to make programming better. The proselint would've been able to do just that for writing as well. When looking at the library itself, it is easy to see that it has great tools to use for checking and providing suggestions based on what exactly you may be looking for. The code would look something like this:

```
import proselint
```

```
suggestions = proselint.tools.lint("This sentence is very unique")
```

From this code you would get this output:

```
[('weasel_words.very', "Substitute 'damn'
every time you're inclined to write 'very;'
your editor will delete it and
the writing will be just as it should be."
, 0, 17, 17, 22, 5, 'warning', None),
('uncomparables.misc', "Comparison of an uncomparable:
 'very unique.' is not comparable.",
 0, 17, 17, 29, 12, 'warning', None)]
```

The output showcases the checks that the example program implemented, these checks being weasel_words.very, and uncomparables.misc (*Amperser/Proselint*). These are only two checks implemented for the sake of the example, but looking through the different checks that are available on proselint, I was able to find more that would be useful for the further development of my project:

consistency.spacing consistency.spelling dates_times.am_pm jargon.misc misc.but misc.capitalization misc.false_plurals misc.punctuation redundancy.misc spelling.able_atable spelling.able_ible spelling.em_im_en_in spelling.er_or

These are only a few checks that I am adding, as there are a lot more to be added and I would rather add them in the updated version of the program rather than here. From reading the titles of the checks, some are self explanatory, while others may need some explanation; overall these checks are able to do things like check for spelling, punctuation, capitalization, proper use of present and past tense, plurals, redundant words/sentences. Proselint has a lot of possibilities and being able to specify what exactly the user may want to use could've been a helpful feature.

While trying to implement it, I ran into an issue that was due to changes that underwent my project. I was initially working with my project locally on my computer, meaning that libraries and dependencies were being run from my operating system. I ran into an issue with my operating system in which some dependencies were not working properly. Because of this I had to switch from working locally on my computer to utilizing a Jupyter notebook which was available on Google Colab. The switch to Colab meant

I had to discover what I had to do differently, and this ended up causing my issue with proselint.

When proselint is installed on Colab, there seems to be a Key Error when trying to run it as a function. The program doesn't recognize the checks that are fed into the program's setup. This could've been solved had I been able to access the terminal and try using Proselint's terminal command, but the only way to use a terminal in Colab is by upgrading my privileges; this still wouldn't guarantee that I would be able to fix the program having accessed it from the terminal. Trying to access it from my computer's terminal was also out of the question because of the fact that the program was no longer able to even run due to the issues I initially discussed. Had the development of the program gone smoothly, I feel as though proselint could've been a great asset to increasing the ability of my project.

Thinking about how proselint works, it's also important to understand how exactly I was going to use proselint in my project. Proselint works with sentences, meaning that I would need to use proselint before I tokenized the information. It's important to remember that tokenization changes a sentence by taking out stopwords which include prepositions, and other words that help build sentence structure. I would only be able to use proselint before that happens, which also means that after using proselint, I would ask users to go back and try to edit the sentence that had suggestions offered. This need to go back and edit is helpful to the project, but could be considered annoying when thinking about how the project would then flow.

Users would need to input their responses and wait for the responses to go through the proselint process in order to get back the feedback to then edit their responses. Based on the amount of people who completed the exercise, completing the project was something that was undesirable (I will talk about the exercise later in this section). This means that I would also have to change the way the program flows in order to allow the proselint to happen. Including proselint would mean that user input would need to be submitted through proselint first, and returned back to the user so that they can implement the suggestions, and then submit the revised version which would then be submitted to be graded. This system is one I've seen before and is already similar to gatorgrade checks in the department. Overall, the inclusion of proselint could've really benefited the quality of the project and if able, I would implement the ideas discussed.

Another feature that I feel would've been useful would be the syntax tree. The syntax tree is an idea I discussed while talking about the Stanford-

CoreNLP. The StanfordCoreNLP details the structure of a sentence through the creation of a tree that shows the words that make up a sentence. While I wouldn't be utilizing the creation of the tree completely, I would be utilizing the out of the program that yields the tree.

The tree itself is a design that helps users see the creation of a sentence in a more detailed manner. When creating the tree in a Python file, the output is a little less graceful, but quite descriptive. The tree works by breaking down a sentence into these categories:

Sentence(S) Noun Phrase(NP) Determiner(Det) Verb Phrase(VP) Prepositional Phrase(PP) Verb(V) Noun(N)

These aspects encompass what makes up a sentence and how I'd use it is by essentially looking for what makes up a sentence; in this case what a typical response to the question I am asking entails and if users are able to emulate this kind of response. When looking at an example of how the program works, its usefulness becomes apparent. When running the sentence "The quick brown fox jumps over the lazy dog" through the program that yields a syntax tree, the following output is given:

```
(S
(NP The/DT quick/JJ brown/NN)
(NP fox/NN)
(VP jumps/VBZ) (PP (P over/IN) (NP the/DT lazy/JJ dog/NN))))
```

This may not make sense when looking at it at first, that's where the tree comes in:

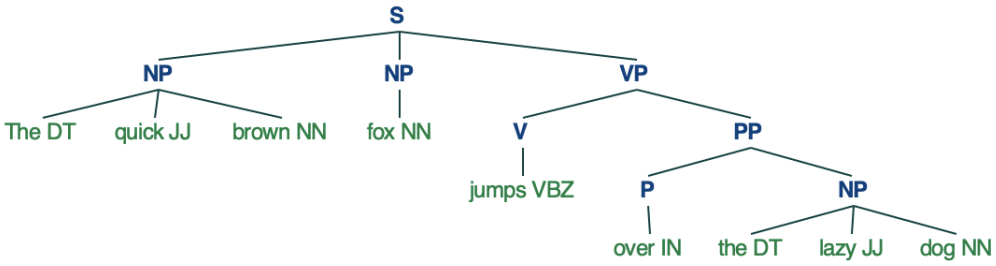

Figure 2: Synax-Tree

The program utilizes the breakdown of words into the aforementioned categories and also includes breakdowns that help with the development of

the program; this is where we see letters like DT, JJ, NN next to words in the sentence.

The usefulness of the syntax tree derives from its ability to identify parts of a sentence. It can be assumed that with the responses I work with, there should be multiple word types used in the responses; from a single response I should see nouns, verbs, prepositions, determiner, etc. This means that if a user could try to take advantage of the system by simply inputting words that are being sought out, they will have the issue of their response being determined as not being a sentence. The output of running their response through the syntax tree function would show if they simply repeated a word to try and get the check or, just input random words that have no real meaning.

Using the syntax tree would've been a convenient addition in order to ensure that the program isn't abused. I also personally wanted to add the syntax tree in as the drawn tree is quite interesting and paints a clear image of the capabilities of NLP.

Finally, the last feature I had hoped to commit was synonym comparison in order to increase vocabulary. This feature would've been implemented in a similar way as the project itself, as part of word2vec functionality is word comparison. I would've hoped to do this by taking vague words from responses and then comparing them with more specific words to show that when compared there wouldn't be a high comparison score, thus highlighting the need to utilize specificity.

This idea wasn't as fully-fleshed out however, as I wasn't sure how I would go about extracting words and what I would compare them to. Extraction of words would be the hardest part, being that I would want to get every aspect of the program automated, and the only way I can think of extracting specific words in order to make them more exhaustive would be by manually having to read and see which words could be improved. Changing those words to words that are more specific would also seem to be a challenge, as I don't think there is a system that could do that as well. The only way I could possibly think of changing this would be utilizing proselint once more which has programs that can check for usage of vague words, specifically "very," and weasel words.

The ideas that I've highlighted are just a few features I believe would have increased the quality of the work by a lot. Working on these features could've also shown me even more that could be added to the project in order to increase its functions and capabilities. The concept is one that should be

explored more because being able to write the high level ideas we work with is a skill that is practiced commonly in the tech industry. Without the skill to communicate a lot can't be done, as software is developed on teams. Being given more time I would have hoped to develop the project more to be able to communicate this importance.

Reflecting on the project as critically as I can, I'd say I wanted the project to be more developed than where I am concluding. The idea for the project is a good one, and explores a topic that isn't commonly talked about in the department. In exploring technical writing, my goal was to focus on the writing more than anything, meaning that the possible different ways of implementing the project didn't align with what exactly I wanted to do. Prediction and scoring is something that is apparent when working with data, but the goal of the project wasn't to work with data and just come up with a score that is the end all be all, but a method that shows the process of working with writing. Nonetheless, the importance of data presented itself, and so it would be important to explore how to attain data in a way that is better.

Learning about cognitive learning theory helped me better understand the reluctance to engage in work that makes someone think about themselves. Reflection goes beyond the task at hand, so asking users to engage in reflection turns out to be quite unfair to users. Engaging in an exercise that has no personal value to participants will never yield results that are accurate, and so the best way to engage in data gathering in the future would be to make this experiment a part of a course.

The experiment being conducted in an official college course would increase the likelihood of gathering data that is usable. This is due to the nature of courses; students participate in courses in order to receive a grade or fulfill a requirement, so because they need to engage and work with the material being taught they will be more willing to engage in reflection. This can extend past simply one course as well. While a course conducting this experiment is useful, more than one course doing the experiment would be the absolute best result. The reason for this is to increase the margin of success. Even if students are engaged in a course, they may still not want to participate in the experiment and engage in reflection, leading to tainted or unusable results being gathered. The more data we can gather overall will only benefit and lead to better results. Better data gathering will increase the accuracy of the project, but there still more to discuss about language and computation overall.

Ethical Implications

Language and computation are a crossroads that becomes very hard for ideas to merge because they work in completely different ways. Computation will always yield an answer that is accurate/accurate enough, while with language there is only so much detail you can pick up on and understand, and the understanding you have doesn't always apply to everyone. Being able to think about these two concepts and try to work in a way that accounts for them both requires a lot of thinking and handling of problems that arise, and I wanted to be able to handle more of the problems that arose in my work.

From problems with my software to having to change the code I used in developing my model, I wanted to be able to more proactively tackle these problems and solve them for the sake of producing something that would accurately reflect the intended goal of the project, and that is where I feel that I could've improved. The ideas that I have highlighted could've been improved had I been better about moving forward in my project and foreseeing the possible issues that could arise. If given the opportunity, I would continue development of the project in order to try and produce something that could be used for my intended purpose. Despite the shortcomings I believe were present, I am still satisfied with the result produced.

A lot of research for the project took me to other repositories in which other people tried to do the same idea, but in a different way. Looking at those ideas showed me that while writing isn't a computational subject (but has the ability to be) it can be turned into a computational subject depending on how it is done. Calculating the score of a response dismisses and diminishes the writing that was done and doesn't even try to acknowledge the content of what was written. This isn't a problem in the sense of crafting responses to fulfill a question, but does become a problem when the writing has value past simply answering a question. This is part of the reason that I didn't want to simply calculate a score for the sake of grading.

The writing I am focusing on, SRW, is writing that goes beyond simply answering a question, and that is because it makes the user think about what it is they learned and how they will relay that information. This kind of writing's effects branch past the scope of simply answering a question; it asks the user to think about themselves and their learning development. Telling users to answer questions related to their cognition and giving them a grade without much explanation, or feedback just didn't seem like the right thing to do in the scope of this project. This is what led me to discovering

ways to possibly produce some sort of suggestion that would show more attention to detail to these responses. Overall, I wanted SRW to be treated better in the department because it is the only form of technical writing that we do in the department. This project also led me to the discovery that students may not want to engage in SRW.

This discovery isn't new, when doing research I learned that there was an overall reluctance in engaging in SRW, which also makes me question if the project would be ethically right to ever be implemented. When thinking about the ethics of the project, I never thought that this project was crossing some ethical lines, and that is due to the presence of SRW in school already. Students don't like SRW, but it is something that is already ingrained in the learning curriculum, so to say that this project would be engaging in an ethical dilemma seems incorrect. Utilizing SRW is the best way to enhance technical writing without having students go out of their way to try and engage in a task that is specifically meant for technical writing; that is the point of this thesis.

The thesis isn't trying to create a whole new standard of teaching and engaging in good technical writing, it is trying to enhance a standard that already exists. If it does turn out that the standard of using SRW to improve technical writing is unethical, then I believe that SRW itself should be reviewed as to whether or not the entire concept itself is unethical. The results of the exercise may be negative, but it isn't necessarily directly related to the disdain for SRW. When thinking about the possible factors as to why students may not want to engage in the exercise, the possibility that the reason wasn't the subject matter, but other personal factors helps in ensuring the integrity of SRW.

The exercise that I had created in hopes of gaining real world data that I could then use to show the validity of the software tool didn't yield the results that I was hoping for. Models are best trained on large amounts of data, and so for I was hoping to gain a considerable amount of data from the exercise, but in the end I am only able to report one response from users meaning that exercise isn't viable to be used. While the exercise can't be used for the study it could be useful in discussing the desire to engage in SRW and how that desire is very low.

For the exercise, I advertised the incentive of being able to win a $20 giftcard. This incentive was meant to draw more people to the exercise and encourage them to participate in it, but in the end, even that wasn't enough to draw people in. From anecdotal evidence, a lot of people told me that the

reason they didn't want to engage in the exercise was because of the length. I estimated the exercise to be around 20-30 minutes, all depending on how long the person wants to spend time working on it. Telling people the length of the exercise was usually meant with some negative reactions. People didn't want to spend so much time on the exercise because of its length, and even when I explained the concept and how I was only looking for the response, I still was not able to gather enough people.

While it is unfortunate that not a lot of people participated in the exercise, it was to be expected through research. I had previously mentioned that with cognitive learning, there is a reluctance to engage in reflective learning; people don't like thinking about how they learn and other concepts related to their own cognition. Using this as a basis can explain why people may have not wanted to participate in the exercise, but there is also the possibility that what the exercise was asking was unreasonable.

The esoteric programming language I used was one that people more than likely not seen or engaged with, so having to learn a completely different language and be asked to answer questions may not seem very appealing. Whatever the reason may be, the fact of the matter is that the exercise wasn't appealing enough to draw in people to engage in it.

The reason the exercise had to include the esoteric programming language is so that everyone is able to answer the exercise, as it won't require people to already have knowledge of the language. This is why the requirements of the exercise was to at least have programming experience, because computational knowledge translates across languages. However, having the exercise be in language that students know could've possibly yielded more results, and is something I could try if I were to continue the project.

Despite only receiving one response from the exercise, one thing that could be done to utilize that data nonetheless would be to use it as a sample for generative AI in order to query it to create more responses that are similar in nature. The lack of data is an issue, and AI is currently the best way to replicate the data in a way that is at least usable. The validity of the AI can depend on the versioning and criteria fed into it, but the results are still able to be utilized for the sake of the development of the project. Without utilizing the generative AI the way I did, I would have no data to work with.

While the idea of using the one response to generate more AI responses is available, this option is unavailable due to the guidelines set up by the IRB. The purpose I stated for using the data did not involve feeding the data into generative AI, meaning that I can't use it for such a purpose. In addition, at

the conclusion of the study, the data that was collected must be destroyed in order to continue in following IRB guidelines.

Having worked on the thesis, engaged with ideas related to NLP and word2vec vectorization I believe that SRW can be improved and even automated in the realm of the department. NLP is a large and vast concept, and while the concept isn't something that people can always put their trust in, I believe that it can be used in a positive way. Word2vec is the perfect example of this, as other data scientists utilize word2vec and its vectorization in scoring prediction already. Further developing the project only would've led to more positive and accurate results as NLP is more developed than it has ever been in the past. The goal of this project was to prove that NLP can be used in the grading process and that point has been proven from the research and experiments I ran. I hope the development of the project benefits the CIS department and that need for practicing good technical writing has been established.

Alikniotis, Dimitrios. "Automatic Text Scoring Using Neural Networks." *arXiv: Computation and Language*, June 2016, http://arxiv.org/abs/1606.04289.

Amperser/Proselint. Mar. 2024, https://github.com/amperser/proselint.

Bus Factor. Mar. 2024, https://deviq.com/terms/bus-factor/.

Defining Technical Communication. Mar. 2024, https://www.stc.org/about-stc/defining-technical-communication/.

Ellis, Matt. *Learn the Rules for Every Sentence Type*. Mar. 2024, https://www.grammarly.com/blog/sentence-structure/.

Ho, Belinda. "Reactions of Students to Reflective Learning in a Technical Report-Writing Course." *English for Specific Purposes*, Mar. 1996, https://www.sciencedirect.com/science/article/pii/S0889490696000257.

Introduction to Word Vectors. Mar. 2024, https://medium.com/@jayeshbahire/introduction-to-word-vectors-ea1d4e4b84bf.

Jurafsky, Daniel. "N-Gram Language Models." *Speech and Language Processing*, Feb. 2024, https://web.stanford.edu/~jurafsky/slp3/3.pdf.

Kaldara, Leonora. "Rubric Development for AI-Enabled Scoring of Three-Dimensional Constructed-Response Assessment Aligned to NGSS Learning Progression." *Frontiers in Education*, Mar. 2022, https://www.

frontiersin.org/articles/10.3389/feduc.2022.983055.

Last, S., and BCcampus. *Technical Writing Essentials: Introduction to Professional Communications in the Technical Fields*. University of Victoria, 2019, https://books.google.com/books?id=iPAlygEACAAJ.

Mudadla, Sujatha. *Difference Between LDA and LSA*. Mar. 2024, https://medium.com/@sujathamudadla1213/difference-between-lda-and-lsa-f7fefa6b4bfd.

Petrosyan, Ani. *Internet and Social Media Users in the World 2024*. Mar. 2024, https://www.statista.com/statistics/617136/digital-population-worldwide/.

Roy, Dwaipayan. *Using Word Embeddings for Information Retrieval: How Collection and Term Normalization Choices Affect Performance*. Mar. 2018, https://dl.acm.org/doi/10.1145/3269206.3269277.

Tadayon Nabavi, Razieh, and Mohammad Bijandi. *Bandura's Social Learning Theory & Social Cognitive Learning Theory*. Jan. 2012.

Taghipour, Kaveh, and Hwee Tou Ng. "A Neural Approach to Automated Essay Scoring." *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, edited by Jian Su et al., Association for Computational Linguistics, 2016, pp. 1882–91, https://doi.org/10.18653/v1/D16-1193.

Thanaki, Jalaj. *Python Natural Language Processing*. 2017.

Vee, Annette. *Coding Literacy: How Computer Programming Is Changing Writing*. 2017.

Watson, Kathryn. *How Can i Help My Child with School Anxiety?* Mar. 2024, https://www.healthline.com/health/school-anxiety.

What Does a Software Technical Writer Do? Mar. 2024, https://technicalwriterhq.com/career/software-technical-writer/.

What Is Information Retrieval (IR) in Machine Learning? | Simplilearn. Mar. 2024, https://www.simplilearn.com/tutorials/machine-learning-tutorial/information-retrieval.