2012

University of North Carolina Wilmington

Master of Science in

Computer Science and Information Systems

Proceedings

https://csbapp.uncw.edu/mscsis

LANGUAGE ANALYSIS OF SPEAKERS WITH DEMENTIA OF THE ALZHEIMER'S TYPE

Anthony Habash

A Capstone Project (or Thesis) Submitted to the
University of North Carolina Wilmington in Partial Fulfillment
of the Requirements for the Degree of
Master of Science

Department of Computer Science
Department of Information Systems and Operations Management

University of North Carolina Wilmington

2012

Approved by

Advisory Committee

Dr. Laurie Patterson	Dr. Douglas Kline

Dr. Curry Guinn, Chair

Accepted By

Dean, Graduate School

Abstract

Language Analysis of Speakers with Dementia of the Alzheimer's Type. Habash, Anthony, 2012. Capstone Paper, University of North Carolina Wilmington.

This research attempts a discriminative analysis of conversation dialogues involving individuals suffering from Dementia of Alzheimer's Type. Several metric analyses are applied to the transcripts of the Carolina Conversation Corpus (Pope and Davis 2011) in order to determine if there are significant statistical differences between the averages of the individuals with and without Alzheimer's disease. Results from the analysis indicate that Pronoun Rate, GoAhead Utterances, Syllables-Per-Minute, and Honore's Statistic provide defensible means of differentiating the linguistic characteristics of spontaneous speech between healthy individuals and those with Alzheimer's disease.

Table of Contents

Table of Contents

Chapter 1: Introduction

Individuals suffering from dementia of Alzheimer's Type are characterized as being afflicted with a loss in cognitive and communicative functionality (Bucks 2000). This condition is often reflected within their powers of communication with 88 – 95% of patients with Alzheimer's (Thompson 1987) portraying some degree of aphasia (language disability) and cognitive failure including the inability to grasp concepts, events of their past, or the ability to recognize individuals.

Spontaneous speech, the focus of this research, is characterized as allowing for no pre-emptive planning or memorization of a response and demanding the highest level of cognitive action and memory contemplation to produce accurate and effective responses. This research attempts to detect quantitative signs of degradation in speech and cognitive capacities within individuals suffering from Alzheimer's Disease.

1.1 Speech Pathologies in Dementia of the Alzheimer's Type

A vast majority of patients with Alzheimer's are characterized by the degradation of their language and cognitive functionality, resulting in significant complications in vocal communication.

Dr. Barry Reisberg, a leading expert of the field who won the Lifetime Achievement Award for Outstanding Research by the International Conferences on Alzheimer's Disease (ICAD) and by the Alzheimer's Association, categorizes (Reisberg 1994) the levels of impairment caused by Alzheimer's disease into 7 different stages: Stage 1 – Normal, Stage 2 – Normal aged forgetfulness, Stage 3 – Mild cognitive impairment, Stage 4 – Mild

Alzheimer's disease, Stage 5 – Moderate Alzheimer's disease, Stage 6 – Moderately severe Alzheimer's disease, and Stage 7 – Severe Alzheimer's disease.

In early stages of the disease, Stages 1 through 3 (Reisberg 1994), individuals with Alzheimer's tend to have trouble staying on topic in a conversation, require more time to formulate responses, have difficulty finding correct words, and may lose their train of thought (Mace 2006, Ostuni 1986). In the middle stages, Stages 4 and 5 (Reisberg 1994), the inflicted have more difficulty understanding longer conversations, trouble explaining abstract concepts, difficulty finishing sentences, and may speak in vague and rambling ssentences (Mace 2006, Ostuni 1986). In the later stages, Stage 6 and 7 (Reisberg 1994), the individuals can lose the ability to understand most words, lose their ability to form proper grammar, and in the worst cases can become complete mute (Mace 2006, Ostuni 1986).

How to properly analyze and treat patients with Alzheimer's is an ongoing question. In the majority of cases of individuals diagnosed with Alzheimer's disease, the patient has likely suffered the condition for several years before the symptoms became evident enough for testing (Ray 2007), with most earliest cases diagnosed around Stages 3 or 4 (Reisberg 1994). This makes a comparison of previous cognitive and communication ability within the patient with Alzheimer's prior to the onset of the disease extremely difficult if not impossible. Therefore a broad, general comparison of the communication abilities of patients with Alzheimer's to those of individuals who do not suffer the condition might prove more accessible in determining signs of cognitive and linguistic degradation.

Alzheimer's effect on an individual's language capacities is strongly argued to reflect more within the patient's lexicon (their mental dictionaries and ability to understand complex words) than within their ability to formulate fluent and complete sentences (Singh 2000). It is

noted by Singh that the basis for this argument may be due to previous tests proving insensitive to a patient with Alzheimer's capability to compensate for speech degradation by word substitution, paraphrasing, sentence simplification, and non-word use (i.e. what-cha-call-it, thingy), or that the results might be dismissible due to the confrontational nature of the tests used to derive them (Singh 2000). Regardless, both areas of analysis are considered important for characterizing the form in which speech degradation occurs within an individual's communication abilities.

1.2 Computational Models of Speech in Patients with Alzheimer's

Other quantitative pathologies of speech explored within dialogue are the use of pauses, fillers, formulaic speech, restarts, repeats, incomplete statements and speech disfluencies (Davis 2009, Snover 2004, Bortfeld 2001, Yaruss 1998). The degradation of speech capacity exhibited within patients with Alzheimer's should be presented within these factors, with past research suggesting less lexical richness and higher levels of disfluencies - those sounds, distortions, and other non-lexical instances within dialogue (Yaruss 1998).

There is a complication in using these quantitative measures for contrasting individuals with and without dementia of the Alzheimer's type: namely, it can be hard to gauge an individual's pre-existing speech capacity. Also, pre-existing studies (Bucks 2000) which claim effective discrimination of patients with Alzheimer's and healthy individuals using these computational models have done so using a limited and size-controlled corpus for analysis, as well as using interview-style dialogue instead of spontaneous speech. The experiment should ameliorate some of these concerns as it uses a substantially larger corpus than previous research.

1.3 Research Hypotheses

We are specifically examining three hypotheses in this project:

1. **Syntactic hypothesis**: There are statistically significant variations in the syntactic structures of utterances spoken by individuals with and without dementia of the Alzheimer's type. This includes use of Nouns, Pronouns, Adjectives, and Verbs.

2. **Semantic hypothesis**: There are statistically significant variations in the semantic content (lexical richness) of utterances spoken by individuals with and without dementia of the Alzheimer's type. This will be measured by the use of Type-Token Ratio, Brunét's Index, and Honore Statistic Statistic.

3. **Pragmatic hypothesis**: There are statistically significant variations in the pragmatic content of utterances spoken by individuals with and without dementia of the Alzheimer's type. This includes pronoun usage, paraphrasing, and repeats, and syllables-per-minute.

Chapter 2: Literature Review and Analysis Methods

The field of Natural Language pertaining to the effects of dementia and Alzheimer's Disease on cognitive and communicative capacity is an extensive area of focus with several plausible scopes available for concentrative study. Notable scopes undertaken by researchers include the level of speech degradation in relevance to age, gender, relationships, topics, role, and stages of dementia (Bucks 2000, Bortfeld 2001). Regardless of these sub-factors, nearly all research done is built upon the analyses of two more general Natural Language aspects

when studying patients with Alzheimer's dialogue: lexical richness (Bortfeld 2001, Bucks 2000, Singh 1996, Singh 1997) and/or speech fluency (Yaruss 1998, Singh 2000, Davis 2009, Snover 2004).

The focus of the research conducted within this study is to determine if a discriminative analysis of the lexical and fluency properties of spontaneous conversations held by individuals suffering from Alzheimer's will show recognizable and reproducible patterns that reflect the degradation in cognitive and communicative abilities of the Alzheimer's patients. To achieve this end, several linguistic measures are applied, chosen due to previous research indicating their results do not correlate based on text-length (Singh 1996), which is an uncontrolled variable in these studies. Attempts at fluency measurements are also made using the design architectures of previous research (Boyd 2009, Snover 2004) to determine their effectiveness at detecting disfluencies and comparing the rates between Patient and Interviewer dialogues.

2.1 Corpus Annotation

2.1.1 Carolina Conversations Collection (CCC) Corpus

The corpus of spontaneous speech conversations involving patients with Alzheimer's that will be used in this study is the Carolina Conversations Collection. The CCC Corpus, developed in a partnership between the Medical University of South Carolina and the University of North Carolina Charlotte, was constructed in an effort to provide useful data concerning speech patterns with regards to age, gender, social identities, health and illness stories, and explanatory models of disease (Pope and Dave 2011). This makes the corpus

ideal for my needs as it was constructed primarily for such forms of analysis as those undertaken within this project.

The CCC corpus consists of over 400 transcribed conversations with 125 multiethnic, older individuals suffering from any number of possible conditions that have been individually categorized (Pope and Davis 2011). These conversations were originally recorded in only audio format and eventually transcribed into text using the methods defined by Ten Have (2007). However I used only 80 of these conversations, as only 80 transcripts involved patients suffering from Alzheimer's with 33 patients being involved in multiple interviews. To compensate for multiple transcripts involving the same person, all data gathered from transcripts involving the same Patient or Interviewer will be average.

"215.51 Ms. April Yes. Because you don't -- if you don't like one there's always another one and you didn't spend any money."

"239.35 Ms. March I will tell you one thing I have done as I've grown older. I no longer feel compelled to read the whole book. If it starts off and I'm not excited."

CCC Corpus Example 3.1.1-1

Example 3.1.1-1 is a sample of a short dialogue that portrays the form in which the conversations in the CCC are transcribed. For security and privacy reasons, all samples from the corpus have had speaker identities altered to sustain anonymity. The number at the beginning of each utterance represents the speaker's syllable-per-minute rate for the utterance.

CCC Corpus Profile	
# of Transcripts Used	80
# of Interviewers	59
# of Patients	33

Interviewers	Utterances	Word Count
Minimum	9	30
Maximum	362	4529
Average	92	689

Patients	Utterances	Word Count
Minimum	5	55
Maximum	196	2137
Average	81	805

2.1.2 Switchboard Corpus

The Switchboard Corpus is a collection of transcribed, spontaneous telephone conversations between healthy individuals (Godfrey 1992). This corpus was formulated as part of a project by Texas Instruments that was meant to address a growing need for large multispeaker databases of telephone bandwidth speech. Like with the CCC Corpus, conversations for the Switchboard Corpus were originally recorded via audio and then transcribed by hand into text using the methods defined by Ten Have (2007).

The Switchboard Corpus was included within this project in the early months of programming the implementation, before approval was obtained by the Institutional Review Board and the University of North Carolina Charlotte for access to the CCC Corpus, in order to provide a data set for testing of the programmed analysis methods. The format of the Switchboard Corpus was observed to be very close in comparison with the CCC Corpus and therefore a prudent means of early testing.

Switchboard Corpus Profile	
# of Transcripts Used	60
# of Speakers	181
Average Utterance Count per Speaker	130
Average Word Count per Speaker	1055

This corpus was also used informally as a baseline for testing against the analysis results of the CCC Corpus, specifically against the Interviewers, to see if the comparison would provide any possible value to this research project. The results of the comparison (Appendix E) showed little to no correlation that would validate using the Switchboard Corpus in this research beyond testing of analysis methods and so were not included in the implementation.

2.2 Syntactic Modeling

2.2.1 Part of Speech

Identifying the parts of speech within a speaker's dialogue allows for the break-down and categorization of the cognitive strength and capacity based on their lexical richness. This is achieved using the Natural Language Toolkit, a series of analytical programs designed to systematically parse and tag raw text (Bird 2009). The NLTK includes the WordNet lexical database within its toolset, which is then used for parsing raw text for determining Part-of-Speech tags based on word form and phrase syntax. This is further explained in Section 3.2 WordNet.

As the patient with Alzheimer's cognitive capacities fail, their ability to grasp facts such as names, places, and actions deteriorates, indicated by a rise in the use of pronouns, and a decrease in the use of proper nouns, verbs, and adjectives (Singh 1996).

For this analysis, the number of Nouns (N), Pronouns (P), Adjectives (A), and Verbs (V) are counted for each speaker within a corpus and then averaged against the Total Word Count (W) of that individual speaker to determine their Noun Rate (NR), Pronoun Rate (PR), Adjective Rate (AR), and Verb Rate (VR).

$$\mathbf{NR = N/W} \qquad \mathbf{AR = A/W}$$

$$\mathbf{PR = P/W} \qquad \mathbf{VR = V/W}$$

2.3 Semantic Modeling

2.3.1 Vocabulary Richness

Several studies have established that a form of communication degradation caused by the onset of Alzheimer's is the weakening of an individual's vocabulary (Bortfeld 2001, Bucks 2000, Singh 1997).

Three different forms of linguistic measurement are applied to the corpus to analyze the vocabulary richness employed by the speakers: Type Token Ratio (TTR), Brunét's Index (BI), and Honore's Statistic (HS) (Bucks 2000, Singh 1996). These weighted measurements provide an applied approach to measuring the lexical richness of a dialogue by providing algorithms for weighing the significance of unique vocabulary versus word count and total vocabulary.

Type-Token Ratio (TTR) provides a comparison to the total vocabulary used in a dialogue (V) to the total word count (N) of the dialogue.

$$\mathbf{TTR = V/N}$$

However Type-Token Ratio does not account for the variations in word count, which is an uncontrolled factor in spontaneous conversation. Brunét's Index (BI) is unique from

Type-Token Ration in that it attempts to quantify the vocabulary used without considering the word count. Brunét's Index will then likely show a more applicable result when applied.

$$\mathbf{BI = N^{V(-0.165)}}$$

Honore's Statistic (HS) attempts a deeper analysis by accounting for words that are only used once (V_1), indicating a higher lexical richness.

$$\mathbf{HS = (100 \log N) / (1 - V_1/V)}$$

2.4 Pragmatic Modeling

Distinct from a language's lexicon, disfluencies reflect a failure of concept more than a failure of vocabulary, where the speaker is uncertain, unclear, or doubtful of what they are trying to communicate.

Sara: "We went to that small place downtown… you know, they have those, uh… bamboo everywhere, with the shells, ya know…"

Jim: "The Koola Bar. Yeah, I've been there."

Speech Fluency Example 2.4-1

Spontaneous speech by its nature is commonly very disfluent (Bortfeld 2001). As shown in Example 2.4-1, Jim was able to understand what Sara was referring to, despite her use of disfluencies and incomplete sentences. Such forms of conversation occur completely naturally and reflect that speech degradation is not in itself a sign of dementia. These features, while maybe not inducing an impairment of comprehension, do however produce a complication in terms of parsing and analysis. Research done previously with the use of spontaneous speech corpuses attempts to account for this by rationalizing the different forms

of disfluencies with arguably related factors, including age, demography, race and gender (Bortfeld 2001), and applying that rationale to a large enough corpus to make the outliers evident.

The analysis of disfluencies within the corpus dialogue is further extended upon by analyzing certain identifiable properties within the speaker's utterances. These properties are theorized to further reflect a fault in the speaker's cognitive abilities and can be used to gauge the discrimination between patients with Alzheimer's and the non-impaired.

2.4.1 Fillers

Fillers are non-word and short phrase utterances that serve a communicative purpose with several possible meanings depending on their place (Bortfeld 2001). When placed at the beginning or end of a dialogue they could possibly be "hints" given by the speaker in order to indicate that they had trouble understanding something or that they desire input. They could also serve to indicate that the speaker has misspoken and desires to take back what was said or reword it, which is shown more the case when they occur in the middle of a dialogue than at the ends of it (Bortfeld 2001). In general though, they do indicate some form of cognitive lapse, where the speaker fails to communicate properly, hence we theorize that the rate of fillers would be higher for patients with Alzheimer's and weigh them accordingly.

2.4.2 GoAhead Utterances

GoAhead Utterances are instances in dialogue in which a speaker provides responses that do not add anything in a conversation beyond a minimal response. GoAhead Utterances usually serve as an indicator by one of the speakers that they either have nothing to input within a conversation or wish for another speaker to continue. GoAheads also serve as means

of validation that a person is listening to what someone is saying, or that they agree or disagree with what is being said.

207.82	Ms. April	It was a nonprofit organization.
0.00	Ms. March	Ummm.
351.11	Ms. April	And they didn't provide any of that.
0.00	Ms. March	Awww.
190.95	Ms. April	So, all my health coverage is still my husband who works for

CCC Corpus Example 2.4.2-1

Notice in Example 2.4.2-1 that Ms. March is not adding any information to the conversation, but rather is using filler phrases to respond to Ms. April's remarks. Ms. April continues the conversation, finding Ms. March's limited input as sufficient feedback. While used often enough in conversation among healthy individuals, it is arguable that patients with Alzheimer's will likely use GoAhead Utterances more often due to the lack of cognitive functionality necessary to produce such responses. For this study, GoAhead Utterances are classified by those phrases consisting of four words or less that also consist of fillers or any short-phrase instance as listed in Appendix A.

To provide a relative metric for comparison, the total number of GoAheads (GA) are counted for each speaker and then averaged by comparing to the total number of Utterances (U) for that speaker to produce their GoAhead Rate (GR).

$$\mathbf{GR = GA/U}$$

2.4.3 Repetitions

Repetition is a notable coping mechanism for cognitive failure where the speaker will repeat a stated word or words in order to allow the information provided within the discourse to become more evident within their cognitive process.

Repeats usually occur surrounding small pauses within an utterance where the speaker recaps or reaffirms to themselves what they are trying to communicate and then continue forth by picking up where they left off.

"That was a very… very intriguing speech we just heard."

Repeat Example 2.4.3-1

Repeats are a natural coping method for individuals who experience a form of cognitive lapse. As such it is arguable that individuals with Alzheimer's whose cognitive functionality has suffered degradation would be prone to use repeats more often than healthy individuals.

Repetitions were analyzed in terms of single-word and two-word repeats with both counted as having equal significance for each speaker. The total Repeat Counts (RC) were then divided by the total word count (W) of each speaker to give a relative metric showing the number of repeats per total words, or Repeat Rate (RR).

$$\mathbf{RR = RC/W}$$

2.4.4 Incomplete Words

Incomplete words are instants within the corpus where the speaker begins to pronounce a word but then inexplicably stops before completing the pronunciation, indicated within the CCC corpus by the use of a dash or tilde at the end of a word.

"Until my Mom got home and uh, as **gro~** as I was growing up, it was, I had a great mother."

Example 2.4.4-1

The cause of this behavior is creditable as a lapse in cognitive ability where the speaker intercepts their thought process before saying something wrong, or it can serve as an indicator that the speaker is unsure of what they are trying to communicate and stop altogether. Incomplete words were totaled (I) for each speaker in the corpus and divided against their word count (W) to give a relative metric of the number of incomplete words per the total number of words (IR).

$$IR = I / W$$

2.4.5 Syllables-Per-Minute

The rate of speech is provided within the CCC corpus using the metric of syllables-per-minute for both the Interviewers and the Patients with Alzheimer's disease. This form of measurement is indicated to be an effective means of comparing the speech capacity of individuals (Yaruss 1998).

155.22 Ms. Reid I don't have any grandchildren like that.
263.42 Ms. Jones Oh… you just kind of think of her as a grandchild?

Example 2.4.5-1

The rate of syllables-per-minute is provided within the CCC Corpus for each utterance. In the formatted text shown in Example 2.4.5-1, the value is placed at the beginning of each utterance for reference.

Relatively low values of this metric could possibly reflect cognitive impairment where individuals require more time to complete a vocal response due to having trouble finding the words they want to say, especially for more abstract concepts, or in wanting to ascertain that what they are saying is correct (Phinney 2002).

. Syllables-per-minute are totaled (S) for each speaker then divided by the total number of utterances (U) for that speaker to provide a relative metric of the average rate of syllables-per-minute (SR).

2.4.6 Paraphrasing – Direct, Indirect, and Reflexive

Paraphrasing is the act of taking what has been previously stated and reusing it in one's own context. People paraphrase all the time, be it something they read or something they heard prior to a situation in which they restate it in their own words. Paraphrasing is also a natural occurrence within conversation, where an idea or topic is passed back and forth between speakers with each adding their own knowledge in part to the matter and commenting on what the other has expressed.

James: "Did you like the football game yesterday?"
Mark: "No, I'm not really interested in football, or most sports."

Paraphrasing Example 2.4.6-1

In Example 2.4.6-1, James has introduced the topic of football. Mark has picked up on this topic and responds using the exact same word. This form of exact word usage is what we refer to as Direct Paraphrasing.

Next, by mentioning football, James has also introduced the subject of sports to the conversation, though he does not say that word directly. However Mark is aware that football

is a form of sports and furthers the conversation by addressing it. This is a form of Indirect Paraphrasing.

Finally, James designated the subject of conversation as Mark by using the pronoun "you". In turn Mark used the reflexive form of this by replying with the word "I". This is a form of Reflexive Paraphrasing where the two speakers pass comments concerning to topic of Mark using pronoun representation.

Each of these forms of paraphrasing are completely natural in healthy individuals, but in order for conversations to progress speakers have to apply their cognitive abilities to add new information to the conversation. This is shown in Example 2.4.6-1 as well when instead of James simply replying that he was not at the football game he added new knowledge to the conversation – his disinterest in sports. As this act of adding new information to a conversation becomes more difficult for those who find their cognitive functions weakening, it can be theorized that people suffering from Alzheimer's might show higher levels of paraphrasing in order to compensate for the lack of ability to expand or add input otherwise.

Chapter 3: Methodology

Implementation of this study was executed using Python-based programming. Python was chosen for its well-rounded toolsets for string and character manipulation and comparison methods. All corpuses were encoded in Unicode UTF-8 format to account for any possible non-ASCII characters.

3.1 Patients and Interviewers

When implementing each of the analyses methods described within Chapter 2 upon the transcripts of the CCC Corpus, the results are by Patients – those with Alzheimer's disease – and by Interviewers – those without Alzheimer's disease. Once the results of each metric have been grouped by speaker-type, they are then measured for three specific comparative metrics: Mean, Standard Deviation, and T-Testing the averages of each metric between the results gathered for Interviewers and Speakers to determine if there is significant statistical difference between the two groups.

The Mean and Standard Deviation are self-explanatory. With the T-Test, we calculated the P-Value of each comparison. We chose a p-value < 0.05 to indicate strong statistical significance. We chose a p-value < 0.1 to indicate weak statistical significance. We also took note of p-values close to 0.1 that may indicate statistical significance, but more data would be needed to verify or invalidate the null hypothesis.

3.2 WordNet

A major contribution to the work within this project was through the use of WordNet (Princeton 2010). WordNet is a large lexical database with implementation toolsets for use in programming and data manipulation. WordNet provided the means for Part-Of-Speech tagging as well as word comparison for paraphrasing measurements.

3.2.1 Part-Of-Speech Tagging

WordNet uses the Penn Treebank Project's Tag Set (Appendix B) for analyzing text for its part-of-speech structure. The Penn Treebank Project is a language analysis project headed by the University of Pennsylvania for the purpose of analyzing language to determine its

syntax. WordNet implements a syntax base parsing method, using predefined standard phrase forms, to determine the POS of each word in a line a text.

Pre-Parse: "I'm going to draw you just a little sketch map of where I go everyday just on a typical day"

Post-Parse: "I'/**PRP** m/**VBP** going/**VBG** to/**TO** draw/**VBG** you/**PRP** just/**RB** a/**DT** little/**RB** sketch/**NN** map/**NNP** of/**IN** where/**WRB** I/**PRP** go/**VBG** everyday/**NNP** just/**RB** on/**IN** a/**DT** typical/**JJ** day/**NN**"

WordNet Parsing Example 3.2.1-1

There are instances of inaccurate tagging that occur when applying the WordNet parser to the text, most plausibly due to the inclusion of disfluencies and improper syntax, but these instances have been found to be rare in relation to the size of the transcripts, so it is assumed that they do not provide major variance within the results.

Although part-of-speech tagging was revealed to be included with the CCC Corpus, we maintained using the WordNet method due to time constraints and consistency.

3.2.2 Paraphrasing

As stated in Section 2.4.6, three forms of Paraphrasing are being analyzed within this project: Direct, Indirect, and Reflexive. While it is arguable that each form might have different significance, for this experiment they are each weighed equally and together.

The Paraphrasing method begins with utterance selection where each utterance within a transcribed conversation is looked at individually and compared with the next valid utterance both before (Backwards Paraphrasing) and after it (Forward Paraphrasing). In this experiment, any utterance that is not defined as a GoAhead utterance is considered valid.

Paraphrasing is scored for each utterance in a range of 0 to 1, where a score approaching 0 indicates the utterance has no relation to the other, and a score approaching 1 indicating that almost every word within the utterance is used in the other it is being compared with. Once a score is determined for every word in an utterance, an average is produced by taking the sum of word paraphrase scores and dividing it by the total number of words within the utterance. Afterwards that score is summed for each utterance of a speaker and then divided by the total number of utterances spoken by that individual, providing a relative scoring method for how much paraphrasing is done by an individual.

Direct and Reflexive Paraphrasing are both relatively simply to parse within a body of text. Direct Paraphrasing (DP) is found by doing a one-to-one word matching between two utterances within a dialogue. If two words are found to be the same, a value of 1 is added to the total paraphrasing score of the target utterance.

Reflexive Paraphrasing (RP) required the use of mapping out the reflexive forms of all pronouns. It is notable that even though a reflexive form a pronoun in one utterance may be found in another utterance that it does not mean that both pronouns are referring to the same subject. However as all utterances being compared are in close proximity with each other in terms of placement within the conversation, such instances were considered negligible in this experiment.

It is with the detection of Indirect Paraphrasing (IP) that WordNet becomes invaluable in providing means of detection.

Mary: “Do you like to **dance**?”
Steve: “Sure, I like **dancing**.”

Word-Form Altering Example 3.2.2-1

In the example, Steve replies to Mary's question by answering with the word "dancing" instead of "dance" which have the same base concept but are not the same word. In these cases, when compared to each other, WordNet does a comparison of each word's synonym sets, or synsets, which are a collection of linked abstractions of a word covering several possible meanings. Once a path of synonymous meaning is found between two words' synsets, the path similarity is determined between the two words and returns as a number between 0 and 1 with 0 indicating that the words have no relation and 1 indicating that they are practically the same word. In the case of Example 4.1.2-1, the words "dance" and "dancing" would return a score of 1 as they are different forms of the same concept.

John: "There's a **pastry** shop down the road."
Sally: "A **croissant** sounds nice."

Indirect Paraphrasing Example 3.2.2-1

In Example 4.1.2-1, John uses the word "pastry" when referring to a shop. Sally in turn uses the word "croissant", a type of pastry, indicating that she was using the information from John's statement in forming her response. In this case, WordNet would compare the synsets of the words "pastry" and "croissant" and determine they share a relationship, returning a value between 0 and 1 representing the path similarity between the two words' closest identical synonym. In this case, WordNet would return a value of 0.72, indicating that the words are extremely close in concept and a form of Indirect Paraphrasing.

Once all the scores are tallied for each word within an utterance, the values are summed (S) then divided by the total word count (W) for the utterance, providing a score that represents the utterance's level of paraphrasing (P).

$$P = \frac{(\sum DP + RP + IP)}{W}$$

With this methodology, all three forms of paraphrasing are detectable, providing a means of analyzing for differences in the amount of paraphrasing conducted by individuals with or without Alzheimer's Disease.

3.2 Implementation

Initial work began in September 2011, with extensive research conducted to review past experimentations and findings. After a consensus was reached concerning which areas of study to pursue, I began programming in November and started the process for obtaining the CCC corpus for implementation of analysis.

Unforeseeable setbacks prevented access to the CCC Corpus until March 2012. In the time before then, testing was accomplished through the use of the Switchboard Corpus (Godfrey 1992). With this data we were able to test program functionality and also obtained data for comparison with that retrieved from the CCC Corpus.

Corpuses were broken down based on speakers, identifying dialogue as that spoken by the Alzheimer's Patients and all others as Interviewers. All results for the Switchboard Corpus were categorized as a single group.

The measurements for lexical richness and fluency were applied to each corpus after the dialogue was diagnosed, using Mean, Standard Deviation, and T-Testing as basis for comparing results.

Chapter 4: Analysis Results

After each method of analysis was applied to the CCC Corpus, the results were compiled and organized for review (Table 4).

	Mean (Standard Deviation)		
Metric	**Interviewers**	**Patients**	**T-Test**
Noun Rate	0.40677(0.054397)	0.396943(0.049574)	0.197794
Pronoun Rate	0.162067(0.029697)	0.169256(0.024976)	◊0.116581
Adjective Rate	0.01565(0.013256)	0.01982(0.018976)	0.143498
Verb Rate	0.079944(0.017234)	0.082821(0.014029)	0.200505
Type-Token Ratio	0.414266(0.140182)	0.406233(0.114436)	0.3865
Brunét's Index	13.239077(1.895905)	13.211988(1.823095)	0.474043
Honore's Statistic	615.108485(93.748051)	623.211816(93.918416)	0.350794
GoAhead Utterances	0.411846(0.163757)	0.348812(0.153503)	*0.038422
1-Word Repeat Rate	0.01126(0.011325)	0.014054(0.009207)	◊0.107715
2-Word Repeat Rate	0.005526(0.007035)	0.003877(0.003401)	ǂ0.069793
Syllables-Per-Minute	191.736991(58.757117)	169.732184(58.328372)	*0.049221
Forward Paraphrasing	0.318599(0.057077)	0.286192(0.0666)	*0.012904
Backwards Paraphrasing	0.313971(0.059407)	0.30639(0.056964)	0.279504
Incomplete Word Ratio	0.004169(0.006505)	0.010424(0.027456)	◊0.11383

Table 4 : Comparison of metrics between Interviewers and Patients (Repeated in Appendix C)

* Indicates a T-Test p-value score of less than 0.05, which suggests a significant statistical difference in the averages of the Interviewers and Patients.
ǂIndicates a T-Test p-value score of less than 0.1 which suggests a weakly significant statistical difference in the averages of the Interviewers and Patients.
◊Indicates a T-Test p-value score close to 0.1 which suggests a metric may be significant but more data is needed to confirm or refute this hypothesis.

4.1 Syntactic Analysis Measurements

4.1.1 Review of Part-of-Speech Results

Previous research indicates that individuals with Alzheimer's usually show changes in their speech patterns by using less nouns, verbs, and adjectives and more pronouns (Bortfeld 2001). This would reflect a decrease in an individual's cognitive capacities due to the degradation caused by Alzheimer's disease.

The results of the Part-of-Speech analysis upon the CCC Corpus indicate no significant decrease within the Noun Rates of any of the individuals with Alzheimer's compared to the Interviewers. The other Part-of-Speech analysis show results along those same lines with no significant difference shown within the CCC Corpus concerning Adjective Rate and Verb Rate either between the individuals with or without Alzheimer's.

The results for the Pronoun Rate do indicate an increase in the use of pronouns by the Patients, which supports the claim by previous research that suggests individuals with Alzheimer's are likely to use more pronouns in the daily speech to compensate for inability to remember proper names and titles (Singh 1997).

4.2 Semantic Analysis Measurements

4.2.1. Review of Vocabulary Richness Results

Type-Token Ratio, Brunét's Index, and Honore's Statistic have each been argued by previous research to provide sufficient means of measuring Lexical Richness to differentiate between individuals with Alzheimer's and otherwise healthy individuals (Bucks 2000)

The results for Type-Token Ratio show a slight difference in Vocabulary Richness with Interviewers scoring the highest. The scores of the T-Test indicate that these results are not likely supportive of differentiating between the two groups. The results for Brunét's Index actually show a slightly higher level of Vocabulary Richness among the Patients than the Interviewers, as indicated by the lower Mean score achieved by the Patients. But the results from the T-Test do not support the ability to differentiate between them. Similar results were found for Honore's Statistic, where the T-Test indicated no significant statistical difference between Interviewers and Patients. In conclusion, the results are not supportive of the theory

that these methods for measuring lexical richness are an effective means for differentiating between individuals with Alzheimer's and those without it.

4.3 Pragmatic Analysis Measurements

4.3.1 Review of GoAhead Ratio Results

The results of the GoAhead analysis indicated that healthy individuals use a considerable amount of filler phrases in their conversation, accounting for nearly 40% of all utterances by the Interviewers. Interestingly the Patients with Alzheimer's disease only used GoAhead utterances about 35% of the time, with the T-Test results indicating that there significant difference in the averages between Patients versus Interviewers. This could be interpreted as suggesting that individuals with Alzheimer's disease naturally try to communicate more in their speech than others.

4.3.2 Review of One-Word and Two-Word Repeat Results

Previous research theorizes that individuals with Alzheimer's are more likely to have instances of repeating themselves in conversation than healthy individuals (Bortfeld 2001). The results of this analysis support this claim.

Patients with Alzheimer's are shown to have a higher number of Single-Word Repeats than the Interviewers (25% more), with the T-Test result very weakly supporting a claim of significant difference between the two (T-Test p value = 0.1077). Conversely, interviewers were shown to produce more Two-Word Repeats than either the Patients, and the T-Test indicates significant difference between the Interviewer results with those of the Patients.

4.3.3 Review of Incomplete Word Ratio Results

The results of the Incomplete Word Ratio analysis indicate a difference with the number of incomplete words spoken by Patients with Alzheimer's to those of the Interviewers. The Patients displayed a substantially higher amount on average than the Interviewers (+145%); however the T-Test score only very weakly indicates a statistical difference between the two groups (T-test p value = 0.113). This supports the theory that individuals with cognitive impairment will more often implore the use of incomplete words (Bortfeld 2001) as a means of indicating that a spoken phrase needs to be altered before continuing.

4.3.4 Review of Syllables-Per-Minute Results

It is shown from the data collected from the CCC Corpus that the Interviewers portrayed a higher rate of syllables-per-minute than the Patients (13% faster). The T-Test also indicates that the averages for the Patients and Interviewers are substantially different from each other. This is probably the strongest supporter discovered in this project that it is possible to differentiate between individuals suffering from dementia and healthy individuals. This also suggests that focus on measurements of speech rate might be prudent for further research.

4.3.5 Review of Backwards and Forward Paraphrasing Results

While no significant difference was shown in the levels of Backwards Paraphrasing between Interviewers and Patients, the results showed significant difference in the amount of Forward Paraphrasing. The Interviewers are shown to have a higher rate. When considering the relatively equal amount of Backwards Paraphrasing conducted by both parties, this possibly indicates that Interviewers take more lead within a conversation than the Patients in

terms of providing topic. It could also be a result of simply the Interviewers simply working to spark conversation more than the Patients.

Chapter 5: Summary and Future Work

The results from the Semantic Analysis have shown very little in suggesting that such means can be used to differentiate between Patients with Alzheimer's and non-inflicted individuals. It should be noted that there is a substantial difference in the lexical richness using Honore's Statistic between the Switchboard subjects and the subjects in the CCC (see Appendix D). Both interviewers and persons with Alzheimer's disease displayed statistically significant less lexical richness. A hypothesis for future study would be to determine whether conversational partners in an environment where one partner has Alzheimer's disease lower their lexical richness to match the (presumed) lower lexical richness of the person with Alzheimer's disease.

While previous research suggested that there would be significant difference in the Noun, Adjective, and Verb Rates of individuals with Alzheimer's disease when compared to that of healthy individuals (Singh 1997), almost all Part-of-Speech rates showed direct correlation between Interviewers and Patients with Alzheimer's, with the T-Tests indicating no significant difference between the averages obtained from the corpus. Only the results for the Pronoun Rate showed results possibly supporting the ability to discern between healthy and afflicted individuals.

The results from the Pragmatic Analysis were more supportive of the theory that differentiation between Patients with Alzheimer's and un-inflicted individuals through study of dialogue was possible. The paraphrasing scores signified a higher reliance of Patients with Alzheimer's to use another's dialogue to generate responses. The most significant piece of data was the syllables-per-minute results which indicated that Patients with Alzheimer's are generally slower in their dialect. However the results in analyzing Repeats were not

conclusive. The use of GoAhead phrases was significantly different with Interviewers using more on average than the Patients. The results from the syllables-per-minute analysis also suggest that speech rate may prove an effective method to distinguish individuals who might be suffering from Alzheimer's disease.

There are many variables unaccounted for within the project that may have affected the results of the analyses. The number of transcripts was significantly less than was previously expected, limiting the ability to draw solid conclusions from the results. No discrimination between individuals with Alzheimer's was applied within this project. Factors that may have considerable consequences into the effectiveness of this attempt at analysis are age, ethnicity, level of education, location of long-term residence, and cognitive capacity prior to the onset of Alzheimer's disease, and what stage of the disease the Patients were suffering from when they conducted the conversations in the CCC corpus. Given that the results of the Patients and the Interviewers correlated closely with each other in several of the metric analyses, it suggests that the Patients were likely in the early stages of the disease, possibly Stage 3 or 4 (Reisberg 1994), and have yet to suffer severe cognitive degeneration.

While the stage of the disease and prior cognitive capacity are not available within the CCC Corpus, age, residence, education level, and ethnicity are supplied for use. A more discriminative analysis of the CCC corpus that takes these variables into consideration may provide significantly different results than what were found with this general analysis. However the limited size of the CCC corpus's number of transcripts with individuals suffering from Alzheimer's disease might make a discriminative attempt at analysis unreliable.

In conclusion, several of the metrics analyzed using the CCC Corpus's collection of spontaneous speech involving Alzheimer's Patients show results suggesting they effectiveness in differentiating between individuals with and without Alzheimer's disease. This ability to differentiate will be significant in determining means of analyzing the effectiveness of conversation tactics with Alzheimer's patients like those suggested by Pope and Davis (2011) and possibly help in developing new tactics to improve conversation.

Capstone Project Schedule

Task Name	Completed by
Download, install and integrate Natural Language Toolkit (NLTK) with Python	September 30, 2011
Coding for Part-of-speech analysis	October 15, 2011
Coding for fluency analysis	November 01, 2011
Preliminary Literature Review	November 01, 2011
UNCW IRB Approval	November 01, 2011
Proposal Defense	December 14, 2011
MUSC IRB Approval	January 04, 2012
Download CCC corpus	January 10, 2012
Syntactic Analysis of CCC	January 30, 2012
Semantic Analysis of CCC	February 30, 2012
Pragmatic Analysis of CCC	March 15, 2012
Written Proposal to Committee	April 2, 2012
Capstone Defense	April 27, 2012
Submit capstone to graduate school	April 27, 2012

Appendix A – GoAhead Utterance Instances

The following is a list of words associated with those conversation instances that are specified as being GoAhead Utterances.

YES
YEAH
NO
AND
THEN
REALLY
GOOD
UHN
NOPE
RIGHT
SURE
GREAT
HUNH
HUH
SO
COURSE
MAYBE
MAY
WELL
HM
HMM
YOU
KNOW
HEY
MMM
MHM
BECAUSE
WHY
WOW
COOL
OH
OHH
OOH
OOOH
DEAR
WHAT
BUT
YOU
DID
NOT
YWN
ARE
UNHUNH
UHHUH
UNHUH
PLEASE
WHAT
UM-HMM
UM-HM
YEP
MMHMM"
UHM
I MEAN
AH
MM
ER
UH
AW
NAH

Appendix B – Penn TreeBank II Part-Of-Speech Tags

- CC - Coordinating conjunction
- CD - Cardinal number
- DT - Determiner
- EX - Existential there
- FW - Foreign word
- IN - Preposition or subordinating conjunction
- JJ - Adjective
- JJR - Adjective, comparative
- JJS - Adjective, superlative
- LS - List item marker
- MD - Modal
- NN - Noun, singular or mass
- NNS - Noun, plural
- NNP - Proper noun, singular
- NNPS - Proper noun, plural
- PDT - Predeterminer
- POS - Possessive ending
- PRP - Personal pronoun
- PRP$ - Possessive pronoun
- RB - Adverb
- RBR - Adverb, comparative
- RBS - Adverb, superlative
- RP - Particle
- SYM - Symbol
- TO - to
- UH - Interjection
- VB - Verb, base form
- VBD - Verb, past tense
- VBG - Verb, gerund or present participle
- VBN - Verb, past participle
- VBP - Verb, non-3rd person singular present
- VBZ - Verb, 3rd person singular present
- WDT - Wh-determiner
- WP - Wh-pronoun
- WP$ - Possessive wh-pronoun
- WRB - Wh-adverb

Appendix C – CCC Corpus Results

The results included within this graph are derived by analyzing and comparing the values obtained from the CCC Corpus after grouping the speakers as either Interviewers or Patients with Alzheimer's Disease.

	Mean (Standard Deviation)		
Metric	**Interviewers**	**Patients**	**T-Test**
Noun Rate	0.40677(0.054397)	0.396943(0.049574)	0.197794
Pronoun Rate	0.162067(0.029697)	0.169256(0.024976)	◊0.116581
Adjective Rate	0.01565(0.013256)	0.01982(0.018976)	0.143498
Verb Rate	0.079944(0.017234)	0.082821(0.014029)	0.200505
Type-Token Ratio	0.414266(0.140182)	0.406233(0.114436)	0.3865
Brunét's Index	13.239077(1.895905)	13.211988(1.823095)	0.474043
Honore's Statistic	615.108485(93.748051)	623.211816(93.918416)	0.350794
GoAhead Utterances	0.411846(0.163757)	0.348812(0.153503)	*0.038422
1-Word Repeat Rate	0.01126(0.011325)	0.014054(0.009207)	◊0.107715
2-Word Repeat Rate	0.005526(0.007035)	0.003877(0.003401)	‡0.069793
Syllables-Per-Minute	191.736991(58.757117)	169.732184(58.328372)	*0.049221
Forward Paraphrasing	0.318599(0.057077)	0.286192(0.0666)	*0.012904
Backwards Paraphrasing	0.313971(0.059407)	0.30639(0.056964)	0.279504
Incomplete Word Ratio	0.004169(0.006505)	0.010424(0.027456)	◊0.11383

* Indicates a T-Test P-Value score of less than 0.05, which suggests a significant statistical difference in the averages of the Interviewers and Patients.
‡Indicates a T-Test P-Value score of less than 0.1 which suggests a weakly significant statistical difference in the averages of the Interviewers and Patients.
◊Indicates a T-Test P-Value score close to 0.1 which suggests a statistic may be significant but more data is needed to confirm this hypothesis.

Appendix D – Results for CCC Corpus Interviewers and Switchboard Speakers

These results included within the graph are derived from the analysis applied to the CCC Corpus' Interviewers and the individuals of the Switchboard Corpus. Both groups are assumed to be composed of individuals with good mental health, allowing for a side-by-side comparison of the results in order to gauge any differences that might exist.

	Mean (Standard Deviation)		
Metric	**Interviewers**	**Switchboard**	**T-Test**
Noun Rate	0.40677(0.054397)	0.382054(0.039419)	*0.000919
Pronoun Rate	0.162067(0.029697)	0.151914(0.031576)	*0.013543
Adjective Rate	0.01565(0.013256)	0.015523(0.00751)	0.472271
Verb Rate	0.079944(0.017234)	0.079641(0.016387)	0.452924
Type-Token Ratio	0.414266(0.140182)	0.397954(0.13222)	0.216914
Brunét's Index	13.239077(1.895905)	13.356139(1.648635)	0.336168
Honore's Statistic	615.108485(93.748051)	652.255122(99.592523)	*0.005348
GoAhead Utterances	0.411846(0.163757)	0.411786(0.160197)	0.49903
1-Word Repeat Rate	0.01126(0.011325)	0.016077(0.013926)	*0.004317
2-Word Repeat Rate	0.005526(0.007035)	0.00423(0.004444)	0.093664
Forward Paraphrasing	0.318599(0.057077)	0.276928(0.079684)	*0.000012
Backwards Paraphrasing	0.313971(0.059407)	0.277711(0.079838)	*0.000016
Incomplete Word Ratio	0.004169(0.006505)	0.028264(0.046926)	*0

* Indicates a T-Test P-Value score of less than 0.05, which suggest a significant statistical difference in the averages of the Interviewers and Patients.

The results gathered from the comparison of the speakers from the Switchboard corpus when compared to those of the Interviewers were perceived as being too different from each other to indicate any sort of correlation within speech behavior.

Appendix E – Coding For Analysis Program

```
    import os

    ################
    #generate own corpus from html file (FAIL)
    ################
    #folder = fg.open_folder()
    #from urllib import urlopen
    #url = "C:\Users\A.R. Habash\Documents\Visual Studio 2010\Projects\NLP Dementia
Analysis\NLP Dementia Analysis\Corpus\TESTHTML.html"
    #html = urlopen(url).read()
    #raw = nltk.clean_html(html)
    #tokens = nltk.word_tokenize(raw)
    #print tokens

    ######################
    #Using Single text file, load corpus into a Text type variable (FAIL)
    ########################
    #from nltk.corpus import PlaintextCorpusReader as ptcr
    #corpusFilePath = fg.open_folder()
    #print corpusFilePath
    #wordLists = ptcr(corpusFilePath,'TestSample1.txt')
    #print wordLists.fileids()
    #print wordLists.sents()

    ########################
    #Load a text file, remove all tags to get the raw text (Keep the speaker
identifiers to be removed on next step of process
    #Save as 'FILENAME Raw.txt in "Raw Text" Directory'
    ##############################
    CorpusList = []
    corpusPath = 'C:\Users\A.R. Habash\Documents\Visual Studio 2010\Projects\NLP
Dementia Analysis\NLP Dementia Analysis\Corpus\\'
    rawPath = 'C:\Users\A.R. Habash\Documents\Visual Studio 2010\Projects\NLP Dementia
Analysis\NLP Dementia Analysis\Raw Text\\'
    fileList = os.listdir(corpusPath)

    for item in fileList:
        newCorpus = ccc.Corpus(corpusPath + item)
        del newCorpus
        #CorpusList.append(newCorpus)



    ##Anthony Habash
    ##Dementia Analysis Project
    ##Dr. Curry Guinn
    import os
    import TextParseTools as tpt
    import DialogueClass as dial
    import TextAnalysisTools as tat
    ##Each instance of this class will be for a different corpus (document)
    ## Each corpus is divided into two instances of DialogueClass - one belonging to
the interviewer, and
    ##  one belonging to the DATSpeaker (Dementia of Alzheimer's Type)
    ## the Dialogue Class instance will do the analysis based on the entire dialogue
of speaker (i.e. everything they say in the Corpus)
```

```
    ## both Dialogue Class instances will also maintain a list of Monologue class
instances (monologues consist of everything the
    ## speaker says in one instance, no matter how many sentences or pauses in
between, before the other speaker talks again)
    ## the Monologue Class does all the same calculations as the Dialogue Class, but
only on the individual part rather
    ## than on everything the speaker says in the Corpus as a whole)
    ## EXAMPLE: - CORPUS:
    #                       SMB: Hello. How are you?
    #                       IWF: I'm fine, thank you. And you?
    #                       SMB: Doing good, thanks.
    #               DIALOGUE:
    #                       SMB: Hello. How are you? Doing good, thanks.
    #               MONOLOGUE:
    #                       SMB: Hello. How are you?
    #               MONOLOGUE:
    #                       SMB: Doing good, thanks.

    class Corpus:

        #self.rawCorpus = the raw Corpus (STRING)
        #self.rawCorpusList = the raw CORPUS as a LIST divided by MONOLOGUES
        #self.interviewer = the TAG (STRING) of the INTERVIEWER
        #self.datSpeaker = the TAG (STRING) of the DATSPEAKER
        #self.InterviewerDialogue = an instance of the DIALOGUE CLASS for the
INTERVIEWER which will maintain all the analysis variables
                                    #for the complete Interviewer's Dialogue
(everything the interviewer says in the corpus)
        #self.DATSpeakerDialogue = an instance of the DIALOGUE CLASS for the
DATSPEAKER which will maintain all the analysis variables
                                    #for the complete datSpeaker's Dialogue
(everything the datSpeaker says in the corpus)


        def __init__(self, filePath):
            self.fileName = os.path.basename(filePath)
            #Indicates which Corpus is being used for the programs. 0 = CCC, 1 =
SwitchBoard
            self.corpusType = 0

            self.rawCorpus = tpt.getTextFromFile(filePath)

            rawSpeakerListAndIndividuals  = tpt.divideBySpeaker(self.rawCorpus,
self.corpusType) #For CCC Corpus
            self.rawCorpusList = rawSpeakerListAndIndividuals[0]
            self.speakerList = rawSpeakerListAndIndividuals[1]
            if u":" in self.speakerList:
                self.speakerList.remove(u":")

            self.DialogueDictionary = {}
            for speaker in self.speakerList:
                self.DialogueDictionary[speaker] =
dial.Dialogue(tpt.rawSpeakerDialogueList(self.rawCorpusList,speaker),speaker,
self.corpusType, self.fileName)
            #self.cleanCorpusList = []
            #for monologue in self.rawCorpusList:
            #    self.cleanCorpusList.append(tpt.cleanText(monologue))
            #print self.rawCorpusList
```

```
            #print self.cleanCorpusList
            #tat.paraphrasingComparison(self.cleanCorpusList, self.speakerList,
self.corpusType)

    import TextParseTools as tpt
    import MonologueClass as mc
    import TextAnalysisTools as tat

    class Dialogue:

        #self.rawDialogue = a single STRING containing everything said by the speaker
in the entire Corpus w/tokens
        #self.cleanDialogue = a single String containing everything said by the
speaker in the entire Corpus without tokens
        #self.monologueList = a list of MonologueClass Instances dividing the
speaker's monologues into individual instances
        #self.wordCount = an INTEGER of how many words are in the dialogue

       def __init__(self, dialogueList, speaker, corpusType, fileName):
            self.rawDialogue = compileDialogue(dialogueList, corpusType)
            self.speaker = speaker
            #self.cleanDialogue = tpt.cleanText(self.rawDialogue)
            #self.cleanDialogue = tpt.cleanUnicode(self.cleanDialogue)
            #self.monologueList = processMonologues(dialogueList, self.speaker)
            #self.wordCount = tat.wordCount(self.cleanDialogue)
            #self.vocabularyCount = tat.vocabCount(self.cleanDialogue)
            #self.uniqueVocabularyCount = tat.uniqueVocabCount(self.cleanDialogue)
            #self.typeTokenRatio = tat.typeTokenRatio(self.vocabularyCount,
self.wordCount)
            #self.brunetsIndex = tat.brunetsIndex(self.vocabularyCount, self.wordCount)
            #self.honoresStatistic = tat.honoresStatistic(self.wordCount,
self.vocabularyCount, self.uniqueVocabularyCount)
            #self.POSDictionary = tat.POSRates(self.cleanDialogue)
            self.incompleteWordCount = tat.hiccupCount(self.rawDialogue)

            #if corpusType == 0:
            #    self.syllableRates = tat.syllableRate(dialogueList)
            #
            #if self.wordCount == 0:
            #     self.nounRate = 0
            #     self.pronounRate = 0
            #     self.adjectiveRate = 0
            #     self.verbRate = 0
            #     pauseCalcs = 0
            #     self.pauseFrequency = 0
            #     self.pauseAverageLength = 0
            #     self.repeatCount = 0
            #else:
            #    self.nounRate =
float(tat.nounRate(self.POSDictionary))/float(self.wordCount)
            #    self.pronounRate =
float(tat.pronounRate(self.POSDictionary))/float(self.wordCount)
            #    self.adjectiveRate =
float(tat.adjectiveRate(self.POSDictionary))/float(self.wordCount)
            #    self.verbRate =
float(tat.verbRate(self.POSDictionary))/float(self.wordCount)
            #    pauseCalcs = tat.pauseCalculations(self.rawDialogue)
            #    self.pauseFrequency = pauseCalcs[0]
```

```
            #    self.pauseAverageLength = pauseCalcs[1]
            #    self.repeatCount = tat.repeatedWordCount(self.cleanDialogue)
            #printValues(self)
            printForReview(self, corpusType, fileName)


    def printValues(self):
            print
'_____________________________________________________________________________________
_____________________________'
            print 'DIALOGUE VALUES FOR - ' + str(self.speaker)
            print '           WordCount: ' + str(self.wordCount)
            print '          VocabCount: ' + str(self.vocabularyCount)
            print '    UniqueVocabCount: ' + str(self.uniqueVocabularyCount)
            print '      TypeTokenRatio: ' + str(self.typeTokenRatio)
            print '      Brunet\'sIndex: ' + str(self.brunetsIndex)
            print '  Honore\'sStatistic: ' + str(self.honoresStatistic)
            print '            NounRate: ' + str(self.nounRate)
            print '         PronounRate: ' + str(self.pronounRate)
            print '       AdjectiveRate: ' + str(self.adjectiveRate)
            print '            VerbRate: ' + str(self.verbRate)
            print '         PauseCount: ' + str(self.pauseFrequency)
            print ' AveragePauseLength: ' + str(self.pauseAverageLength)
            print '       Repeat Count: ' + str(self.repeatCount)

    def printForReview(self, corpusType, fileName):
        if corpusType == 0:
            print fileName + ' ' + str(self.speaker) + ' ' +
str(self.incompleteWordCount) # + ' ' + str(self.wordCount) + ' ' +
str(self.vocabularyCount) + ' ' + str(self.uniqueVocabularyCount) + ' ' +
str(self.typeTokenRatio) + ' ' + str(self.brunetsIndex) + ' ' +
str(self.honoresStatistic) + ' ' + str(self.nounRate) + ' ' + str(self.pronounRate) +
' ' + str(self.adjectiveRate) + ' ' + str(self.verbRate) + ' ' +
str(self.pauseFrequency) + ' ' + str(self.pauseAverageLength) + ' ' +
str(self.repeatCount) + ' ' + str(tat.averageSyllableRate(self.syllableRates))
        else:
            print fileName + ' ' + str(self.speaker) + ' ' +
str(self.incompleteWordCount) #+ ' ' + str(self.wordCount) + ' ' +
str(self.vocabularyCount) + ' ' + str(self.uniqueVocabularyCount) + ' ' +
str(self.typeTokenRatio) + ' ' + str(self.brunetsIndex) + ' ' +
str(self.honoresStatistic) + ' ' + str(self.nounRate) + ' ' + str(self.pronounRate) +
' ' + str(self.adjectiveRate) + ' ' + str(self.verbRate) + ' ' +
str(self.pauseFrequency) + ' ' + str(self.pauseAverageLength) + ' ' +
str(self.repeatCount)

    def compileDialogue(dialogueList, corpusType):
        compiledString = ''
        if corpusType == 1:
           for item in dialogueList:
               compiledString = compiledString + item
        else:
           for item in dialogueList:
                itemList = item.split(None)
                subString = ''
                if len(itemList) > 2:
                    for index in range(2,len(itemList)):
                       subString = subString + u" " + itemList[index]
                compiledString = compiledString + subString
```

```
        return compiledString

    def processMonologues(dialogueList, speaker):
            monologueList = []
            for item in dialogueList:
                monologueList.append(mc.Monologue(item, speaker))

            return monologueList

    import TextParseTools as tpt
    import TextAnalysisTools as tat

    class Monologue:

        def __init__(self, rawSentence, speaker):

            self.rawMonologue = rawSentence
            self.speaker = speaker
            self.cleanMonologue = tpt.cleanText(self.rawMonologue)
            self.wordCount = tat.wordCount(self.cleanMonologue)
            self.vocabularyCount = tat.vocabCount(self.cleanMonologue)
            self.uniqueVocabularyCount = tat.uniqueVocabCount(self.cleanMonologue)
            self.typeTokenRatio = tat.typeTokenRatio(self.vocabularyCount,
self.wordCount)
            self.brunetsIndex = tat.brunetsIndex(self.vocabularyCount, self.wordCount)
            self.honoresStatistic = tat.honoresStatistic(self.wordCount,
self.vocabularyCount, self.uniqueVocabularyCount)
            self.POSDictionary = tat.POSRates(self.cleanMonologue)
            if self.wordCount == 0:
                self.nounRate = 0
                self.pronounRate = 0
                self.adjectiveRate = 0
                self.verbRate = 0
                pauseCalcs = 0
                self.pauseFrequency = 0
                self.pauseAverageLength = 0
            else:
                self.nounRate = tat.nounRate(self.POSDictionary)/self.wordCount
                self.pronounRate = tat.pronounRate(self.POSDictionary)/self.wordCount
                self.adjectiveRate =
tat.adjectiveRate(self.POSDictionary)/self.wordCount
                self.verbRate = tat.verbRate(self.POSDictionary)/self.wordCount
                pauseCalcs = tat.pauseCalculations(self.rawMonologue)
                self.pauseFrequency = pauseCalcs[0]
                self.pauseAverageLength = pauseCalcs[1]
            #printValues(self)


    def printValues(self):
            print 'MONOLOGUE VALUES FOR - ' + str(self.speaker) + self.cleanMonologue
            print '           WordCount:' + str(self.wordCount)
            print '          VocabCount:' + str(self.vocabularyCount)
            print '    UniqueVocabCount:' + str(self.uniqueVocabularyCount)
            print '      TypeTokenRatio:' + str(self.typeTokenRatio)
            print '      Brunet\'s Index:' + str(self.brunetsIndex)
            print '  Honore\'s Statistic:' + str(self.honoresStatistic)
            print '            NounRate:' + str(self.nounRate)
            print '         PronounRate:' + str(self.pronounRate)
```

```
            print '       AdjectiveRate:' + str(self.adjectiveRate)
            print '            VerbRate:' + str(self.verbRate)

    #textFileTools
    #contains all methods for reading and parsing a textFile
    import nltk
    import re
    import codecs
    import math
    import unicodedata as ud

    #Divides the text by speakers, which are identified by the handles, returns list
where each individual speakers'
    #dialog is a separate slot:
    #Handles need to be identified before the conversation can be parsed.
    #Recorded handles are as follow from Finding a balance: The Carolinas Conversation
Collection
    #And are broken in either of two groups - interviewer or speaker (person suffering
dementia)
    #Acronyms identify Role(Interviewer/Speaker), Race(Black/White), and Gender
(Male/Female)
    #Interviewer
    #['IWF','IWM','IBM','IBF']

    #Speaker
    #['SBM', 'SBF','SWM','SWF']
    def divideBySpeaker(textList, corpusType):

        dividedMonologueList = []
        ########################################
        #If using switchBoard Conversations - UNCOMMENT THIS!!
        ########################
        if corpusType == 1:
            textList = switchboardListClean(textList)

        speakerList = getSpeakers(textList, corpusType)

        if corpusType == 1:
            #Divide strings by speaker and add speakers to speakerList
            #grab the first word from every line, if it matches re.find('.*:') but not
(':.*') then it is a name.
            #make sure to compensate for those that start with ('\ufeff')
            currentBuild = textList[0] #currentBuild is comprised of those lines in
the text that are continuations of eachother until the next speaker
            for index in range(1,len(textList)):
                speakerFound = 0
                for speaker in speakerList:
                    if speaker in textList[index]:
                       speakerFound = 1
                if speakerFound == 1:
                    dividedMonologueList.append(currentBuild)
                    currentBuild = textList[index]
                else:
                    currentBuild = unicode(currentBuild) + u' ' +
unicode(textList[index])

            dividedMonologueList.append(currentBuild)
```

```
        else:#CCC Corpus. All lines are already divided by speaker, so no appending
needed.
            for index in range(len(textList)):
                wordList = textList[index].split(None)
                wordList.pop(0)
                wordList.pop(0)
                dividedMonologueList.append(unicode.upper(' '.join(wordList)))
        #Remove empty monologues from the list
        deleteList = []
        for monologue in dividedMonologueList:
            if len(monologue.split(None)) <= 1:
                deleteList.append(monologue)
        for item in deleteList:
            dividedMonologueList.remove(item)

        #Remove false speakers from list
        dividedMonologueList = removeFalseSpeakerDialogue(dividedMonologueList)
        speakerList = removeFalseSpeakers(speakerList)
        return (dividedMonologueList, speakerList)

    #Grabs the raw text from a file
    def getTextFromFile(textFileDir):
        testFile = codecs.open( textFileDir, "r", "utf-8" )
        u = testFile.readlines()
        return u

    #LIST OF TOKENS
    ########################
    #
       #'['     Indicates point of overlap onset
       #']'     Indicates the point at which an utterance or utterance-part terminates
       #'='     One at the end of one line and one at the beginning of a next,
indicate no 'gap' between the two lines. This is often called latching
       #'(#.#)' Indicates elapsed time in silence by tenth of seconds
       #'(.)'   Indicates a tiny 'gap' within or between utterances
       #'___'   Underscore indicates some form of stress, via pitch and/or ampitude;
an alternate method it to print the stressed part in italic.
       #'::'    Colons indicate prolongation of the immediately prior sound. Multiple
colons indicate a more prolonged sound.
       #'-'     Dash indicates a cut-off
       #'.'     Indicates a stopping fall in tone
       #','     Comma indicates a continuing intonation, like when you are reading
items from a list.
       #'?'     Indicates a rising intonation
       #        Combined Question Mark/Comma indicates stronger rise than a comma but
weaker than a question mark
       #' '     The adsence of an utterance-final marker indicates some sort of
'indeterminate' contour
       #'? ?'   Arrows indicate marked shifts into higher or lower pitch in the
utterance part immediately following the arrow.
       #'WORD'  Uppercase indicates especially loud sounds relative to the surrounding
talk
       #'DEGREE SYMBOL'     Utterance or utterance parts bracketed by degree signs are
relatively quieted than the surrounding talk
       #'<>'    Right/left carets bracketing an utterance or utterance-part indicate
speeding-up
       #'?hhh'  A dot-prefixed row of hs indicates an inbreath. Without the dot, the
hs indicate an outbreath
```

```
      #'w(h)ord'   A parenthesized h, or a row of hs within a word, indicates
breathiness, as in laughter, crying, etc.
      #'()'    Empty parentheses indicate transcriber's inability to hear what was
said. The length of the paranthesized space indicates the length
      #        of the untranscribed talk. In the speaker designation column, the
empty parentheses indicate inability to identify a speaker.
      #'word'  Parenthesized words are especially dubious hearings or speaker
identifications
      #'(())'  Double parentheses contain transcriber's descriptions rather than, or
in addition to, transcriptions.

   #finds the tokens within a list of strings: parameter is a List of strings divided
by speaker
   #These token are as defined in Appendix A of "Doing Conversation Analysis" by Paul
Ten Have
   def parseTokens(sentenceList):
       reTokenList = ['\[=', '=\]', ':::', '\(.*\.[0-9]\)', '<.*>', ',', '\[.*\]']
       foundInstances = []
       for item in sentenceList:
           for token in reTokenList:
               if re.match(token,item) != -1:
                   foundInstances.append(re.findall(token,item))

           print '(1)Item:'
           print item
           print '(2)List: '
           print foundInstances
           print ''
           del foundInstances[:]

   #use NLTK Text and Word Count (len)
   def NLTKConvertAndCount(textFile):
       string = getTextFromFile(textFile)
       tokenList = string.split(' ')
       print tokenList
       textForm = nltk.Text(tokenList,'conversation')
       textForm.concordance(':')

   ###################
   #Remove C.A. Tags from string so string can be parsed and tokenized (SUCCESS)
   def cleanText(rawText):
       rawText = unicode.upper(unicode(rawText))
       reTokenList = ['\[=', '=\]',':::', '::','\[.*\]', '\([0-9].[0-9]\)','\(.[0-
9]\)', '<','>', '\[.*','.*\]', '/','\(.*\)', '\.*', '\?', '-', ',']
       switchList = {u'WANTA':u'WANT TO',
                     u'ISN\u2019T':u'IS NOT',
                     u'IT\u2019S':u'IT IS',
                     u'THAT\u2019S':u'THAT IS',
                     u'IDN\u2019T':u'IS NOT',
                     u'\ufeff':u''}
       for tag in reTokenList:
           foundList = re.findall(tag,rawText)
           for item in foundList:
               rawText = rawText.replace(item,'')

       for switchListKey in switchList.keys():
           if switchListKey in rawText:
               rawText = rawText.replace(switchListKey, switchList[switchListKey])
```

```
    rawText = ' '.join(rawText.split())
    return rawText

#removes unicode symbols from dialogue
def cleanUnicode(rawText):
    return ''.join([x for x in rawText if ord(x) < 128])

#returns the labels of all the speakers in a conversation
def getSpeakers(rawText, corpusType):
    if corpusType == 1: #switchBoard corpus
        speakerList = []
        for line in rawText:
            word = line.split(None)[0]
            if u'\ufeff' in word:
                word = word.replace(u'\ufeff','')
            otherResults = re.findall('::',word)
            nameResults = re.findall('.*:',word)
            if len(nameResults) == 1 and len(otherResults) == 0:
                if word not in speakerList:
                    speakerList.append(word)
        return speakerList
    else: #CCC corpus
        speakerList = []
        for line in rawText:
            lineList = line.split(None)
            if len(lineList) > 0:
                possibleName = unicode.upper(lineList[2])
                if possibleName not in speakerList:
                    speakerList.append(possibleName)
        return speakerList

#If any false speakers are encountered within a dialogue, have them removed
def removeFalseSpeakerDialogue(dialogueList):
    falseSpeakers = ['ENV:']
    removeLines = []
    for line in dialogueList:
       for speaker in falseSpeakers:
           if speaker in line:
               if line not in removeLines:
                removeLines.append(line)
    for line in removeLines:
        dialogueList.remove(line)
    return dialogueList
def removeFalseSpeakers(speakerList):
    falseSpeakers = ['ENV:']
    for speaker in falseSpeakers:
       if speaker in speakerList:
          speakerList.remove(speaker)
    return speakerList

# Returns a list of all of a speaker's Dialog within a speaker-divided corpus list
def rawSpeakerDialogueList(conversationList, speaker):
    speakerList = []
    for item in conversationList:
        if speaker in item:
            namelessItem = item.replace(speaker,'')
            speakerList.append(namelessItem)
```

```
        return speakerList

    def switchboardListClean(rawList):
        cleanList = []
        reTokenList = ['@','=','\,','\[','\]','\?', '<', '>', 'VOX', '[0-9]', 'XP',
'PX', 'X', '%', '\$','!', 'COMMA','PERIOD', 'PEROID']
        #'-','\.',
        for line in rawList:
            lineList = line.split(None)
            newLine = ''
            removeParenthesisList = []
            reTokenInstances = []
            for index in range(2,len(lineList)):
                for reToken in reTokenList:
                    reTokenInstances.extend(re.findall(reToken,lineList[index]))
                for tokenInstance in reTokenInstances:
                    lineList[index] =
lineList[index].replace(unicode(tokenInstance),u'')

                #remove items in parenthesis
                if lineList[index].startswith(u'(') == 1 and
lineList[index].endswith(u')') == 1:
                    removeParenthesisList.append(lineList[index])
                newLine = newLine + ' ' + lineList[index]
            for parenthesisItem in removeParenthesisList:
                newLine = newLine.replace(parenthesisItem,u'')
            newLine = ' '.join(newLine.split())
            if len(newLine) != 0:
                cleanList.append(newLine)
        return cleanList

    import nltk
    import re
    import codecs
    import math
    from nltk.corpus import wordnet as wn

        ################################
        #Confidence Determinant
        ########################
        #Each sentence by the speaker is given a confidence value based on the tokens
within (pauses, rise and fall in
           # tone) which are to used to calculate the confidence value.
           # method will be given a list of the conversation divided by speaker. Only
interested in those sentences
           # spoken by dementia patient. Will return dictionary with sentence as key
and confidence level as value.
    def ConfidenceDeterminant(speakerList, interviewer, speaker):
        print ''

        #These are the UTF-8 Conversione Codes
        ####
        #       (Up Arrow) = \u2191
        #     (Down Arrow) = \u2193
        #              (') = \u2019 (Don't need yet)

        ################################
        #Word Count
```

```
        ##########################
        #Counts the number of words in a a specified talker's (Interviewer or
Speaker(DAT)) dialogue. Returns a value
    def wordCount(monologueString):
        wordList = monologueString.split(None)
        dialogWordCount = len(wordList)
        return dialogWordCount

      ########################
      #Vocabularity Count
      ########
      #Calculates the vocabulary count and adds the value to the dictionary value[1]
    def vocabCount(monologueString):
        wordList = []
        countList = []
        lowerCaseString = unicode.lower(unicode(monologueString))
        wordList = lowerCaseString.split(None)
        for word in wordList:
                if word not in countList:
                    countList.append(word)
        return len(countList)

    ######################
    #Unique Vocab Count
    #########
    #This method determines how many words within the dialogue appear once and only
once, and adds final value to [2]
    def uniqueVocabCount(monologueString):

        uniqueWordCount = {}
        uniqueCount = 0
        wordList = monologueString.split(None)
        for word in wordList:
            word = unicode.lower(word)
            if word not in uniqueWordCount:
               uniqueWordCount[word] = 1
            else:
               uniqueWordCount[word] = 2

        wordKeys = uniqueWordCount.keys()
        for uniqueWord in wordKeys:
            if uniqueWordCount[uniqueWord] == 1:
                uniqueCount = uniqueCount + 1

        return uniqueCount

        ##########################
        #Type-Token Ratio
        ############
        #This does a straight ratio comparison between the vocabultar count versus
word count of a sentence (ie V/W)
        # Input is to be Dictionary from Wordcount.
    def typeTokenRatio(vocabSize, wordCount):
            if vocabSize == 0 or wordCount == 0:
                return 0
            tokenRatio = float(vocabSize)/float(wordCount)
            return tokenRatio
```

```
        ####################
        #Brunet's Index
        #############
        #Unlike Type-Token Ratio, this quantifies lexical richness without being
sensitive to text length. It is calculated
        #using this equation : W = N^(V-0.165)
    def brunetsIndex(vocabCount, wordCount):
        if vocabCount == 0 or wordCount == 0:
                return 0
        bIndex = float(pow(wordCount,pow(vocabCount, -0.165)))
        return bIndex

    def honoresStatistic(wordCount, vocabCount, uniqueVocabCount):
        if vocabCount == 0 or wordCount == 0 or uniqueVocabCount == 0:
                return 0
        if uniqueVocabCount == vocabCount:
            r = (100 * math.log(wordCount))
        else:
            r = (100 * math.log(wordCount)) /(1 - (uniqueVocabCount/vocabCount))
        return r

    # Tags for Nouns = NN,NNS,NNP,NNPS, RegExp = N.*
    # Tags for Pronouns = PRP,PRP$, NO RegExp
    # Tags for Adjectives = JJ,JJR,JJS, RegExp = J.*
    # Tags for Verbs = VB, VBD, VBG, VBN, VBP, VBZ, RegExp = V.*
    #
    def POSRates(textString):
        from nltk import pos_tag, word_tokenize
        taggedItem = []
        tagDict = {}
        wordList = textString.split(None)
        for word in wordList:
            taggedItem = pos_tag(word_tokenize(word))
            if (taggedItem[0])[1] not in tagDict:
                tagDict[taggedItem[0][1]] = 1
            else:
                tagDict[taggedItem[0][1]] = tagDict[taggedItem[0][1]] + 1

        return tagDict


    def nounRate(posDict):
        tagDictKeys = posDict.keys()
        nounRate = 0
        for key in tagDictKeys:
            if re.match('N.*',key):
                nounRate = nounRate + posDict[key]
        return nounRate

    def pronounRate(posDict):
        tagDictKeys = posDict.keys()
        pronounRate = 0
        for key in tagDictKeys:
            if re.match('PRP.*',key):
                pronounRate = pronounRate + posDict[key]
        return pronounRate

    def adjectiveRate(posDict):
```

```
        tagDictKeys = posDict.keys()
        adjectiveRate = 0
        for key in tagDictKeys:
            if re.match('J.*',key):
                adjectiveRate = adjectiveRate + posDict[key]
        return adjectiveRate

    def verbRate(posDict):
        tagDictKeys = posDict.keys()
        verbRate = 0
        for key in tagDictKeys:
            if re.match('V.*',key):
                verbRate = verbRate + posDict[key]
        return verbRate

    #Since pauseCount and averagePauseLength require the same data, this returns a
list with [0]pauseCount and [1]averagePauseLength
    def pauseCalculations(rawMonologueString):

        ##############################
        #If using switchBoard, set to 1, else to 0
        switchBoardCorpus = 0
        ##############
        if switchBoardCorpus == 1:
            return switchBoardPauseCalculations(rawMonologueString)
        else:
            return CCCPauseCalculations(rawMonologueString)


    def switchBoardPauseCalculations(rawMonologueString):
        #print rawMonologueString
        pauseDictionary = {u'...':2.0,
                           u'..':1.0,
                           u'---':2.0,
                           u'--':1.0}
        foundInstances = list()
        for token in pauseDictionary.keys():
            for word in rawMonologueString.split(None):
                if token == unicode(word):
                    foundInstances.append(token)
        if not foundInstances:
            return (0,0)
        pauseCount = len(foundInstances)
        totalPauseLength = 0.0
        for pauseStringRaw in foundInstances:
            if pauseStringRaw == u'...' or pauseStringRaw == u'---':
                totalPauseLength = totalPauseLength + 2.0
            else:
                totalPauseLength = totalPauseLength + 1.0
        pauseAverage = totalPauseLength/pauseCount
        return (pauseCount, pauseAverage)

    def CCCPauseCalculations(rawMonologueString):
        return (0,0)

    # Count the numbers of repeated words (i.e. "I... I didn't" but not "this is...
this is"
    def repeatedWordCount(cleanConversationString):
```

```
        stringSize = 2
        repeatCountDict = {}
        for index in range(1,stringSize + 1):
            repeatCountDict[index] = 0

        wordList = unicode.upper(cleanConversationString).split(None)
        for stringSizeIndex in range(1,stringSize + 1):
            for index in range(0,len(wordList) - stringSizeIndex - 1):
                firstString = u''
                secondString = u''
                for firstIndex in range(index,index+stringSizeIndex):
                    firstString = firstString + wordList[firstIndex] + u' '
                for secondIndex in range(index+stringSizeIndex, index+stringSizeIndex
+ stringSizeIndex):
                    secondString = secondString + wordList[secondIndex] + u' '
                if  firstString == secondString:
                    #print u'Part One: ' + firstString + u'   Part Two: ' +
secondString
                    repeatCountDict[stringSizeIndex] =
repeatCountDict[stringSizeIndex] + 1

        return repeatCountDict

    #NOT FINISHED!!!!
    def fillerCount(rawText):
        positiveFillers = [u"UM-HMM", u"UM-HM", u"YEP", u"MMHMM"]
        negativeFillers = []
        nonDescriminativeFillers = [u"UHM", u"I MEAN", u"AH", u"MM", u"ER", u"UH"]
        print ''

    def paraphrasingComparison(sentenceList, speakerList, corpusType):
        #Generate the tables needed for data comparison
        #Use names from speakerList as Keys for dictionaries

        utteranceCountDict = {}
        numberOfGoAheads = 0
        forwardScoreListDict = {}
        backwardsScoreListDict = {}
        numberOfGoAheadsCountDict = {}
        for index in range(0,len(sentenceList)):
            sentenceList[index] = unicode.upper(sentenceList[index])

        for speaker in speakerList:
            utteranceCountDict[speaker] = 0
            forwardScoreListDict[speaker] = []
            backwardsScoreListDict[speaker] = []
            numberOfGoAheadsCountDict[speaker] = 0


        bestScore = 0.0
        bestWord = ''

        #first part is to determine if the sentences are valid for comparison
        for index in range(0,len(sentenceList)):
            speaker = sentenceList[index].split(None)[0]
            if speaker in utteranceCountDict.keys():
                utteranceCountDict[speaker] = utteranceCountDict[speaker] + 1
                if isAGoAhead(sentenceList[index]) == 0:
```

```
                    previousMonologueResults =
grabPreviousMonologue(sentenceList,index)
                    nextMonologueResults = grabNextMonologue(sentenceList,index)
                    #print '*********************************'
                    if previousMonologueResults[1] == 1:
                        scoreList =paraphraseExecution(sentenceList[index],
previousMonologueResults[0])
                        backwardsScoreListDict[speaker].append(average(scoreList))
                    #    print unicode('PreviousDialogue') +
previousMonologueResults[0]
                    #    print scoreList
                    #    print 'Average: ' + str(average(scoreList))
                    #
                    #print unicode('MainDialogue') + sentenceList[index]

                    if nextMonologueResults[1] == 1:
                        scoreList = paraphraseExecution(sentenceList[index],
nextMonologueResults[0])
                        forwardScoreListDict[speaker].append(average(scoreList))
                        #print scoreList
                        #print 'Average: ' + str(average(scoreList))
                        #print unicode('ForwardDialogue') + nextMonologueResults[0]
                else:
                     numberOfGoAheads = numberOfGoAheads + 1
                     numberOfGoAheadsCountDict[speaker] =
numberOfGoAheadsCountDict[speaker] + 1
            #print
'_____________________________________________________________________________________
_______'
            #print 'Conversation Results'
            #print 'NumberofUtterances:' + str(len(sentenceList))
            #print 'NumberofGoAheads:' + str(numberOfGoAheads)

        for speaker in speakerList:
            print speaker + ' '+ str(utteranceCountDict[speaker])+ ' ' +
str(numberOfGoAheadsCountDict[speaker])+ ' ' +
str(average(backwardsScoreListDict[speaker]))+ '
'+str(average(forwardScoreListDict[speaker]))

    def paraphraseExecution(mainMonologue, secondaryMonologue):
        pronounReflexives = {u'EVERYBODY':[u'EVERYONE',u'THEY',u'THEM'],
                                u'EVERYONE':[u'EVERYBODY',u'THEY',u'THEM'],
                                u'HE':[u'HE',u'HIS', u'HIM',
u'HIMSELF',u'HISSELF',u'THEY',u'THEIR',u'THEM',u'THEIR',u'THEIRS'],

u'HER':[u'SHE',u'HER',u'HERSELF',u'THEY',u'THEIR',u'THEM',u'THEIRS',u'HERS'],

u'HERS':[u'SHE',u'HER',u'HERSELF',u'THEY',u'THEIR',u'THEM',u'THEIRS',u'HERS'],

u'HERSELF':[u'SHE',u'HER',u'HERSELF',u'THEY',u'THEIR',u'THEM',u'THEIRS',u'HERS'],
                                u'HIM':[u'HE',u'HIS', u'HIM',
u'HIMSELF',u'HISSELF',u'THEY',u'THEIR',u'THEM',u'THEIR',u'THEIRS'],
                                u'HIMSELF':[u'HE',u'HIS', u'HIM',
u'HIMSELF',u'HISSELF',u'THEY',u'THEIR',u'THEM',u'THEIR',u'THEIRS'],
                                u'HIS':[u'HE',u'HIS', u'HIM',
u'HIMSELF',u'HISSELF',u'THEY',u'THEIR',u'THEM',u'THEIR',u'THEIRS'],
                                u'HISSELF':[u'HE',u'HIS', u'HIM',
u'HIMSELF',u'HISSELF',u'THEY',u'THEIR',u'THEM',u'THEIR',u'THEIRS'],
```

```
u'I':[u'YOU',u'YOUR',u'Y\'ALL',u'YOURSELF',u'YALL',u'YALLS',u'Y\'ALLS',u'YOURSELVES',u
'YOURS'],
                                u'IT':[u'IT',u'THIS',u'THAT'],
                                u'ITS':[],
                                u'ITSELF':[],
                                u'ME':[u'YOU', u'THAT'],
                                u'MINE':[],
                                u'MY':[u'YOU',u'YOUR',u'YOURS'],
                                u'MYSELF':[],
                                u'NOBODY':[],
                                u'NOTHING':[],
                                u'OTHERS':[],
                                u'OUR':[],
                                u'OURS':[],
                                u'OURSELVES':[],
                                u'SHE':[u'SHE', u'HER'],
                                u'SOMEBODY':[],
                                u'SOMEONE':[],
                                u'SOMETHING':[],
                                u'THAT':[u'IT', u'THAT', u'THIS'],
                                u'THEIRS':[],
                                u'THEM':[],
                                u'THEMSELVES':[],
                                u'THESE':[],
                                u'THEY':[],
                                u'THIS':[u'THAT',u'IT'],
                                u'THOSE':[],
                                u'US':[u'YOU'],
                                u'WE':[u'YOU'],
                                u'WHICHEVER':[],
                                u'WHO':[u'IT',u'WHO'],
                                u'WHOEVER':[],
                                u'WHOM':[u'IT',u'WHO',u'I'],
                                u'WHOMEVER':[],
                                u'WHOSE':[],

u'YOU':[u'I',u'WE',u'ME',u'US',u'OURSELVES',u'OURSELF',u'MYSELF'],
                                u'YOURS':[u'MY',u'MINE',u'OUR', u'OURS', u'ME'],
                                u'YOUSELF':[],
                                u'YOURSELVES':[]}
        mainMonologueScoreList = []
        mainMonologueWordList = mainMonologue.split(None)
        secondaryMonologueWordList = secondaryMonologue.split(None)
        mainMonologueWordList.pop(0)
        secondaryMonologueWordList.pop(0)
        mainMonologueWordListCopy = list(mainMonologueWordList)
        secondaryMonologueWordListCopy = list(secondaryMonologueWordList)

        #1-to-1 comparison
        for word in mainMonologueWordList:
            if word in secondaryMonologueWordList:
                mainMonologueWordListCopy.remove(word)
                mainMonologueScoreList.append(1.0)
                if word in secondaryMonologueWordListCopy:
                    secondaryMonologueWordListCopy.remove(word)

        #First 3-letter comparison
```

```
        mainMonologueWordList = list(mainMonologueWordListCopy)
        for word in mainMonologueWordList:
            for postWord in secondaryMonologueWordList:
                if len(word) > 3 and len(postWord) > 3 and isAPronoun(word) == 0 and
isAPronoun(postWord) == 0:
                    if word[0:3] == postWord[0:3]:
                        mainMonologueScoreList.append(0.98)
                        if word in mainMonologueWordListCopy:
                            mainMonologueWordListCopy.remove(word)

        #Pronoun Mapping
        mainMonologueWordList = list(mainMonologueWordListCopy)
        for word in mainMonologueWordList:
            if word in pronounReflexives.keys():
                for reflexive in pronounReflexives[word]:
                    if reflexive in secondaryMonologueWordList:
                        if word in mainMonologueWordListCopy:
                            mainMonologueScoreList.append(1.0)
                            mainMonologueWordListCopy.remove(word)

        #Synset Mapping
        mainMonologueWordList = list(mainMonologueWordListCopy)
        secondaryMonologueWordList = list(secondaryMonologueWordListCopy)
        for word in mainMonologueWordList:
            bestScore = 0.0
            mainWord = u''
            bestSecondWord = u''
            if isAPronoun(word) == 0:
                for mainSynset in wn.synsets(word):
                    for secondaryWord in secondaryMonologueWordList:
                        if isAPronoun(secondaryWord) == 0:

                            for secondarySynset in wn.synsets(secondaryWord):
                                if mainSynset.path_similarity(secondarySynset) >
bestScore:
                                    mainWord = word
                                    bestSecondWord = secondaryWord
                                    bestScore =
mainSynset.path_similarity(secondarySynset)
                                    bestMainSynset = mainSynset
                                    bestSecondarySynset = secondarySynset
                                    bestSecondaryWord = secondaryWord
            #print '**********************'
            #print u'mainWord:' + mainWord
            #print u'bestSecondaryWord:' + bestSecondWord
            #print u'Score:' + unicode(bestScore)
            mainMonologueScoreList.append(bestScore)
        return mainMonologueScoreList

    def average(numbers):
        sum = 0
        zeroCount = 0
        if len(numbers) == 0:
            return 0
        for number in numbers:
            sum += number
        if zeroCount == len(numbers):
            return 0
```

```
        mean = sum/len(numbers)
        return mean

    def isAPronoun(word):
        pronouns =
[u'EVERYBODY',u'EVERYONE',u'HE',u'HER',u'HERS',u'HERSELF',u'HIM',u'HIMSELF',u'HIS',u'I
', u'IT',u'ITS',u'ITSELF',u'ME',u'MINE',u'MY',u'MYSELF',

u'NOBODY',u'NOTHING',u'OTHERS',u'OUR',u'OURS',u'OURSELVES',u'SHE',u'SOMEBODY',u'SOMEON
E',u'SOMETHING',u'THAT',u'THEIRS',u'THEM',u'THEMSELVES',u'THESE',u'THEY',

u'THIS',u'THOSE',u'US',u'WE',u'WHICHEVER',u'WHO',u'WHOEVER',u'WHOM',u'WHOMEVER',u'WHOS
E',u'YOU',u'YOURS',u'YOUSELF',u'YOURSELVES']
        if word in pronouns:
            return 1
        else:
            return 0

    def isAGoAhead(monologue):

        monologueList = monologue.split(None)
        monologueList.pop(0)
        goAheadList =
[u'YES',u'YEAH',u'NO',u'AND',u'THEN',u'REALLY',u'GOOD',u'UHN',u'NOPE',u'RIGHT',u'SURE'
,

u'GREAT',u'HUNH',u'HUH',u'SO',u'COURSE',u'MAYBE',u'MAY',u'WELL',u'HM',u'HMM',u'YOU',u'
KNOW',

u'HEY',u'MMM',u'MHM',u'BECAUSE',u'WHY',u'WOW',u'COOL',u'OH',u'OHH',u'OOH',u'OOOH',u'DE
AR',u'WHAT',u'BUT',u'YOU',u'DID',u'NOT',u'YWN',u'ARE',u'UNHUNH'
                        ,u'UHHUH',u'UNHUH',u'PLEASE',u'WHAT']
        if len(monologueList) >= 5:
            return 0
        if len(monologueList) <= 2:
            return 1
        for word in monologueList:
            if word in goAheadList:
                return 1
        return 0

    def isSameSpeaker(sentenceOne, sentenceTwo):
        if sentenceOne.split(None)[0] == sentenceTwo.split(None)[0]:
            return 1
        return 0

    def grabPreviousMonologue(speakerList, index):
        for prIndex in range(index,-1, -1):
            if isAGoAhead(speakerList[prIndex]) == 0 and
isSameSpeaker(speakerList[prIndex],speakerList[index]) == 0:
                        return (speakerList[prIndex], 1)
        return ('',0)

    def grabNextMonologue(speakerList,index):
        for nxtIndex in range(index, len(speakerList)):
            if isAGoAhead(speakerList[nxtIndex]) == 0 and
isSameSpeaker(speakerList[nxtIndex],speakerList[index]) == 0:
                return (speakerList[nxtIndex], 1)
```

```
    return('',0)

def syllableRate(dialogueList):
    syllableRateList = []
    for line in dialogueList:
        splitLine = line.split(None)
        if len(splitLine) > 3:
            syllableRateList.append(splitLine[0])
    return syllableRateList

def averageSyllableRate(syllableRateList):
    if len(syllableRateList) == 0:
        return 0
    sum = 0.0
    zeroCount = 0
    for rate in syllableRateList:
        if rate == 0.0:
            zeroCount = zeroCount + 1
        else:
            sum = sum + float(rate)
    if len(syllableRateList) == zeroCount:
        return 0
    return sum/(len(syllableRateList) - zeroCount)

def hiccupCount(rawDialogueList):
    hiccups = 0
    words = rawDialogueList.split(None)
    for word in words:
        if word[-1] == u'-' and len(word) > 1 and word[0] != u'-':
            hiccups += 1
    return hiccups
```

References

Baron, D.; Shriberg, E. & Stolcke, A. (2002), Automatic punctuation and disfluency detection in multi-party meetings using prosodic and lexical cues, *in* 'Seventh International Conference on Spoken Language Processing'.

Bird, S.; Klein, E. & Loper, E. (2009), *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*, O'Reilly, Beijing.

Bortfeld, H.; Leon, S.; Bloom, J.; Schober, M. & Brennan, S. (2001), 'Disfluency rates in conversation: Effects of age, relationship, topic, role, and gender', *Language and Speech* **44**(2), 123--147.

Bourgeois, M. (1990), 'Enhancing conversation skills in patients with Alzheimer's disease using a prosthetic memory aid.', *Journal of Applied Behavior Analysis* **23**(1), 29.

Bucks, R. S.; Singh, S.; Cuerden, J. M. & Wilcock, G. K. (2000), 'Analysis of spontaneous, conversational speech in dementia of Alzheimer type: Evaluation of an objective technique for analysing lexical performance', *Aphasiology* **14**(1), 71-91.

Cuello, A. (2008), *Pharmacological mechanisms in Alzheimer's therapeutics*, Springer Verlag.

Davis, B. (2007), 'Culturally Competent Materials on Communication and Dementia: Year Two. Charlotte: University of North Carolina'.

Davis, B. & Maclagan, M. (2009), 'Examining pauses in Alzheimer's discourse', *American journal of Alzheimer's Disease and other dementias* **24**(2), 141--154.

Dempster, M.; Alm, N. & Reiter, E. (2010), Automatic generation of conversational utterances and narrative for Augmentative and Alternative Communication: a prototype system, *in* 'Proceedings of the NAACL HLT 2010 Workshop on Speech and Language Processing for Assistive Technologies', pp. 10--18.

Godfrey, J.; Holliman, E. & McDaniel, J. (1992), SWITCHBOARD: Telephone speech corpus for research and development, *in* 'Acoustics, Speech, and Signal Processing, 1992. ICASSP-92., 1992 IEEE International Conference on', pp. 517--520.

Lickley, R. (1996), Juncture cues to disfluency, *in* 'Spoken Language, 1996. ICSLP 96. Proceedings., Fourth International Conference on', pp. 2478--2481.

Liu, Y.; Shriberg, E. & Stolcke, A. (2003), Automatic disfluency identification in conversational speech using multiple knowledge sources, *in* 'Eighth European Conference on Speech Communication and Technology'.

Liu, Y.; Shriberg, E.; Stolcke, A.; Hillard, D.; Ostendorf, M. & Harper, M. (2006),

'Enriching speech recognition with automatic detection of sentence boundaries and disfluencies', *Audio, Speech, and Language Processing, IEEE Transactions on* **14**(5), 1526--1540.

Mace, N. L., & Rabins, P. V. (2006). The 36-hour day: A family guide to caring for people with Alzheimer disease, other dementias, and memory loss in later life (4th ed.). Baltimore, MD: Johns Hopkins University Press.

Ostuni, E., & Santo Pietro, M. J. (1986). Getting through: Communicating when someone you care for has Alzheimer's disease. Princeton Junction, NJ: The Speech Bin.

Phinney, A. (2002), 'Living with the symptoms of Alzheimer's disease', *The person with Alzheimer's disease: Pathways to understanding the experience*, 49--74.

Pope, C. & Davis, B. (2011), 'Finding a balance: The Carolinas Conversation Collection', *Corpus Linguistics and Linguistic Theory* **7**(1), 143--161.

Princeton University "About WordNet." WordNet. Princeton University. 2010. <http://wordnet.princeton.edu>

Ray, S.; Britschgi, M.; Herbert, C.; Takeda-Uchimura, Y.; Boxer, A.; Blennow, K.; Friedman, L. F.; Galasko, D. R.; Jutel, M.; Karydas, A.; Kaye, J. A.; Leszek, J.; Miller, B. L.; Minthon, L.; Quinn, J. F.; Rabinovici, G. D.; Robinson, W. H.; Sabbagh, M. N.; So, Y. T.; Sparks, D. L.; Tabaton, M.; Tinklenberg, J.; Yesavage, J. A.; Tibshirani, R. & Wyss-Coray, T. (2007), 'Classification and prediction of clinical Alzheimer's diagnosis based on plasma signaling proteins', *Nat Med* **13**(11), 1359--1362.

Reisberg B, Sclan SG, Franssen EH, *et al* (1994). 'Clinical stages of normal aging and Alzheimer's disease: the GDS staging system'. *Neurosci Res Commun* 1993;13 (Suppl 1):551-4

Singh, S. (1996), 'Neural Networks In Spontaneous Speech Assessment Of Dysphasic Patients', .

Singh, S. & Bookless, T. (1997), 'Analysing spontaneous speech in dysphasic adults', *International Journal of Applied Linguistics* **7**(2), 165--181.

Singh, S.; Bucks, R. & Cuerden, J. (2001), 'Evaluation of an objective technique for analysing temporal variables in patients with Alzheimer's spontaneous speech', *Aphasiology* **15**(6), 571--583.

Snover, M.; Dorr, B. & Schwartz, R. (2004), A lexically-driven algorithm for disfluency detection, *in* 'Proceedings of HLT-NAACL 2004: Short Papers', pp. 157--160.

Schwarz, R. (2008), 'Cell Phone Communication Versus Face-to-Face Communication: The Effect of Mode of Communication on Relationship Satisfaction and the Difference in

Quality of Communication', PhD thesis, Kent State University.

Taylor, A.; Marcus, M. & Santorini, B. (2003), 'The Penn treebank: an overview', *Treebanks*, 1--22.

Ten Have, P. (2007), *Doing conversation analysis*, Sage Publications Ltd.

Waller, A. (2006), 'Communication access to conversational narrative', *Topics in Language Disorders* **26**(3), 221.

Wisenburn, B. & Higginbotham, D. (2009), 'Participant evaluations of rate and communication efficacy of an AAC application using natural language processing', *Augmentative and Alternative Communication* **25**(2), 78--89.

Yaruss, J. (1998), 'Real-time analysis of speech fluency: Procedures and reliability training', *American Journal of Speech-Language Pathology* **7**(2), 25.