

1. Introduction

The purpose of this research project is to take a closer look at what it would take to develop one portion of an AWE system using the data at Paragon. Specifically, this work focuses on the vocabulary and morphology of test takers' written responses on the CELPIP test. Due to time constraints, the tools being created for this project are only a portion of a full system, and would require much more research and work in order to become a fully functioning AWE system. The analysis was conducted over all W02 and W01 test taker responses ever scored from the CELPIP test. This project poses the following questions:

1. What are the top 30 words in what we will consider high-level, mid-level, and low-level test takers?
2. What unique words are there in each of the 3 groups?
3. What clusters of morphemes appear more at higher levels than lower levels?

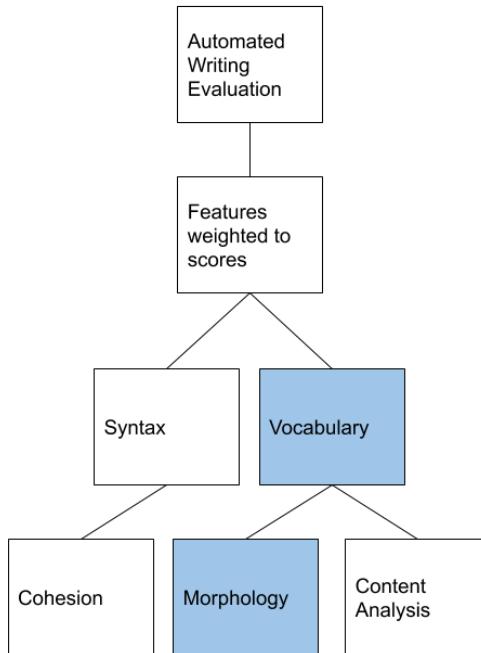


Figure 1. A tree of possible components in an Automated Writing Evaluation system.

This exploratory data analysis will provide some insights on what can be found within the area of morphology in relation to language level, and how to increase the validity and credibility of a future AWE system if one were to be built for Paragon.

2. Literature Review

The first seen commercial use of Automated Writing Evaluation (AWE) systems was to help score more essays and to provide instant feedback for students in the English Language Arts Curriculum (Foltz, Streeter, Lochbaum, Landauer, 2013). AWE systems are complex and are built by combining other, smaller -- but just as complex -- systems that look at specific features of the English language. similar to the rating scales used for many standardized assessment systems, including the Canadian English Language Proficiency Index Program (CELPIP) and Canadian Academic English Language (CAEL) tests. Looking at figure 1, we can see a partial breakdown of possible components in an AWE system. It is important to note that regardless of the system implementation that is picked, it will never be able to actually represent language. Creating systems to enable computers to analyze English essays is difficult because, in order for them to be trained, human-rated essays are required. This is also compounded by the fact that natural language processing needs to be created in order to break down the scored essays so that the features of each essay can be learned by a scoring algorithm (Dronen, Foltz, Habermehl, (2015). The purpose of the AWE will define the implementation of the natural language processing features to calculate proper and reliable scores for its intended use cases (Foltz et al, 2013).

Morphology as an indicator of language level has been neglected in favour of syntax and word frequencies when it comes to assessing language proficiency and development (De Clercq, Housen, 2019). English, although it is not as morphologically rich as other languages, has shown in research that as proficiency levels increase, understanding of more morphologically complex words increases (De Clercq et al, 2019). These findings have also been replicated with another set of English L2 Learners,

which was able to show the same positive correlation of an increase in understanding of morphologically complex words (Chen, 2011).

In order to process the morphology of words, there needs to be some logic that can do lemmatization or stemming. The two processes differ very slightly in that lemmatization is the process of finding the normalized stem of the word, while stemming is simply removing any affixes from the word (Plisson, Lavrac, Mladenic, 2004). Stemmers' and lemmatizers' implementations are also similar, but lemmatizers rely on dictionaries or labelled corpuses to ensure that a word is produced after affixes are removed (Koskenniemi, 1983). The corpus method requires a lot of manual labour in that the corpus needs to be tagged and labeled before being given to a machine learning algorithm.

There are some highly developed morphological parsers, such as Morfessor 1.0, which can learn morphological segmentation unsupervised using probability (Creutz, Lagus, 2005). Segmentation isn't too far from lemmatization, but rather the word is broken into its individual morphemes. These programs can be used instead of recreating the lemmatizing logic, but there are a few issues in using pre-built systems like Morfessor. One such problem is that there is little to no control over what data is stored when using the program. This is a problem as the Morfessor program does not count the number of morphemes or store the parts-of-speech for the words it segments, which could be used for further data analysis.

3. Methodology

3.1 Data

The data being analyzed comes from the Canadian English Language Proficiency Index Program (CELPIP). This test is used to evaluate test taker's English language skills for things such as immigration and citizenship. The specific area being looked at is the writing section of the test, which is comprised of two prompts. The first prompt, W01, elicits an email response from the test taker, with the contexts of

the emails varying very greatly. These emails can be about home arrangements with a landlord or job applications from an employer. The second prompt, W02, elicits an opinion-based answer based on a survey question. The contexts in how this is done also varies quite greatly in that they can be about the surrounding community wanted to build a park or what gift to buy for a co-workers birthday. The answers which test takers write for these prompts are the responses that will be analyzed for this study.

3.2 Tools

To analyze the data, natural language processing tools needed to be created to break down the words into their respective stem words. More specifically, a morphological parser was needed in order to extract the target data being analyzed. This morphological parser was built in Python version 3.7 to enable the use of the Natural Language Tool Kit (NLTK) (Bird, Steven, Loper, Klein, 2009), and is separated into 3 classes: WordBuilder, WordCounting, and WordStatistics. These 3 classes are supported by 3 other helping classes called DictionarySearch, FileHandling, and StringCleaning. All data is stored in 2 custom object classes called Word and TestTaker.

The helping classes perform functions very similar to their names. DictionarySearch helps the WordBuilder class by checking lemmatized words at dictionary.com to ensure they are proper words. FileHandling deals with all the reading of files to be used by the program and writing to files in order to display what's been done. StringCleaning takes words and strips them of any punctuation and white spaces so clean data can be worked with throughout the lemmatization process.

Custom objects were built to store the data being read in and created. The object Word would contain all the information pertaining to the surface form of the word such as the length, base word, the number of morphemes it contains, and its part of speech tag. The TestTaker object holds all information regarding the writing, and the person who wrote it.

WordBuilder is the first class that was created, and its purpose is to take in any form of writing in a .txt document to lemmatize and break down words to their stem and place them in a word bank. This class was built using the wordnet object and WordNetLemmatizer function from the NLTK (Bird et al, 2009). The algorithm in pseudocode for this class is as follows:

1. Read in any pre-existing word bank to add to.
2. Read in the text document to be parsed.
 - a. Make the entire text document lower case
3. Tokenize each word in the document, placing them in a list called wordlist.
4. For all words in wordlist
 - a. Clean the string of any punctuation and white space using StringCleaning.
 - b. If the word is still a word, keep the word.
5. If there is a pre-existing word bank to add to
 - a. Check to ensure all words in wordlist have not already been parsed.
6. For all words in wordlist
 - a. Send them through WordNetLemmatizer.
 - b. Create a word object for each and append them to a new list wordbank.
7. For all suffixes in the suffix text file
 - a. Check all words in wordlist for the suffixes
 - i. Keep the lemmatized word .
 - ii. If the words contain the suffix, remove the suffix and check if it's a word using DictionarySearch.
8. Sort wordbank alphabetically according to its pre-lemmatized state.
9. Store wordbank in a .pickle file.
10. Print the word bank out to a .txt file.

WordCounting is the class which takes test taker responses and counts them towards the word bank. WordCounting also tallies the response levels of each test taker that it parses and generates a graph of the level distribution. Test taker responses rated as "M" were randomly assigned a score from 0 to 2 to reflect a random distribution of test taker responses too low-level to receive a rating. The algorithm for this class is as follows:

1. We create 3 separate word banks for high-level, mid-level, and low-level test takers.
2. If test taker level is 9 or above, their words are counted in the high-level word bank.
3. If test taker level is in between 5 and 8, their words are counted in the mid-level word bank.

4. And if test takers scored 4 or less, their words are counted in the low-level word bank.
5. Save each word bank to a .pickle file for later use.
6. Ask the user if they want to remove word entries with a count less than 6 and if they want to remove words that have only the stem.
7. Print each word bank to a file.

WordStatistics displays the research questions stated above. It generates a list of the top 30 words in each word bank, the number of unique words in each word bank, and the clusters in each word bank. The function will display or create images and graphs to help analyze the findings. The algorithm used for this class is as follows:

1. Open pickled word banks from WordCounting.
2. Load in stop words for the stopwords .txt file.
3. Sort the word banks by their pre-stemmed representations.
4. Remove stop words from the high-level, mid-level, and low-level word banks.
5. Print the top 30 words in each bank to a .txt file.
6. Print the unique words in each level.

Only after the creation of these tools would it be possible to do the steps presented in the following section.

3.3 Data Analysis

After creation of the tools discussed in section 3.1 Tools, three books were given to the WordBuilder class to process and create a beginning word bank. Harry Potter and the Sorcerer's Stone, The Hunger Games, and The Hound of the Baskervilles were chosen arbitrarily and were readily available for public use. After the creation of this list, **** random test taker responses were checked against the word bank to see which words were counted and uncounted as a test run. This resulted in a list where there were more words that were uncounted than there were counted, so we added to the wordlist by sending these uncounted words to the WordBuilder class and then further testing again with another random **** responses. This again resulted in further uncounted words which were also added to the list, and then began to analyze the entire set of **** test taker responses. After all words from all responses had been counted, the word banks that were saved in the counting process were used to

display answers to the questions set forth in the introduction, which are: what unique words show up in which level ranges, what are the top 30 words in each skill level, and what are the morpheme word clusters in each level?

4. Analysis

The top 30 words in each test taker level don't show too much, but the low-level test takers seem to have a much different list than the high and mid-level test takers. This is expected due to the gap in lexicons of the different levels, but nothing more can really be obtained from looking at and comparing these lists. This does preface the rest of the analysis in that the features being looked for may only be between the mid and low-level test takers.

Looking at the unique words in each level grouping, there is only 1 word that is unique to the high-level list and 387 words that are unique to the mid-level with 0 unique words for the low-level list. Many of the unique words found in this mid-level list are spelling mistakes, and that is to be expected because the program has trouble filtering out misspelled words. After combing through the words, and removing some obvious names and misspellings, the unique list for mid level words was reduced to 301. There are a few implications that we can draw from these findings. Looking at the mid-level unique word list, there is a wide spread of words ranging from informal slang like suffix "-ish" or abbreviation "thurs" and stalling words such as "huh" or "yeh" to very low frequency words like "propitious" and "primordial". This might also be an indication that there is a limit, at least with the implemented method, in predicting test takers higher than a middle level. Due to the sheer number of test takers that place within the middle range, inevitably the chances of getting low frequency or misspelled words is also higher.

Looking at the entire list, the distribution was **** low-level, **** mid-level, and **** high-level test takers. Low-level test takers had **** distinct words, mid-level test takers had ****, and high-

level test takers had ****. There is a quick takeaway from the distinct word lists and that is we are seeing the correlation of English writing skills and the size of the test takers' lexicons at play in the current lists. In order to control for anomalous words that test takers might use in their writing sample, any words with a count of 4 or lower will not be considered when making observations. The reasoning behind this number is arbitrary, but the goal is to remove low count words from analysis. In figure 2, we have two word-clusters: "warn" and "warm". There are some obvious things we can see here, such as the low-level test takers not having as many variations on each word, with only 2 for warm and 1 for warn. A peculiar finding is that we see the mid-level test takers having more variations than high-level test takers, but this is within the realm of possibility because of the large number of responses at the mid-level.

Words	High Level Count	Mid Level Count	Low Level Count
warm	warm		
warmer	warmer		
warmers	warmer		
warmest	warm		
warming	warm		
warmly	warm		
warms	warms		
warmth	warmth		
warn	warn		
warned	warn		
warner	warner		
warning	warn		
warnings	warn		
warns	warns		

Figure 2. Side by side comparison of the word counts from high, mid and low-level respectively.

Looking at a word cluster that was a little more even across all skill levels, we can see obvious reductions in frequency when looking at a word with 2 morphemes across all levels. That is not to say that the word is more difficult. The surface form frequency compared to the sub word frequencies would need to also be analyzed with this data in order to draw any real conclusions. An interesting thing to note here is the very low frequency of the present tense or plural "-s" suffix. If the context and syntax

of the sentences that the words were used in were also factored in, it would be possible to get a better understanding of how and why these words were as high frequency or low frequency as they are.

Words		High	Mid	Low
surprise	surprise			
surprised	surprised			
surprises	surprise			
surprising	surprising			
surprisingly	surprising			

Figure 3. Word cluster where the counts are more significant all around

There are obvious shortcomings with the implementation that we currently have in our lemmatizing algorithm. For example, looking at our data, there are stemmed words like in figure 3 and figure 4, notice that there are some words that have not been properly lemmatized.

enjoy	enjoy
enjoyable	enjoy
enjoyed	enjoy
enjoying	enjoy
enjoyment	enjoyment
enjoys	enjoys

Figure 4. A word that was not fully lemmatized

families	fami
family	fami

Figure 5. Word group that was over lemmatized

The two suffixes “-ment” and “-s” in figure 3 are not something we lemmatize for, and there is no way for the program to learn to lemmatize for these words. In figure 4, we see another extreme case where the program is over lemmatizing, because “fami” is not actually a word in the English language. The over lemmatizing issue has to do with the fact we are searching dictionary.com for definitions and is not a fault of the lemmatizing algorithm. The dictionary search has a major drawback in that when building the word bank, because it relies on making website queries to dictionary.com, it is bottlenecked by the number of requests that can be made to the website. As the program goes through the

writing samples quite quickly, the requests may look like a Distributed Denial-of-Service (DDoS) attack. It is also bottle-necked in that it only lemmatizes for the suffixes that are in the designated text file. It is not possible for it to learn new affixes on its own, so another set of algorithms would be needed.

Looking at the uncounted word file, there are a lot of words that are spelling mistakes. In the list of uncounted words during the first smaller trial runs of **** and **** writing samples, they yielded **** to **** uncounted words per run, with the full run giving a ***-word list. A small sample is given in figure 5.

18089	libarey
18090	librarey
18091	librareyadrr
18092	libraru
18093	librert
18094	library
18095	mantly
18096	membar
18097	mesegas
18098	mintenens
18099	mintens
18100	myagsins
18101	pablik
18102	pepar
18103	pipall
18104	plees
18105	ples
18106	plis
18107	poblig
18108	pons
18109	responsebal
18110	rideg
18111	rilaks
18112	scool
18113	sespato
18114	stori
18115	sud

Figure 6 A portion of the uncounted word list showing the many spelling errors in the list

As can be seen in figure 5, many of the uncounted words are misspellings or random characters that the test takers would insert if they didn't have an answer to the prompt. For the purpose of our parser, this list would then be given to the WordBuilder class in order to further grow our saved word

banks and to update them with new words. This aspect of the system makes it a machine learning algorithm. In the broader context, these words would be counted towards the scoring process, but for our purposes, they have no real utility.

5. Moving Forward

Looking at the routes that this research can take, it might be more beneficial to take a more mathematical approach as used in the Morfessor 1.0 parser. The dictionary searching method employed for this study is too primitive a method for any high stakes rating situation, but for applications in reviewing practice tests or online practice material it might prove passable. We would need to be careful about possible repercussions with the validity of the scoring and possible washback on test takers that use the automated system.

Originally, the program was not going to rely on the NLTK wordnet lemmatizer to allow the program to also count the number of morphemes being removed from each word to see the complexity of the word. Unfortunately, due to time constraints, that feature needed to be abandoned, but should be developed in further research to allow the computer to make judgements on complexity of words in the response.

6. Conclusion

The project aimed to design and element of an automated scoring system, which, in conjunction with other tools, would support and enhance the rating process for written responses in a standardized assessment context. The design process was enlightening and highlighted a number of valuable insights. The test taker response data proved to be difficult to analyze because the program needed to be able to handle all forms of character strings. The algorithms used to parse through the words were simple so that also hindered the avenues of analysis that could be done on the test taker responses. The decision to use publicly available fictional books to build an initial preference corpus proved to be a much easier

process than working with the test taker data as the words were correctly spelled not artificially inflating the word list that was being built. In terms of using Python as the platform for building natural language processing and data processing, it was a very versatile tool and had many packages readily available to use.

This work lays a foundation for further analysis of morphological structures and their relationship to overall language proficiency. As predicted the relationship between specific morphological units and their use has only a limited relationship to overall language production. However, it is clear that the complexity and richness of the morphological tools used by the test takers in the target sample make this a worthy area of future study. Some limitations of this study include the use of simple, non-predictive lemmatizing algorithm, and the reliance of web API. Although the program is able to learn new words, it is unable to learn new affixes which is an important aspect to the progression and growth of this kind of system. Although it is a supervised learning algorithm, it is limited by the number of suffixes added to the list to check for. The reliance of querying dictionary.com also adds another bottle neck in that there are limits to using the API the website gives for programs to query it, and that caps word checking to 1000 per day. Possible future directions for this research would use the same response set, but would employ a different analytical method for lemmatizing or segmenting words. For instance, would creating a syntactic or lexical analytical measure alongside the morphological parser enhance the predictive power of the overall system?

Appendix

Top 30 high level words

would	would
time	time
also	also
work	work
like	like
could	could
people	people
new	new
one	one
may	may
believe	believe
company	company
children	child
know	know
regards	regard
think	think
day	day
employees	employee
school	school
get	get
us	u
many	many
option	option
help	help
community	community
make	make
thank	thank
need	need
office	office
able	able

Top 30 mid level words

would	would
time	time
like	like
also	also
work	work
people	people
one	one
know	know
company	company
good	good
think	think
new	new
could	could
get	get
need	need
option	option
help	help
day	day
make	make
us	u
school	school
children	child
first	first
regards	regard
really	real
want	want
best	best
thank	thank
better	well
may	may

Top 30 low level words

like	like
time	time
good	good
people	people
work	work
need	need
think	think
want	want
one	one
would	would
company	company
also	also
know	know
go	go
make	make
day	day
every	every
problem	problem
many	many
new	new
option	option
help	help
give	give
children	child
money	money
family	fami
school	school
thank	thank
hope	hope
food	food

References

- Bird, Steven, Loper, E., Klein, E. (2009). Natural Language Processing with Python. O'Reilly Media Inc.
- Chen, Shiyao. 2011. *How Does Morphological Awareness Have an Impact on Taiwanese Learners' Word Identification and Reading Comprehension?* Cheng Shiu University.
- Creutz Mathias and Lagus Krista. 2005. Unsupervised Morpheme Segmentation and Morphology Induction from Text Corpora Using Morfessor 1.0. Publications in Computer and Information Science, Report A81, Helsinki University of Technology, March. URL: <http://www.cis.hut.fi/projects/morpho/>
- De Clercq, B., & Housen, A. (2019). The development of morphological complexity: A cross-linguistic study of L2 French and English. *Second Language Research*, 35(1), 71-97.
- Dronen, N., Foltz, P. W., & Habermehl, K. (2015, March). Effective sampling for large-scale automated writing evaluation systems. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale* (pp. 3- 10).
- Foltz, P. W., Streeter, L. A., Lochbaum, K. E., & Landauer, T. K. (2013). Implementation and applications of the Intelligent Essay Assessor. *Handbook of automated essay evaluation*, 68-88.
- Koskenniemi, K. (1983). *Two-level morphology: A general computational model for word-form recognition and production* (Vol. 11). Helsinki, Finland: University of Helsinki, Department of General Linguistics.
- Plisson, J., Lavrac, N., & Mladenic, D. (2004). A rule based approach to word lemmatization. In *Proceedings of IS* (Vol. 3, pp. 83-86).