Summer 7-16-2018

A Path To Alignment

Gregory Richard Arnold
University of New Mexico

Gregory R. Arnold
Candidate
Linguistics
Department

This thesis is approved, and it is acceptable in quality and form for publication:

Approved by the Thesis Committee:

Sherman Wilcox, Chairperson

Melissa Axelrod

Jill Morford

A PATH TO ALIGNMENT

BY

GREGORY R ARNOLD

B.A. Linguistics, Seattle Pacific University, 1985

THESIS

Submitted in Partial Fulfillment of the
Requirements for the Degree of

Master of Arts
Linguistics

The University of New Mexico
Albuquerque, New Mexico

July, 2018

DEDICATION

In memory of Wayne Otto, a life well-lived, an avid reader, a professor of education, a mentor of inquisitive pursuits and pastiche. We will miss you dearly.

ACKNOWLEDGMENTS

I heartily acknowledge Dr. Sherman Wilcox, my advisor and thesis chair, for continuing to encourage me to not give up writing and rewriting this thesis, and introducing me to cognitive linguistics. His guidance and professional style will remain with me as I continue my career. I also thank my committee members, Dr. Melissa Axelrod, and Dr. Jill Morford, for their valuable recommendations pertaining to this study and their assistance with professional development. Gratitude is extended to Sandia National Laboratories for the funding to pursue this research. To Dr. William Croft for his spurning instruction in typology, morphosyntax and semantics, it was foundationally essential. To Bruce Carpenter for engaging me in countless conversations about consciousness. And finally, to my wife and best friend, Eleni Otto, your love and support is the greatest gift of all.

A PATH TO ALIGNMENT

By

Gregory R Arnold

B.A., Linguistics, Seattle Pacific University, 1985

ABSTRACT

What is really needed to make a machine into a verisimilitude of a language using human? Clearly there are holes in human communication, missing linguistic forms, and yet we manage to convey meaning. The under-determinacy of language seems to play an integral part in the adaptive system that all humans possess for perceiving, processing and producing language with shared semantic value. We invent symbols that index the missing contextual elements, allowing partial production of linguistic units. A machine would require the same abilities of indexicality and inventiveness.

In this pilot study, I attempt to understand how semantic values shift and align during conversation with the further hope of developing a model for a computer to be able to interact with a human. I draw on the data available from YouTube closed-caption text, and build a corpus of discourses with the aim of developing meta-data in the form of dimensional values. This data represents the temporal flow of usage events with a semantic value system that seems to prod the activation of more usage events, align or misalign coordinated meaning during the semiotic cycle. I finally propose a usage-based data driven application, Chatbot that stores the tokens of conversation between a person and the computer as symbolic units for memory and exemplars for construction.

Table of Contents

List of Figures

List of Tables

Preface

My thesis asserts that the “meeting of minds”, coordination of meaning, is achieved through common lexical knowledge shared between two or more people engaged in conversation, i.e. the semiotic cycle (Steels, 2016, pp. 3-4)). Under-determinacy in language production (Everett, 2017, pp. 3-4,66,251,256) indicates there is a clear human invention of symbols, from the early cave drawings to the present day, which refer to or index missing contextual elements and semantic values. There is something else “prodding” activation in language processing, however, when attaining “joint attention” or a fixpoint in conceptual space (Gärdenfors, 2014, pp. 91,260,272-275).

Mutual alignment or un-alignment of meaning occurs; linguistic categorization (memory, learning and performance) is biased dynamically by a set of one or more value systems (dimensions), re-entrant mapping of motor activities on sensory information of an individual speaker or hearer (Thelen & Smith, 2000, p. 160). This is evident because alignment of meaning can fail even with common lexical knowledge. Also, linguistic units of input that are different from linguistic units containing the intended meaning are often employed in conversation to activate coordination. The value systems hold the linguistic criteria, including indexing missing context for language processing, activation and selection of the best adaptive meaning.

1 Introduction

If I wanted to build a robot that could communicate with humans very much like a human would with all the irrational and variant behaviors that can be attributed to humankind, I would need a cognitively sufficient model that considered many forms of input, many sensorimotor mechanisms, many use-specific processing units with cooperation abilities, value systems that act as biases for processing, and a storage medium where processing units can be activated. Figure 1 shows three boxes modelling a body-centric process. The first box is the world-environment input that is abundant with various forms of waves and particles, e.g. light and sound. The second box contains the bodily sensorimotor mechanisms that human evolution has adapted, e.g. the senses, ears, eyes, skin and tongue. The third box embodies the human consciousness full of processing units that are clustered and inter-connected. Bi-directional arrows connect the three boxes as data flows into and out of the human machine.

Consciousness has been an elusive topic for a very long time in human years. It has been a topic of fireside discussion since human predecessors came down out of the trees. More recently cognitive and neuroscience research has aided in developing theories about how consciousness emerges; what goes on in the brain as part of a human interacting with the world. Language is one of the more central emergent observations as it is foundational to human communication. In computer science, many algorithms have been applied to human interaction via computer interfaces, AKA Natural Language Processing. This has been achieved by processing language data using Machine Learning algorithms, and then

applying the learned indexical knowledge to software programs like SIRI[1] and others. Technology has achieved perceptive machine capabilities to analyze and disseminate data from spoken, written and kinetic sources. Although this is truly amazing, it is not even close to approaching the Human Consciousness level of processing. Each Machine Learning algorithm is targeted to solve a specific problem, and then collectively used to manifest a verisimilitude of consciousness.

The human difference is observed when quick decisions are made, and sometimes not the most logical ones. There are many theories that show a multi-modal input can be processed by a machine quickly and probabilistically, based on massive amounts of indexed data, i.e. IBM's Watson[2], even to the extent of competing and winning in a popular trivia game show. Another current example is the computer in self-driving cars that can process input based on trained data sets, and learn from actual road experience. These applications are wondrous examples of modern utopia-driven utilitarian technology, but what about a machine that makes mistakes, or makes irrational decisions? Is it even possible to create a "Conscious" machine?

The situation of most interest to my thesis involves, in its simplest form, a three-step model for language processing. The first step is the processing of input, which takes as its input multiple raw signals (R where R is all the signals [0 to n]), and processes each of them into a unit (p where p is the construction (Croft, Radical Construction Grammar Syntactic Theory in Typological Perspective, 2001, p. 4) (Steels, 2013, p. 153) in the

---

[1] "Siri (pronounced /ˈsɪəri/) is an intelligent personal assistant, part of Apple Inc.'s iOS." https://en.wikipedia.org/wiki/Siri

[2] https://en.wikipedia.org/wiki/Watson_(computer)

form of a symbolic unit (Langacker, 2008)). The second step is the activation function (f(x) where x is the valuation of a product of contextual weight (w) and the indexical vector of (p), and summed with the bias or vector of value system calculations (Friston, Tononi, Reeke Jr., Sporns, & Edelman, 1994, p. 232)), which takes the processed unit and activates its usage against semantic domains (Gärdenfors, 2014, pp. 30-38). The third step is the result or output or response, which is a point in conceptual space (Gärdenfors, 2014, pp. 271-275) that refers or points to a production unit or units. These three fundamental steps are repeated in a semiotic cycle (Steels, 2016, pp. 3-4) until the point in conceptual space maps onto itself (f(x) = x), or the meaning is coordinated (Gärdenfors, 2014, p. 94), of course this mapping may or may not happen in a discourse because of any number of interruptions or divergent meanings. Consciousness emerges, like foam atop a cold pint of beer[3], from the collinear, multi-cued dynamical system of usage of this language processing model (Onnis & Spivey, 2012, p. 140).

My research focuses on what steers or prods (Thelen & Smith, 2000, p. 160) the emergences of consciousness for more than one human when they attain joint attention (Gärdenfors, 2014, p. 92) while in conversation with each other. Edelman proposes that consciousness emerges as part of the re-entrant loop between value-category memory and current perceptual categorization, as shown in Figure 2.

Value systems could be modelled as the input to calculating a bias in the Neural Network algorithm affecting the actual linguistic activation as hinted at above in the robot analogy. I ask the following questions then to narrow the scope of my research. What is a

[3] Simile

Linguistic Value System, and more specifically what is the “Bias” that steers activation and selection? Can we identify sensorimotor sources of linguistic value, e.g. visual preference? In the next section I summarize some of the germane research already out there.

2 Background

The multi-disciplinary domain of cognitive science research is vast, contemporary-popular and extremely complex, somewhat overwhelming really. Neuroscience, psychology, philosophy, education and linguistics, for example, make up some of the areas of study. It is befitting for consciousness to have so much attention because of its emergent contribution to human evolution and global communications. The question of how a human brain works, by itself, has entire institutional focus yielding many benefits to humanity from curing diseases to adaptive technologies applied to sensory impairments. In the following sections I try to summarize some relevant studies of cognition from various disciplines to provide a stage for my proposed thesis and research.

2.1 Neuroscience

Edelman pioneered the TNGS[4] model that greatly furthers the study of consciousness. Neuronal group selection is the theory that value systems modulate synaptic changes to provide constraints for the selection of adaptive behaviors in somatic time. Value systems are modified with experience, a feedback loop of selection (re-entrant). Brain functions are mediated by:

i. Selectional events occurring among interacting cells in the developing embryo to form large repertoires of variant neural circuits
ii. Further selectional events occurring among populations of synapses to enhance those neuronal responses having adaptive value for the organism

[4] The Theory of Neuronal Group Selection (Edelman & Tononi, Universe of Consciousness, 2000, p. 83)

iii. re-entrant signals, exchanged via parallel and reciprocal connections, that serve through synaptic selection to integrate response patterns among functionally segregated brain areas in an adaptive fashion

Biases (innate values) constrain the selectional system instead of it being governed by pre-programmed rules or syntax. “The value of a global pattern of neuronal responses to a particular environmental situation (stimulus) is reflected in the capacity of that response pattern to increase the likelihood that it will recur in the same context.” (Friston, Tononi, Reeke Jr., Sporns, & Edelman, 1994, p. 230) Value is most effective when movement becomes part of the learning sequence. “An interesting consequence of value-dependent plasticity in afferents to sensory units in the model is that receptive field properties can change preferentially to sample cues having potential value.” (Friston, Tononi, Reeke Jr., Sporns, & Edelman, 1994, p. 237)

2.2 Philosophy

Embodied Cognition

It is all about a “Theory of Mind”, and two major theoretical camps that branch several times only to merge loosely as Radical Embodied Cognition (Chemero, 2011, pp. 17-24). The Representational Theory of Mind (RTM) is dialectically followed by the Eliminativism Theory of Mind (ETM). RTM has five tenets:

i. Proposition attitude states are relational
ii. Some relata are mental representations
iii. Mental representations are symbols (form and meaning)
iv. Mental representations have causal roles through form
v. Propositional attitudes get meaning from other object mental representations

A branch of RTM is computational, and asserts that computation is a rule-governed manipulation of symbols (CTM). This also involves the traffic of discrete tokens in the mind.

ETM works from a natural, ecological perspective and lays out three main assertions:

i. Perception is direct, i.e. no computation, no representation, no addition
ii. Perception is for guidance of action (perhaps a value system) i.e. always an action, we perceive to do
iii. Perception is of affordances, i.e. environmental opportunities for behavior, affordances can be both subjective and objective

A branch from this theory is situated semantics, or embodied cognition which is composed of indexicals (here, now, there, I, etc.), and the meaning of thoughts which are relationships between thinker and environmental information. The situation or thought is continuous in “act and check again” cognition. The applications that have sprung from this theory are numerous, including robotics, simulated evolution, developmental psychology to name a few.

Dynamical System Theory (DST), defined in Figure 3, comes out of situated semantics as well, and play a fundamental role in Radical Embodied Cognition. DST asserts a multi-variable input, each with a formula to calculate the action. “The agent produces representations that are geared toward the actions it performs from the beginning.” (Chemero, 2011, p. 27) The calculations are therefore differential equations that one might be tempted to view as part of the RTM camp, but the ETM camp, at this state of theory evolution, are not anti-representationalists. Instead they view these perception variables in an indexical-functional manner.

Radical Embodied Cognition (REC) and Embodied Cognition (EC) differ, therefore, in that REC evolves from ETM, and EC evolves from RTM, but both posit that embodied cognition is explained by way of tools like DST. REC rejects the idea of mental representation maps, and EC is more computationally tolerant. So, EC is defined as "Scientific study of perception, cognition, and action as necessarily embodied phenomenon, using explanatory tools that don't posit mental representations." (Chemero, 2011, p. 29) This is aligns well with the somatic processing model, Figure 1, where perception is the world, action is the body, and cognition is the brain, i.e. it models human existence philosophically.

Enactive Perception

The basic declaration of the enactivists is that they reject the idea that perception is a process inside the brain (Noë, 2006, p. 2). Instead perception is the bodily activity that humans engage in because of brain processes. The sensorimotor control in the brain enacts perception by usage in the form of embodied movement. Like what Edelman and others suggested that a value system maintained in neuronal groups is responsible for the apparent patterning of these perception activities. The so-called senses are used by the controlling neurons, not the other way around. Adaptive behavior of people with certain sensory deficits, such as blindness, shows how usage will shift to other sensual mechanisms to achieve the necessary perceptions (Noë, 2006, pp. 7-11). In other words, perception is a result of bodily actions.

Sensorimotor knowledge is then, if I understand this correctly, only the neurological pattern that would bodily reenact the perception. The implications are staggering regarding memories of what happened. The brain does not record the event itself, but

instead records the neuronal paths created by chemicals which provide a guidance, i.e. “Perception” for the sensorimotor action. The memory recall then recreates the pathways, and provides another, not likely identical, guidance. In other words, “perception” is a space in between thought and action that guides the action.

2.3 Computational Linguistics

Construction Grammar

There are three types of linguistic data considered in research by construction grammarians: introspective, observational and experimental. For the sake of evaluation of data on a continuum, because linguists love continuums, three dimensions of perception are considered: setting, stimulus and unit/response (Gries S. T., 2013, p. 94). The application of an introspective approach to data in a corpus involves a researcher evaluating “what sounds right”. This approach is somewhat prone to personal bias, and has been dismissed in favor of the other two approaches. Observational approaches to data look mostly at textually analyzable tokens, and provide statistical results for the three dimensions using frequencies of (co)occurrence, conditional probabilities, association strengths, and multi-factorial and multi-variance analysis. Experimental approaches include many studies involving psycholinguistic methods to elicit responses, such as priming effects. More recently there have been more uses of measure devices, like an EEG, to collect data that arises from production or perception of corpus data. These experimental activities would still fall at the natural end of the continuum, but here are also artificial dimensions used in studies involving computational linguistics and machine learning.

The models used in the latter experimental approach vary greatly in how activation, simulation and learning occurs in the artificial environment, but still provide for the three dimensions. “Finally, with the importance that usage plays in most contemporary incarnations of Construction Grammar, computational simulations of first-language acquisition or diachronic change will assume a more central role that they have done so far and (Edelman S. , 2007) surveys some notions relevant in this context.” (Gries S. T., 2015, p. 108). It is noted lastly that these approaches provide a richer toolbox for the Construction Grammarian.

Fluid Construction Grammar (FCG) “attempts to capture intuitions and theoretical notions from cognitive linguistics in general and Construction Grammar in particular.” (Steels, 2013, p. 153) FCG has been computational since 1998, and has yielded two main components: FCG-System and FCG-Interpreter. The system is embedded in the Common-LISP programming environment, and the interpreter is a web tool for linguists to use and interact with. Many computational algorithms are employed; such as those employed in machine learning. There are two levels of approaches for constructionists: the processing level, which uses transient structures to represent information about what is being parsed or produced, and the design level, encompasses the complexity of writing grammars by maintaining methods and techniques.

The processing level looks at a sentence as having two poles (Steels, 2013, p. 155): a semantic and a syntactic. The semantic pole contains a transient tree structure that is read from left to right, perhaps capturing the schematicity of the linguistic unit. The Syntactic pole is also a transient tree structure that is read right to left, and is notionally equivalent to the phonological pole in cognitive grammar. Tree structures also represent the unit and

sub-unit hierarchies. A matching and merging process occurs during the parsing of a construction where the condition already exists in the corpus for a given pole, and the contribution is matched to and merged with the condition. The semantic and syntactic poles are completely dependent on the design of a construction grammarian. All this is very much like a usage-based exemplar (Bybee, 2010, p. 19) in that the construction is the same for both parsing and production, and variation is well-represented. The novel differences or changes in the construction are called footprints, and are tagged for historical reference. Sets and networks are used to determine contextual priority of execution. Chunks are also used for construction units that are highly entrenched, which provides a more effective triggering.

The design level is the construction grammarian's workshop. All constructions work more efficiently if there are constraints. The grammar designer starts with the higher-level abstractions. Over the years of grammar implementation design patterns have been uncovered, and now can be used as starting templates. Common sets of features are often bundled, and then matrixed to differentiate feature bundle competition. Several human steps must be accounted for in the grammar design. "The first step is to embed the production of comprehension of sentences in a complete semiotic cycle." (Steels, 2013, p. 165). This includes the internal world model of perception and action, the categorization of reality, i.e. meaning, application of constructions, and articulation. And then the reverse steps for the hearer. FCG emphasizes reversibility of constructions, it goes both ways.

Conceptual Spaces

"Il mondo ha la struttura del linguaggio e il linguaggio ha le forme della mente." (*The world has the structure of language and language has the form of the mind*) (Gärdenfors, 2014, p. 8). This is profound in that it says language is experiential, and we are not disconnected from the world we live in. Gärdenfors lays out the background of semantics, and provides a cognitive linguistic understanding of its importance. Two conceptual spaces are defined with three important themes: convexity, domains and dimensionality. Convexity supports the learnability of categories and effectiveness of communication (Gärdenfors, 2014, p. 26). Concepts are learned, we are not born with them. It adheres to the semantic hypothesis that the typical meaning is the prototype at the center of a convex region assigned to the linguistic unit. Domain is broadly interpreted as indicating any kind of conception of realm of experience (Langacker, 2008, p. 31). Domains are gradient from basic to abstract and locational to configurational. Dimensionality is important in describing the space in which domains exist. Gärdenfors argues that all domains can be described dimensionally.

Gärdenfors explores semantic domains in depth. Basic domains, the ones that are more closely tied to sensorimotor processing, are learned early in child development. Thus, the value system of "language must be learnable to a child" (Gärdenfors, 2014, p. 54) arises[5]. Gärdenfors refers to it as the epistemological learning criterion. It is easier to explain to a child "chartreuse" and "mauve" than it is "inflation" and "mortgage" because the color domain is closer to the sensorimotor visual input. The main domain thesis is "a close

[5] I want to collect any linguistic value system I find, as it will be important to my thesis.

parallel exists between the development of intersubjectivity and the development of semantic domains. Intersubjectivity is the representation in the mind of the emotions of others, the desires of others, the attentions of others, the goals and intentions of others and beliefs and knowledge of others. These five components of intersubjectivity are crucial to language development (Gärdenfors, 2014, p. 57). Emotions being first as shown in Figure 4.

The emotion domain is the first to be learned by the infant, in the womb with sounds and movement, and by the touch of the parents before the eyes open. The development of semantic knowledge begins with emotion. This is not the only a semantic domain developing, but is the most salient communication occurring for the developing child. Metaphors are a blending of conceptual spaces. A mental frame is a conventional bundle of ideas (Coulson, 2001, p. 26). A mental space is an array of connected mental elements simultaneously activated by a person. A mental web is a set of mental spaces that are activated and connected as one is thinking about a topic. Vital relations are the most frequent and important connections. Blending mental spaces in a mental web yields a blend. Projections are the elements and relations that come into mental spaces and are blended. There is an emergent structure in the blend and in the mental web. There is a scale of human thought bundles from not tractable and manageable, where they are beyond the mental limits, to very tractable and manageable. Blends are tight compression of emergent ideas, but with less information than in the entire mental web.

Blending appears to be a human experience that no one is even partially aware of except rarely and it seems elusive to science to measure. This process of frame-shifting (Coulson, 2001, p. 34) is critical to creating new concepts and domains experientially.

Conceptual blending occurs at all schematic levels, and tends to group in identifiable patterns (Coulson, 2001, p. 123) of concepts and domains, like neuronal groupings (Edelman G. M., 2006, p. 55).

Cognitive Grammar

Cognitive Grammar is a framework (Langacker, 2008, p. 3) from which a comprehensive and coherent view of language structure manifests inclusive of all human experiences. In this framework, several key concepts are offered, the first and foremost of which is that grammar is meaningful and symbolic by nature (Langacker, 2008, pp. 3-5). Another important concept is that a linguistic unit is emergent due to its use in language. The usage event occurs repeatedly, and as life happens the linguistic unit is entrenched or conventionalized within a community of human language users. From the continuous use of these linguistic units or the first-time use, the individual processes and stores them with other units that are related. The symbolic structure of a unit has both a semantic and a phonological pole which are bound together by its use, i.e. meaning and form are bound by their relationship. Each pole has varying Schematicity. A new expression is specific. A frequently used expression is more abstract. An expression can become more schematic over time and with experience (Langacker, 2008, p. 21). “Google” for example, started as a name of the search engine, and is now used quite frequently for more than just the name of the trademark. All this is mapped as the conventional unit status shown in Figure 5 Conventional Unit Status.

Symbolic assemblies are manifest by their gradient schematicity, salience and elaboration, which correspond nicely to levels of entrenchment, contextual priming and overlap, and could be implemented in a neural network as three axes in a coordinate

system, i.e. a vector. The vector could be derived from the unit itself and existing corpus data (more on that later). The symbolic assembly, shown in Figure 6 Symbolic Assembly, can somewhat easily be written in a format that is amenable to storing data in a corpus using a construction grammar framework such a Fluid Construction Grammar[6]. That will not be discussed here, but it is important to note the complexity of the assembly as having five symbolic structures using three symbolic units.

It could represent, for example, the usage event of "Eat your soup", and the translated construction grammar could look like "[[EAT/eat V IMP][[PERSON REFERENCE / your 2P POSS][SOUP / soup N]]]". Notationally this construction is completely fabricated on my part, but is based in part by how Langacker has suggested that a construction might appear (Langacker, 2008, p. 161). It is important to note the disagreement with CG construction ubiquity, i.e. cross-linguistically there is much evidence of typological variance in constructions (Croft, 2001, p. 104). Also, Croft describes the typological variants using the concept temporal orderings, e.g. de-ranked hierarchy as an example of temporal ordering in constructions as related to subordinate clauses (Croft, Radical Construction Grammar Syntactic Theory in Typological Perspective, 2001, p. 360). This is however represented, and could allow for variance in the nested nature of cognitive grammar symbolic structures. The construction is important for our perceptive machine to analyze the input, and properly store data in the

[6] See https://www.fcg-net.org/tutorial/lectures/ for more information regarding Fluid Construction Grammar, an endeavor led by Luc Steels, and his book about FCG design patterns (Steels, Design Patterns in Fluid Construction, 2011)

corpus. With a clean form of construction, the bias can be derived by calculating the level of entrenchment, contextual priming or salience and the amount of overlap from each symbolic unit in the construction. The sum of unit values for each structure are then passed to the function that determines storage, e.g. in the “Eat your soup” example there would be five structures passed to the function.

Language Processing

If we look at the human body as a perceptive machine, again with the robot analogy, the dynamical system then needs to mimic the somatic model shown in Figure 1. Machine Learning (ML) is among the vast number of computer science academic pursuits currently being researched today. The broad goal of many of these researchers is to mimic, or simulate functions of the human brain in its cognitive ability to process incoming sensory data, and to store it such that it can be queried again and used in the continuous processing of new input. These researchers spend much of their time studying and devising computational algorithms to allow a computer to learn. This area of study has been called many things, among which is the neural network. It is more recently referred to as deep learning in the ML domain, but I will refer to it as neural network. This paper is concerned with a small part of the process of matching input with stored data, and focusses on the Cognitive Grammar (CG) means of selection, namely the level of entrenchment, contextual priming and amount of overlap between target and potential categorizing structure or schema.

Computational linguistics would benefit greatly from a cognitive grammar approach to the Deep Learning algorithm. I argue that a neurological value system using the Bias in the neural network forward-propagation algorithm could be used to guide the dynamic

system matching process. This bias**,** unlike the typical bias used by researchers in the NN algorithm, would be a vector representing the constraints of a multi-modal symbolic structure input as mentioned above. It could be used to guide the storage of tokens, their symbolic unit relationships and contextual or discourse information in both short-term and long-term corpora like what we call “memory”.

The equation shown in Figure 7 Forward-Propagation, which is computed by multiplying a weight (w) by a scalar input (p) and adding it to the bias (b) and this is summed for all inputs (R), is passed into a function (f) and yields the activation or output (α). The algorithm, shown in (Figure 7 Forward-Propagation), was developed in the 1950s, and has since been used in most machine learning and artificial intelligence endeavors that are constantly making improvements in speech processing and near instantaneous speech translation, for example (Müller & Guido, 2016, p. 364). We can think of the NN algorithm as representing a single neuron, and when we consider the brain, we can expand this to a vast network of neurons that relate to each other dynamically when input is received. Considered by many computer scientists as the black box calculation of artificial intelligence, for cognitive linguists with a computational slant it is ideal for computationally gathering and evaluating usage events as they are perceived, and storing the output in corpora. There is obviously a great deal to this algorithm, but in this paper, I want to focus on the Bias, which is often disregarded or trivialized in computer science. I believe it is essential to a dynamic system, and even more specifically to the matching process where symbolic units are stored following a usage event.

The perceptive machine, as I am calling it, is on the receiving end of information, i.e. sensory data. This fictional machine could, conceivably, be a person's smart phone, but that is not as important as what it could do. As sensory data is perceived it is first translated into a machine-readable format. The machine then can parse the token into its symbolic units. Each unit is valued and weighted. At this point the machine creates the bias vector from data already stored in its corpora. The bias is summed with input value, and then passed to the activation function that is responsible for storing the processed input, i.e. adding data to the corpora, short-term and long-term. It is therefore critical to examine the bias more closely considering neuro and cognitive sciences and cognitive grammar ideas.

Edelman describes the value system in the brain as a selection mechanism that releases neurotransmitters such as dopamine to govern behavior. These rewards are pleasurable in the sense that learning is facilitated allowing the selection of favorable activities within the network of neuronal groups of synapses. "Selection within these networks determines the categories of an individual animal's behavior; value systems provide the biases and rewards (Edelman G. M., second nature , 2006, p. 31)." The neuron, shown in (Figure 8 Neuron), provides a selection process that activates per the neurotransmitters received at the synapse. The activation then in turn has relationships with other neurons. "…neurons that fire together wire together…no two brains…are wired the same (Edelman G. M., 2006, p. 55)".

Neurotransmitters provide the bias or value system that determine the course of activation, and over time, the selection of behaviors. A continual flow of pleasurable molecules acts as a reward system enabling memories to exist. The value system is

essential to the development of a dynamic system. Temporal association of perception and action are foundational to changing skill and behavioral development. Developing systems exhibit emergent properties their interactions, and not dependent on preexisting codes (Thelen & Smith, 2000, p. 142). “So value systems may jump-start the building of oughts in a society, but do not directly determine them.” (Edelman G. M., second nature , 2006, p. 95) Whether we decide to view the biases as Constraints or Prods or Guides, there is an effect on the output that propagates forward to the next usage event that is perceived. The bias determines the storage of memories of usage events, which emerge as favorable or not per individual. In the NN the bias shifts the activation function output. For our purposes in cognitive grammar there is one bias per Symbolic Unit. The NN bias is either 1 or -1 in most machine learning algorithms, and it is determined by where the input value falls on a threshold curve; “Biases are weights associated with vectors that lead from a single node whose location is outside of the main network and whose activation is always 1 - Gallant (1993, pp.65-66), Bishop (1995a, p.78), and Reed and Marks (1999, pp.15-17)” referenced by (Hagan, Demuth, Beale, & De Jesus, 2016, pp. 2-2,2-8). In a dynamic system of matching usage events to long-term corpus, and short-term corpus storage, I propose a cognitive grammar bias vector (level of entrenchment, context priming or salience, amount of overlap) because these values can be determined from corpus data, a kind of bias feedback loop, or forward propagation.

Since the bias is typically a value of 1 for most researchers of machine learning, this approach is somewhat unorthodox, but not unheard of, e.g. it is used in supervised learning algorithms such as the k-nearest neighbor calculation (Müller & Guido, 2016,

pp. 30-46), which uses a Support Vector Machine algorithm[7]. It will however require a slower processing time to calculate the three values for our bias vector. In Figure 9 Symbolic Unit Bias Vector I show the three-dimensional conceptual space of the vector for ease of visualization. Each of the axes represents a gradient value between 0 and 1, which allows for maximal variation. The conceptual space is like a network of neurons in that each coordinate represents a location in memory. The visualization could of course be spherical or even better a blob where the distance from the center is boundless, but perhaps self organizes into the shape of a brain, (complete conjecture). The point however is that locations of vector endpoints can cluster together, and thus represent a "wire together, fire together" concept (Edelman G. M., 2006, p. 55). The bias vector then serves to guide the storage of the input, and the resulting activated vector becomes an address or index of the perceived symbolic unit. Also, other data can be queried easily from this storage configuration such as the ever popular "neighborhood density" value. It is important to discuss how we can derive each of the values that make the vector. Level of entrenchment is gradient from the specific use to the schematic or conventionalized use. Salience or contextual priming is gradient from the unit being completely novel to very salient. The amount of overlap is gradient from elaborative to baseline. The task then is to calculate each of these values such that a vector can be used in the matching process. In the perception process shown in Figure 10 Dynamic System

[7] A Support Vector Machine (SVM) is a discriminative classifier formally defined by a separating hyperplane. In other words, given labeled training data (supervised learning), the algorithm outputs an optimal hyperplane which categorizes new examples.
https://en.wikipedia.org/wiki/Support_vector_machine

Matching Process, it is the steps where the bias vector is determined, circled in green, which needs to be described.

Starting with the level of entrenchment, I would propose that we could obtain the frequency of use from the long-term corpus storage by querying all tokens containing the unit and calculating a distributed frequency value. Bybee noted, “the conservative behavior of high-frequency forms is related to the faster lexical access of high-frequency form: the more form is used, the more its representation is strengthened (Bybee, 2007, p. 271).” Therefore, by counting use of a symbolic unit among all the tokens a value of frequency can be computed to give us the probability of it occurring again.

The amount of overlap can be computed by looking at the nested level of the unit within the symbolic structure, querying for its usage count at that level in the long-term corpus, and computing the elaboration percentage of the baseline, i.e. the highest level of nested structure. This value will require some experimentation to determine the proper fit, maybe employing the Bayesian algorithm[8]. The gradient is flipped for this value because a baseline is more substantive than an elaboration (Langacker, 2016, p. 406). The baseline is therefore 1, and any elaboration would be calculated as a percentage of the baseline.

The context priming or salience is a tricky calculation because we are considering how meaningful a symbolic unit is within the context of the running discourse. Several problems present themselves, for one what is the size a discourse are we looking at, and

[8] In probability theory and statistics, Bayes' theorem (alternatively Bayes' law or Bayes' rule) describes the probability of an event, based on prior knowledge of conditions that might be related to the event. https://en.wikipedia.org/wiki/Bayes'_theorem

two does this cover multiple discourses, and if so, where do we query to obtain a value? I would assert that a short-term corpus be maintained along with the long-term. Like our short-term memory, it stores the current set of usage events, which could be queried to count usage, and determine relevance to the current topic. Obviously, this is not trivial either because aside from the same tokens being stored in long-term, it would require the storage of primary topic units that might be tagged as part of the construction, including the identification of the viewing frame channels which make up the symbolic unit poles (Langacker, 2008, p. 146). With this data stored in the short-term along with the frequency of use in the long-term, a gradient value could be computed.

In summary the multi-disciplinary research that has been looking at cognitive processing of language is contributing to a better understanding of how the CG selection criteria could be applied to a neural network model as the bias for processing language as input. More specifically it is my hope that my research will provide data that will allow me to compare the bias part of the neural network computation to Edelman's Value Systems and to the selection means of entrenchment, context and overlap in Cognitive Grammar. In the next section I describe the methods used to gather the data.

3 Building a Corpus

For a pilot study like this one I felt compelled to gather as much data as I could to be able to determine how the alignment of meaning is attained, or not, during the semiotic cycle of conversation. The three steps I performed in my research were the collecting discourse data from actual conversations between two or more people, coding the conversations with the cognitive grammar symbolic unit parts of phonology, which is the word spoken, and semantic, which is the domain or frame and the dimensionality (vectors) along with a time stamp for temporal ordering, and then analyzing the coded text to determine what causes the alignment.

3.1 Collecting YouTube Data and Building a Corpus

YouTube is a vast jungle of videos that include content like music, movies, instruction, copied media, interviews, discussions, etc. Some videos have subtitles, which are used to as a translation mechanism for various language preferences. Subtitles are usually embedded in the video, but sometimes is included as metadata and queued as time-marked text per language preferred by audience. Since the goal was to collect natural language data, subtitles don’t work. Closed-captions are also sometimes available, and this text is closer to what we want to collect. The video is usually marked with a “CC” hyperlink if they are available. When the “CC” link is activated the video then displays the Closed Caption text as words and sounds occur. This is not always exactly synchronous or entirely accurate, but for acquiring mass amounts of natural language discourse data for a corpus intended for semantic analysis it is adequate.

In order to download the closed-caption text I relied on a scripting language very popular among data scientists, Python, mainly because of the extensive useful code libraries

available. One such code library is called “youtube-dl”, which when employed can download pretty much anything associated with a YouTube video. YouTube videos are referenced by a “display id” that part of the YouTube URL like, https://www.youtube.com/watch?v=hpDHwfXbpfg. The last series of alpha-numeric characters following the “v=” is the display id. My video selection criteria were as follows:

- The speakers in the video should be speaking the English language, and I primarily preferred dialects of US based English.
- The video should contain two or more actively participating speakers such as interviews, discussions or conversations.
- The amount of spoken words should amount to at least 200, but I preferred the longer video discussions because my goal was to collect a million of them.
- The accuracy of the Closed Caption should be close to what is really said, some videos are wildly inaccurate, so I preferred the transcript use, which has the advantage of being punctuated as well as more accurate in most cases.
- I tried to collect a diverse range of topics, preferring to not include topical data that has been repeated more than 5 times in the corpus.
- It is important that some of the conversations contain a situation where initially there is a misunderstanding between speakers, and then both speakers come to a mutual understanding, and some should be a control set of conversations where no common understanding is achieved.

The extraction of closed caption transcripts from YouTube can be accomplished by using some Python code libraries, as I mentioned earlier. I wrote a script that I can

execute from the command line, named “youtube_corpus_maker.py” (8.1). To acquire the metadata of a YouTube video I simply pass the command parameter “download_youtube:hpDHwfXbpfg” with a colon followed by the display id of the video. This downloads the metadata to a folder on the hard drive, which by default is “corpora/youtube/hpDHwfXbpfg”. The metadata for the “hpDHwfXbpfg” display id is approximately 130 megabytes in size, which includes the video file (mp4 format, largest file), annotation files and most importantly the Closed Caption files. Since the discourse is in English I am interested in the file named “hpDHwfXbpfgmp4.en.vtt”. This file contains the video timed transcript (vtt), i.e. the text is marked with a time stamp indicating when the text is displayed during the video playback, shown in Figure 11. Next, I wrote some more code to pull the raw text out of the Closed Caption file, and write it all to a text file (8.1), which is the first of many corpus files that can be queried using a corpus reader. I will briefly describe the Natural Language Toolkit (NLTK) library here because everything the code does from this point on depends on it. NLTK (Bird, Klein, & Loper, 2009) is an extensive Python code library that can be used to analyze text.” We will take Natural Language Processing — or NLP for short — in a wide sense to cover any kind of computer manipulation of natural language. At one extreme, it could be as simple as counting word frequencies to compare different writing styles. At the other extreme, NLP involves "understanding" complete human utterances, at least to the extent of being able to give useful responses to them.” (REF p2). Part of this library includes a corpus reader, which I have extended for this project to be able to query all of the data from all of the video Closed Captions. The plain text file, “hpDHwfXbpfg_plain.txt”, is the foundation from which all other files used in analysis

are created, and therefore has a plain text corpus reader. The next file I produced with the script code is a Part of Speech Tagged corpus file “hpDHwfXbpfg.pos”. I extended the Tagged Corpus Reader code to query this file. With these two files added to the corpus it is possible to perform many important queries and operations for building the symbolic units, and eventually vectors for determining alignment of meaning during the discourse.

3.2 Building YouTube Symbolic Units and Vectors

I wrote more code to construct the discourse symbolic units (Langacker, 2008, pp. 16-17) in the form of an array of discourse segments that contain the following elements:

- The word – a phonological representation of the segment uttered by the speaker
- The Part of Speech – noun, verb, adjective, adverb, etc.
- The Time – the time at which the segment occurred in the discourse
- The Frame – a semantic pole representation of the segment
- The Vector – comprised of three dimensions entrenchment, context and overlap

These units, especially the ones that index the target understanding with value systems (dimensions in a vector) had to be constructed from additional files. The word and parts of speech tagged data are now available for use in creating them. It is important that a value system is quantifiable to be able to calculate the vector dimension. Dimensions are gradient and therefore can be decimal value between 0 and 1, e.g. 0.0 or 0.7899 or 1.0 would be acceptable values. This can also be done using various Natural Language Processing Python libraries[9]. From these dimensions, the alignment vector is formed,

[9] The Python libraries that I propose using are youtube-dl and nltk for extracting and coding. Other math libraries can be used for calculating dimensions for the vectors, such as multi-variant

which can be used to compare against similar usages in the discourse during analysis. First however I created some supplementary files such as an array of words and the times they occur “hpDHwfXbpfg_time_line.json” and a file containing discourse topical information “hpDHwfXbpfg_topics.json”.

The entrenchment dimension is simply a calculation of frequency distribution for a given word within the entire corpus, so the more words in the corpus the more accurate a calculation. NLTK provides a function for calculating the distribution values, so that is easy. To make a vector dimension scaled from 0.0 to 1.0 it is necessary to calculate the percentile across all word distributions. In other words, I took the highest word distribution and subtracted the lowest word distribution, and then divided the target word distribution by that value, i.e. Percentile = Target Distribution / (Maximum Distribution – Minimum Distribution). This becomes the entrenchment dimension.

The overlap dimension is calculated by looking at the target word in the context of synonyms and the percentage of difference between the synonym and its hypernym. The schematic distance from elaboration to baseline can be calculate from data that is also available for the English language called Word Net. It is a very large corpus synonym sets, as one might find in a thesaurus or at the bottom of a dictionary entry. It was necessary, therefore, to first create of file of word, synonym, hypernym and distance “hpDHwfXbpfg_word_nets.json”. NLTK provides a word net corpus reader that provides the ability to query synonym matches of a target word. By querying the word net corpus and selecting the synonym that fits best I could write an array to file. An element of the

---

logistic regression. Note: I will explain my methodology for extracting and coding thoroughly in final document.

array looks like what is shown in Figure 12, which identifies the target word, the part of speech tag, an array of possible synsets that best match the target along with the hypernym synset and the schematic distance or similarity. This target is in the context of

"Hey , welcome back to TED It's great to have you here Thanks for having me So , in the next half hour or so we're going to spend some time exploring your vision what an exciting future might look like , which I guess makes my first question a little ironic Why are you boring Yeah I ask myself that frequently".

The best fit could be any of the first three because they have the greatest similarity, elaboration compared to baseline. The overlap calculation is simply the best similarity. The calculation of the contextual priming or salience dimension requires another file to be created "hpDHwfXbpfg_frames.json". NLTK also provides a Frame Net corpus reader for the English language. With this corpus we can query[10] for lexical units in frame net that match the target word. My code then selected the lexical unit that matched the target word part of speech. The lexical unit has a related frame, which is written to the file along with the target word. I also store the distribution of frames used within the discourse. The salience calculation uses this distribution similarly to how entrenchment uses the corpus word distribution, i.e. Percentile = Target Distribution / (Maximum Distribution – Minimum Distribution). Once everything mentioned above has been calculated, the vector file "hpDHwfXbpfg_vectors.json" is written. This corpus file gives

[10] This query was very slow. It could take a couple of minutes for one word.

us everything we need to produce analysis information and visualizations. Each element in the array contains the symbolic unit with word, frame, time and a three-dimensional vector of entrenchment, overlap and context.

3.3 Producing Useful Information and Visualizations

I wrote some additional code to export a discourse vector file to a tab-delimited file that can be consumed by data analytic tools such as Microsoft Excel. The core objective is to be able to look at the words and corresponding frames during the discourse, and look at how the vectors change. This can be done by pivoting the data in Microsoft Excel, and even charted, but the size of the data can be daunting. I therefore provide for a filtering of words, frames or times as part of the export function. This allows me, after watching the video, and noting specific segments of interest, to export only those vectors. Targeting data in this way gives me something that is visible and easier to analyze. However, this doesn't give me enough to really visualize the alignment.

Another option would be to visualize the vectors in a cube, like (Figure 9 Symbolic Unit Bias Vector), and colorize the dots according to the time scale. I wrote some additional code that implements a few more Python libraries, "pygame (for the user interface), moviepy (allows playback of video) and pyopengl (for the three-dimensional graphing)". It is a challenge to bring vectors that cluster closely to a visualized state that is perceptibly meaningful so scaling is also necessary.

In summary, the effort is time-consuming to create visualizations, and may be more suited for further analysis in the order of a dissertation level of effort. Seeing that time is limited for writing this thesis, just acquiring the data is sufficient for the pilot study. The vectors show dimensionality in language use, and although only three dimensions were

calculated, there are many more, and multi-variant. The tools that I have created using Python are useful for extracting any text source, e.g. written text, transcripts from any source, including multi-lingual, and synthesized text sources. I chose YouTube as a source because it was readily available, and searchable for topics of interest. Of course, more metadata, such as the identity of the speaker, could be collected as well to enhance the corpus. I will continue to enhance the tool of course, but for now I will focus on the data collected from YouTube video closed caption text. In the next section I will show some more interesting examples of collected data, and review terminology used.

4 A Usage Event Data Model

The new corpus is comprised of over a million words from over a hundred hours of YouTube video closed caption text (Table 2 - YouTube Videos for Corpus). Each word is tagged with the part of speech11. The Natural Language Toolkit (NLTK) identifies the part of speech by an English lexicon that includes its grammatical usage and its tag that is part of the NLTK Tag Set (Table 3 - NLTK Part of Speech Tags). In addition to the corpus that is queryable through an extension of the NLTK Corpus Reader, there are data structures linking Frame Net and Word Net Lexical entries, as well as for the vector calculations. All this corpus data is easily available for analysis of dimensionality and value systems that prod the semantic alignment of understandings between interlocutors. To analyze the mutual understanding phenomenon, it is necessary to review the terminology of data analysis, and show how the corpus can be used to mine the analyzable data. Since we know that human language processing is adaptive, and the mutual understanding is attained through multiple exchanges of words and gestures, multiple inputs, then we can look at the many machine learning algorithms used by data scientists as possible ways of processing the corpus data.

4.1 The Three Stages of Corpus Data

There are three forms of data that are generally accepted among data scientists, and they are raw, transformed and information. Raw data is the data closest to the source, so in my corpus of YouTube closed caption text it is the data contained in the closed caption file as

[11] A list of these tags can be acquired by running a python script “nltk.help.upenn_tagset()”

shown in Figure 11 - Closed Caption File Sample. This data could be queried, but would take more time to process because the data is encoded for the specific purpose of displaying during a YouTube video at a specific time. Therefore, it is necessary to transform this data into a format that can be queried. As mentioned earlier it was necessary to transform the data into several forms, and utilize other corpora to produce the most useful query for my needs, i.e. the symbolic unit including vector. The stages of data transformation is shown in (Table 4 - Stages of Corpus Data) for one of the closed caption extracts from YouTube (video id: gZKDInabaPM, words: 2305, set: 590, duration: 00:11:05.878, title: I debate with Dietitian on LIVE TV this morning - My reaction - People Blogs, the first 44 words).

Before I describe the data produced in the transformative phase, I would like to quickly discuss discourse and how information is packaged. Aside from closed caption data being somewhat lacking for several reasons that I will cover in the conclusion, there is some value that can be attained, namely the “word” element, the smallest datum in this corpus. Each word is therefore important to the discourse as a point in the temporal flow of the recorded conversations. I would also assert that the word is the segmental content as part of the usage event viewing frame (Langacker, 2008, p. 146). The view frame is divided into sets of channels, vocalization and conceptualization, and the segmental content being the word as heard in the conversation. The transformation of data then builds upon the word in the conceptualization channels. The closed caption discourse is then comprised of a temporal flow of usage events containing transformed data where the phonological pole contains the word, and the semantic pole contains domains, time, part of speech and the dimensions of entrenchment, overlap and contextual priming (or salience). The

information channel could also be added with some more effort using parts of speech to identify which discourse mode (Smith, 2003) is used. The corpus, however, does not include currently discourse modes.

Following the concepts of the usage event and the bipolar segments and channels as they flow through time in a discourse, I have drawn a (Figure 13 - YouTube corpus Symbolic Unit). Since Cognitive Grammar considers any aspect of a usage event possibly emergent as a linguistic unit (Langacker, 2008, p. 146), then the stages corpus data that I have amassed are also relevant. A phonological pole is comprised of the plain text word and the part of speech. A semantic pole is comprised of the semantic domain, the part of speech, the temporal order of event and the dimensions (entrenchment, overlap and context). In the following parts of this section I describe the stage of data, what information can be derived from it, and how the corpus data is used in processing from the observer's point of view, i.e. the person watching the YouTube video.

4.2 Plain Text

Text is the fundamental structure of closed captions. It is a stream of words marked with time stamps indicating when to display them as the video progresses. As stated earlier in this paper, closed caption text can be with or without punctuation. It can be transcribed accurately, or not so accurately with missing utterances. The YouTube corpus has plain text files that contain text only with or without punctuation. This is the basis for building everything else. Words and punctuation are delimited by a space, and contracted words like "it's" are split and represented as "it" and "'s". With the plain text corpus reader, it is then possible to get an accurate list of words. The Natural Language Toolkit functions available to derive important linguistic information such as frequency distribution,

concordances, n-grams and more, all of which are usable with the plain text part of the corpus. Describe how plain text can be queried for frequency and concordances.

There are many useful queries to be performed on a text only corpus, and can be accomplished using the Python Interpreter. For example, if I wanted to know the dispersion of several related words in a discourse, I can simply load the discourse as text and call a function that creates dispersion plot, using the following Python code:

```
import nltk
from nltk import text
from corpus import youtube
wrds = youtube.words(['y8hy8NxZvFY/y8hy8NxZvFY.pos'])
txt = text.Text(wrds)
txt.dispersion_plot(['darwin', 'darwinian', 'god', 'christian', 'belief',
'church', 'creation', 'biology', 'biological', 'evolution', 'science', 'religion',
'soul', 'language'])
```

The dispersion plot, shown in Figure 14 - Dispersion Plot of discourse Creationism vs. Evolution, has words that are important to both parties discussing “”, and shows an even exchange ending with “belief”. Frequency Distribution is another function available with NLTK, so the following shows the frequency of derivations of “belief” from the same discourse.

```
import nltk.probability
fdist = nltk.FreqDist(txt)
fdist['belief']
Out[12]: 8
fdist['believe']
Out[13]: 46
fdist['believes']
Out[14]: 1
fdist['believer']
Out[15]: 0
fdist['believed']
Out[16]: 2
fdist['believing']
Out[17]: 6
fdist['believers']
Out[18]: 0
```

The concordance can be generated as well, so the following shows where “belief” is used.

```
txt.concordance('belief')
Displaying 8 of 8 matches:
he opium of the people today is the belief that they wo n't be judged by God w
```

- question and that is Does religious belief make the world a better place does
- world a better place does religious belief make the world a better place ? We
- say on the question does religious belief make the world a better place over
- led may be Alleviated somewhat by a belief in God Psychosomatic medicine is we
- ple were killed for their Christian belief in the last century than any other
- nty-six percent saying no religious belief does not make the world where this
- does not make the world where this belief does not make the world a better pl

Another interesting function is collocation, which shows bigrams that are used often in the discourse.

- txt.collocations()
- Richard Dawkins; George pell; 've got; old testament; better place;
- natural selection; non theist; wafer turns; religious belief; homo
- Sapiens; hope nobody; original sin; lawrence krauss; random selection;
- n't believe; give rise; Catholic church; Darwinian natural; creative
- intelligence; silly question

There are also many other functions available to produce additional data and create an informational presentation. For this paper, it was necessary to use some of these functions to build a corpus viable for semantic analysis, i.e. frequency distribution. The stream of plain text is tokenized so that each individual word is packaged as phonological segment content in each usage event, and is viewed or heard by the observer as they occur in the video, or at least close to the visual utterance or articulation segment. It is also important to note, concerning timing, words in conversations are not articulated always “one after another”. Interlocutors will often say words at the same time, creating overlapping usage events, but closed captioning is delivered as chunks, often after the articulation.

4.3 Tagged and Categorized Text

The part-of-speech corpus files have the extension of “.pos”, and each token is represented with the word followed by a part-of-speech tag (Table 3 - NLTK Part of Speech Tags), and delimited with a “/”, for example:

“okay/NN let/NN 's/POS start/VB off/RP with/IN what/WP is/VBZ a/DT computer/NN what/WP is/VBZ it/PRP computer/NN it/PRP 's/VBZ really/RB simple/NN it/PRP 's/VBZ just/RB a/DT simple/JJ machine/NN but/CC it/PRP 's/VBZ a/DT new/JJ type/NN of/IN machine/NN the/DT gears/NNS the/DT Pistons/NNS have/VBP been/VBN replaced/VBN with/IN electrons/NNS”

From this corpus file, we can also get the words, perform the same NLTK functions as with plain text, so in creating other files I have used the part of speech tagged file with the corpus reader. The following is frequency distribution of the part-of-speech tags in the YouTube corpus.

```
import nltk
from nltk import text
from corpus import youtube
import nltk.probability
yt_tagged_words = youtube.tagged_words()
tag_fd = nltk.FreqDist(tag for (word, tag) in yt_tagged_words)
tag_fd.most_common()
Out[8]:
[('NN', 147886), ('IN', 126074), ('PRP', 118907), ('DT', 102274), ('NNP', 82888), ('RB', 79435),
 ('VBP', 62287), ('JJ', 59751), ('VB', 52203), ('CC', 51990), ('NNS', 45488), ('VBZ', 42688),
 ('VBD', 29890), ('TO', 29233), ('VBG', 23552), ('MD', 16486), ('VBN', 15911), ('PRP$', 13747),
 ('.', 10562), ('CD', 10403), ('WP', 10048), ('WRB', 9344), ('WDT', 7822), (',', 6663), (':', 5388),
 ('EX', 5257), ('RP', 4963), ('POS', 2981), ('JJR', 2906), ('RBR', 1854), ('JJS', 1703), ('PDT', 1103),
 ('UH', 757), ('NNPS', 630), ('RBS', 563), ("''", 134), ('``', 117), ('FW', 90), ('$', 73), ('WP$', 56),
 ('(', 53), (')', 53), ('SYM', 6), ('#', 4), ('LS', 2)]
```

Of all the part-of-speech tags, nouns (NN) win the “most occurrences” award. These tags are very useful for parsing grammar according to a pre-defined set of syntax rules, and NLTK provides functions to help with that, but I used them mainly to filter results from querying Frame-Net and Word-Net corpora, which I will cover in the next part. I would also assert that the part-of-speech tag is packaged as part of both the phonological and semantic poles in the usage event. In the semantic pole, the classification of a word is at play, schematically bound to the English language, in the YouTube corpus, as participants, modifiers and events. The entrenched patterns or syntax rules belong in the phonological pole as a larger usage event such as noun phrase and verb phrase. For example, “always”, a temporal adverb precedes verbs mostly as shown below.

```
bi_tags = [b[1] for (a, b) in nltk.bigrams(yt_tagged_words) if a[0] == 'always']
bi_fd = nltk.FreqDist(bi_tags)
bi_fd.tabulate()
```

VB	VBN	VBP	VBG	VBD	IN	DT	JJ	NN	RB	VBZ	PRP	NNS	PRP$	RBR	WRB
100	94	91	61	57	48	47	40	40	40	32	26	13	5	4	4

The pattern of the adverb “always” followed by a verb is recognizable by an observer as a larger usage event. In the computational area of study of distributional semantics, context occurrence prediction relies heavily on word meanings by their contextual representations (Fried, Polajnar, & Clark, 2015, p. 1). The part-of-speech tagging allows such analysis. Another categorization that is part of the collected YouTube data is the discourse categories and tags[12]. A comparison, for example, of frequencies by category across discourses can generated as shown in Table 1 - Modal Verbs Frequency by Category.

```
import youtube_corpus_tool
words = ['would', 'could', 'should', 'will', 'can', 'shall', 'must']
output = 'corpora'
export = 'tabular_modal_frequencies_by_category.txt'
tfdbc = youtube_corpus_tool.get_tabular_frequency_distribution_by_category(words, output, export)
```

From this simple table, a heat map could be generated to show the hot spots modality used in media. Categories are packaged as part of the information structure channel, and can be used to correlate words in as much as they are relevant to the discourse topic. For example, in terms of event modality, the category of “Education” as shows a high frequency for dynamic abilitive modal use, i.e. the word “can”. A usage event, therefore, may have high semantic relevance to the topic of “Education” when part of a dynamic abilitive modal event (Palmer, 2001, pp. 76-77). Much more can be done with tags and categories, of course, but the main importance is the foundational data is generated, and upon this data, semantic data can be generated, which will be described in the remaining parts of this section.

[12] Tags here refer to the annotation words, mostly names of popular people and subjects, attributed to a given discourse, similar to “Hash Tags”.

4.4 Words and Frames

FrameNet is a corpus developed on the theory of Sematic Frames created by Charles Fillmore and colleagues. Fillmore's Semantic Frames model derives its data from lexical sets, which in the case of the FrameNet corpus can be queried in many ways. The YouTube corpus contains a file for each discourse with frames identified for some of the usage events. The files are named with the YouTube video identifier, and suffixed with "_frames.json"[13]. The file is composed of two parts, the frames that are identified as framing the lexical unit or word and the frequencies of use for each frame with in the discourse.

- `{"frames": [["okay", ""], ["evening", ""], ["everybody", ""], ["and", ""], ["welcome", ""], ["my", ""], ["name", "Being_named"], ["is", ""], ["slim", "Body_description_holistic"], ["Charles", ""]…`
- `"counts": {"": 3790, "Being_named": 18, "Body_description_holistic": 2, "Compliance": 127, "Statement": 21…`

The symbolic unit's semantic content packs the frame or domain (Langacker and Gärdenfors), and is quantified to calculate the salience dimension. Frames can form an intonation group of one to several words (Langacker, 2008, p. 154) forming a phonological cohesion, but this corpus data is derived from a single lexical unit. In other words, each word in a discourse is singly queried against the FrameNet Corpus using the following code.

```
def get_word_frames_from_tagged_words(tagged_words):
    stopwords = nltk.corpus.stopwords.words('english')
    word_frames = []
    frame_counts = {}
    for tagged_word in tagged_words:
        if len(tagged_word) > 1:
            word = tagged_word[0]
            pos = get_lexical_unit_pos(tagged_word[1])
            frame = ''
            if word not in stopwords and word.isalpha() and len(pos) > 0:
```

[13] The "json" extension indicates that the data is formatted according to the JavaScript Object Notation standard.

```
                lexical_units = fn.lus(r'(?i)' + word)
                frames = [lu.frame.name for lu in lexical_units if
lu.name.endswith(pos)]
                if frames and len(frames) > 0:
                    frame = frames[0]
            word_frames.append((word, frame))
            if frame not in frame_counts.keys():
                frame_counts[frame] = 1
            else:
                frame_counts[frame] += 1
    return word_frames, frame_counts
```

When FrameNet is queried it returns a list of lexical units that match the word, from which a list of frames filtered by part-of-speech is extracted. Although sometimes more than one frame is returned, I am only relating the first one in the list to the word. This could be changed to allow more than one, but then the salience dimension would be multi-variant, and require much more rigor to calculate. Note that I excluded stop words from my query. I did this to reduce the time it takes to process a discourse. I am aware that by excluding these words I am losing salience, and I may put them back in. Stop words can be acquired with the following code.

```
- import nltk
- from nltk.corpus import stopwords
- stopwords = nltk.corpus.stopwords.words('english')
- print(stopwords)
- ['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', 'your',
  'yours', 'yourself', 'yourselves', 'he', 'him', 'his', 'himself', 'she', 'her',
  'hers', 'herself', 'it', 'its', 'itself', 'they', 'them', 'their', 'theirs',
  'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', 'these', 'those',
  'am', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had',
  'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if',
  'or', 'because', 'as', 'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about',
  'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above',
  'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under',
  'again', 'further', 'then', 'once', 'here', 'there', 'when', 'where', 'why',
  'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some',
  'such', 'no', 'nor', 'not', 'only', 'own', 'same', 'so', 'than', 'too', 'very',
  's', 't', 'can', 'will', 'just', 'don', 'should', 'now', 'd', 'll', 'm', 'o',
  're', 've', 'y', 'ain', 'aren', 'couldn', 'didn', 'doesn', 'hadn', 'hasn',
  'haven', 'isn', 'ma', 'mightn', 'mustn', 'needn', 'shan', 'shouldn', 'wasn',
  'weren', 'won', 'wouldn']
```

These words have a high frequency of use, and do have frames, but are also significant to the flow of the discourse. So, they are packaged as segment and semantic content in a usage event. The frames profile a lexical unit as it is used (Croft & Cruse, 2004, p. 14) in a coherent region of conceptual space. This is also called a domain, and this is why I generated a master file of domains present in the YouTube corpus, named

"_domains.json". This file contains the frame, and all vectors associated with it in the corpus. I will describe vectors more on page 42. The frames file also counts usage of the frame in each discourse. The baseline and elaboration dimension or overlap could also be derived from frames, because there is a hierarchy at work. This proved to be less information, however, and so I used the WordNet corpus, which I describe in the next part of this section. There are too few frames in the FrameNet corpus that profile lexical units, i.e. some words do not have frames.

4.5 Words and Hypernyms

There is another corpus that provides senses of words as synonyms. WordNet (Miller, 1995) is a database that holds semantic relations of synonyms, called "synsets" that are linked hierarchically. The following shows the synsets for "belief".

```
import nltk
from nltk.corpus import wordnet
synonyms = wordnet.synsets('belief')
print(synonyms)
[Synset('belief.n.01'), Synset('impression.n.01')]
```

Each synset is a schematic node in a hierarchy of concepts, where higher level nodes are more abstract, and lower level nodes are more specific. The schema can be traversed by querying hypernyms and hyponyms. The next level below in the schema for "belief", for example, is shown below.

```
synonyms[0].hyponyms()
Out[7]:
[Synset('autotelism.n.01'),
 Synset('conviction.n.01'),
 Synset('doctrine.n.01'),
 Synset('expectation.n.01'),
 Synset('faith.n.02'),
 Synset('fetishism.n.01'),
 Synset('geneticism.n.01'),
 Synset('individualism.n.02'),
 Synset('meliorism.n.01'),
 Synset('opinion.n.01'),
 Synset('originalism.n.01'),
 Synset('pacifism.n.02'),
```

```
    Synset('philosophy.n.03'),
    Synset('public_opinion.n.01'),
    Synset('religion.n.01'),
    Synset('revolutionism.n.01'),
    Synset('sacerdotalism.n.01'),
    Synset('spiritual_being.n.01'),
    Synset('spiritual_world.n.01'),
    Synset('spiritualism.n.02'),
    Synset('suffragism.n.01'),
    Synset('supernaturalism.n.01'),
    Synset('superstition.n.01'),
    Synset('supremacism.n.01'),
    Synset('theory.n.03'),
    Synset('theosophism.n.01'),
    Synset('thought.n.03'),
    Synset('totemism.n.01'),
    Synset('tribalism.n.02'),
    Synset('values.n.01'),
    Synset('vampirism.n.01')]
```

The hypernym is the next level node above, and for “belief” looks like the following.

```
synonyms[0].hypernyms()
Out[8]: [Synset('content.n.05')]
synonyms[0].hypernyms()[0].hypernyms()
Out[11]: [Synset('cognition.n.01')]
synonyms[0].hypernyms()[0].hypernyms()[0].hypernyms()
Out[12]: [Synset('psychological_feature.n.01')]
synonyms[0].hypernyms()[0].hypernyms()[0].hypernyms()[0].hypernyms()
Out[13]: [Synset('abstraction.n.06')]
synonyms[0].hypernyms()[0].hypernyms()[0].hypernyms()[0].hypernyms()[0].hypernyms(
)
Out[14]: [Synset('entity.n.01')]
```

The highest abstraction is “entity”, but the next level up from “belief”, “content” is enough to calculate the baseline/elaboration dimension using the distance from the hypernym.

```
belief = wordnet.synsets('belief')
abstract = belief[0].hypernyms()[0]
print(abstract)
Synset('content.n.05')
print(belief)
[Synset('belief.n.01'), Synset('impression.n.01')]
distance = 1 - belief[0].wup_similarity(abstract)
print(distance)
0.0909090909090909
```

The distance is calculated by subtracting the similarity of “belief” to “content” from 1. This is included in the YouTube corpus in another file suffixed with “_word_nets.json”. For each word in a discourse, an entry contains the word, the part-of-speech, the synsets and the synset distances, as shown below.

- ["believe", "VB", [["Synset('believe.v.01')", "[Synset('accept.v.01')]", 0.8571428571428571], ["Synset('think.v.01')", "[Synset('evaluate.v.02')]", 0.8], ["Synset('believe.v.03')", "[Synset('expect.v.01')]", 0.8571428571428571], ["Synset('believe.v.04')", "[Synset('believe.v.01')]", 0.8888888888888888], ["Synset('believe.v.05')", "[Synset('credit.v.04')]", 0.9230769230769231]]]

Each of the synonyms are senses of the verb “believe”, and could be substituted in a usage event with slightly different amounts of schematic elaboration. In the conceptualization channels (Langacker, 2008, p. 146) of the semantic pole the schema packs information that relates to and coordinates with the usage events before and after, but also can be a reference to other parts of the discourse playing a pragmatic role. With the data queried from WordNet, the YouTube corpus becomes stronger, but it still needs to be pulled all together into a complete symbolic unit. In the next part of this section I describe the vector, a value system, which contains three dimensions of entrenchment, overlap and salience.

4.6 Vectors and Dimensionality and Domains

The previous corpus files were created with the vector in mind. With these files, we can create another file that comprises for each word: the word, the part-of-speech, the time of close caption appearance in the video, the domain, and the vector of entrenchment, overlap and salience/context priming as shown in (Figure 9 Symbolic Unit Bias Vector). For each discourse, the file is the video display identifier suffixed with “_vectors.json”, and each entry is stored as follows.

- {"word": "believe", "pos": "VB", "domain": "Taking_sides", "time": ["believe", "00:02:17.195"], "vector": ["0.0130081300813", "0.0769230769231", "0.0270700636943"]}

The symbolic unit as shown in (Figure 13 - YouTube corpus Symbolic Unit) is complete with the addition of the topical categories and tags that can be queried for each discourse. I can export the vectors to a tabular file that can be imported into Microsoft Excel, and then create pivot table and charts, etc. There are some issues with this structure, however,

that need to be addressed, such as; not all words have domains, and this limits a contiguous representation of conceptual spaces. In other words, there are holes in the data. This is a limitation of FrameNet, or ignorance of the writer of this paper in querying FrameNet. There is always a part-of-speech present however, and this is a conceptual space as well. Another way to perform the contiguous analysis is to remove the vectors with blank domains. Part-of-speech is important to the usage event regarding dimensionality because we can derive the semantic roles and object categories (Gärdenfors, 2014, p. 116).

Taking vectors a little further, I also created a corpus file, suffixed with "_knn.json" that stores the vector, the K-Nearest Neighbor vector and the distance between them where an entry looks as follows.

- {"current": {"word": "biography", "pos": "NN", "domain": "Text", "time": ["biography", "00:02:46.008"], "vector": ["0.00162601626016", "0.0666666666667", "0.0127388535032"]}, "neighbor": {"word": "Island", "pos": "NNP", "domain": "Natural_features", "time": ["Island", "01:04:03.515"], "vector": ["0.00162601626016", "0.0666666666667", "0.0143312101911"]}, "distance": 0.0015923566878999987}

I list all words in (Table 5 - K-Nearest Neighbor for Discourse 6NOSD0XK0r8) where the distance is greater than zero. The above entry shows that within the corpus and within the discourse, the word "biography" has a vector that is closest to the vector of the word "island", and relates the domain of "Text" with the domain of "Natural_features", and they are temporally distant in the discourse.

In summary, the stages of development of the YouTube corpus leads to the vectors. It is the vectors that are important. My thesis is asserting that the value systems, quantified as a vector, of the interlocutors are at play, prodding what words get activated next within the conceptual domains. NLTK and other Python libraries provide the means for visualizing the vector data.

4.7 A Value System Application

An idea for a practical application of a corpus like the YouTube corpus is a “Chatbot”, like popular smart phone voice interaction applications. The YouTube corpus serves as one of the long-term memory storage areas or corpus, and the current chat conversation is stored locally in computer short-term memory, but also stored in a Chatbot corpus, which is like the YouTube corpus structure shown in (Figure 13 - YouTube corpus Symbolic Unit), so that the same queries could be performed. The Chatbot processes incoming text, and generates a response based on the usage event data of each word, chunk and sentence. The vectors are summed and compared to the vectors in the Chatbot corpus using K-nearest neighbor to determine the best-known response, i.e. the responses are activated from the corpus of input-response data, but the usage event vectors are calculated from the entire corpora, a value system at work. This, however, is of course not as “fait accompli” as described because the values stored and queried would always be the same. Although the K-Nearest Neighbor calculation does provide a probabilistic result, it still lacks the intuitive connection with another human being. The transformation from a canned response to a genuine “aware” exchange of meaning just isn’t there. There is obviously still much more to be done to create an “intelligent” Chatbot.

The corpora design is therefore very important to artificial intelligence research, and the conceptual space ontological model (Gärdenfors, 2014, p. 262) plays and important part in getting closer to a meaningful conversation with a machine. The Chatbot corpus can include sentence structure and chunking, and can be categorized by speaker and perceived topic categories. The YouTube closed-caption corpus is inadequate because of its lack of structure, and vectors are calculated at the word level only, providing a very

restricted value system. There are also holes in the semantic data that could easily lead to misunderstanding, or even complete impasse. Thankfully there are many other sources of data, e.g. transcripts of live chat sessions, for building corpora. The long-term and short-term memory corpora data model (Figure 15 - Memory Corpora Data Model) has potential, but still has flaws. The relationships of entities are described as follows:

1. A **corpora** (database) has one or more **corpus** entities
2. A **corpus** entity has one or more **file** entities
3. A **file** entity can be associated with many **topic** entities
4. A **topic** entity can be associated with many **file** entities
5. A **file** entity has one or more **token** entities
6. A **token** entity can be associated with many **classifier** entities
7. A **classifier** entity can be associated with many **token** entities
8. A **token** entity is associated with one or more **sentence** entities
9. A **sentence** entity has one or more **word** entities
10. A **word** entity is identified as a **POS** (part-of-speech) entity
11. A **word** entity can be associated with many **chunk** entities
12. A **word** entity can be associated with many **semantic value** entities
13. A **semantic value** entity can be associated with many **word** entities
14. A **chunk** entity can be associated with many **word** entities
15. A **chunk** entity is identified as a **POS** entity
16. A **chunk** entity can be associated with many **semantic value** entities
17. A **semantic value** entity can be associated with many **chunk** entities

This data abstraction allows each usage event to be stored in a way that can be queried using SQL (Structured Query Language)14, and resulting a matrix set of desired attributes from any of the joined entities. The memory corpora data model is one of many possible data storage and retrieval paradigms, but with this relation database model the value system captured as semantic values calculated from extant data at the time of the usage event. Semantic values, or dimensions, in turn make up the vector associated to the

[14] A structured query language is a procedural programming syntax used in obtaining a relational data set programmatically that joins entities according to their relationships and allows filtering, grouping and ordering of the results (https://en.wikipedia.org/wiki/SQL).

word or chunk. This gives us the perception part of the process, but not the production part. With the activation of data as a usage event takes place, it is then necessary for the Chatbot to construct a response, which means a response data model is required. "CONSTRUCTIONS, NOT CATEGORIES AND RELATIONS, ARE THE BASIC, PRIMITIVE UNITS OF SYNTACTIC REPRESENTATION. The categories and relations found in constructions are derivative…" (Croft, 2001, p. 46). The questions become "How and when does the Chatbot learn constructions?", and "How does the activated data relate to possible or probable constructions?".

I suggest a cursory construction data model, shown in (Figure 16 – Exemplar Construction Data Model), that is not related to the memory data model, but is maintained by an independent parallel process that learns the acceptable constructions from the memory data model, and when a response is needed the response is constructed by passing the activation data to a process that queries the construction data model for the most probable construction, and then assembles a response. This model follows the usage-based exemplar construction model (Hoffmann & Trousdale, 2013, pp. 60-63) as Bybee proposes in her chapter "USAGE-BASED THEORY AND EXEMPLAR REPRESENTATIONS OF CONSTRUCTIONS". The relationships of entities are described as follows:

1. A **corpora** (database) has one or more **exemplar** entities
2. An **exemplar** entity can be associated with many **slot** entities
3. An **exemplar slot** association can be associated with many **lexical unit** entities
4. A **slot** entity is identified as a **POS** (part-of-speech) entity
5. A **lexical unit** is identified as a **POS** (part-of-speech) entity
6. A **vector** can be associated with many **classifier** entities
7. A **lexical unit** entity can be associated with many **vector** entities
8. A **vector** entity has one or more **semantic value** entities
9. A **classifier** entity can be associated with many **vector** entities
10. A **vector** entity can be associated with many **lexical unit** entities

Chatbot now can pick the best response by querying the trained Exemplar Construction Data Model using the activation data from the Memory Corpus Data Model. The data flow, shown in (Figure 17 - Chatbot Data Flow), has two independent processes acting on interdependent data. The chat process is started with an initial request for information about the person who is chatting. The collected data is stored and updated in the classifiers entity. Classifiers are important to machine learning because they provide a feature-set that can be used in the linear regression calculation, used by many of the popular machine learning algorithms. Classifiers such as “age” and “gender” are most popular in recent research. It is important to note that the “name” classifier is associated with the chat text tokens, but the other classifiers of “age”, “gender”, etc. are associated with the answer tokens initially gathered in the chat session. The chat process proceeds with a back and forth exchange between person and Chatbot until the person indicates they want to quit. The input text from the person is transformed into data of which a series of token usage events are comprised, and then stored in Figure 15 - Memory Corpora Data Model. This completes the input phase.

The activation phase is where Chatbot queries the memory for a most probable set of activation data for the provided token. Memory activation occurs because of the set-based query that joins and filters data in the entire corpora to a smaller dataset. The activation data is most important for the last phase of generating a response. The value system determines the activation by looking at close vector matches of each lexical unit in a token against the entire corpora. The bias is calculated in the previous phase with current data, is made up of multiple and various dimensions, dependent on the state of the person corpus and the entire corpora of data. The vectors are summed for chunks and sentences.

The classifier feature-set could be used to filter the resulting data, but probably isn't necessary.

The response phase is dependent on Figure 16 – Exemplar Construction Data Model, which is maintained and updated with an independent process that builds construction exemplars from the corpora. The process is started, and repeats after a configured waiting period. Memory is queried for any changes, and then runs adds new exemplars or updates existing ones. The construction probabilities are then re-calculated for feature-sets and exemplars. By doing this, the response phase of the chat process can identify the best construction and corresponding lexical units to use as the response. The response is displayed after it has been stored in memory, and the process cycles again.

In summary, although the YouTube corpus lacks the classifiers necessary for Chatbot to act with a modicum of simulated intelligence, the overall effort of building the data by extracting, transforming and loading into files has set the stage for an application like Chatbot. The use of a database would provide a perhaps better container for the corpora, allowing the set-based query to be used. The usage event is abstracted in the database in such a way that words, chunks, sentences and tokens are bound to a corpus of files or discourses. The database allows any source of data to be added as well, with some extra programming, which allows the memory and exemplars to be amended, creating a smarter Chatbot. There improvements also that can be made, such as allowing for person or Chatbot to input or display multiple tokens at a time, as usually occurs in chatting between two or more people, instead of the back and forth conversation. Also, the possibility of allowing more than one person to chat with Chatbot could be developed

using the same data models. It is my intent to continue my research in this direction, and implement this application.

5 Conclusions

I want to say that I have gained a linguistically valid understanding of a value system in terms of its influence on coming to, or not coming to, a mutual understanding when two or more people engage in conversation. Is “value system” the correct term? If there is a bias, with multi-variant dimensions (value systems), is it used by a person when the activation occurs, and is the vector really pointing to a location in conceptual space? This pilot study, I think, shed light on these questions. The corpora of today are more robust than ever, and with semantic analysis tools like FrameNet (Ruppenhofer, et al., 2016) and WordNet (Miller, 1995) it will be easier to query for symbolic unit categorization. The end goal, although not conclusive, provided a clearer understanding of a cognitively sufficient “robot” model that considered many forms of input, i.e. a multi-modal, many sensorimotor mechanisms, many use-specific processing units with cooperation abilities, value systems that act as biases for language processing, and a storage medium where processing units can be activated by the perceptive machine. Again, I should stress that this was a pilot study that explored some aspects of how semantic alignment occurs or doesn’t occur in discourse. The dimensionality of vectors could be much more complex if applied to real world artificial intelligence and robotics.

A major success of this pilot study was that I built a corpus from internet resources (YouTube Closed-Captions) using NLP tools like NLTK and Python. The evolution of the corpus structure from plain text to vectors and more demonstrated the potential for building corpora that would support applications like Chatbot (4.7). There are many Python libraries available now, and more being built, that provide tools for data collection from the internet and document sources. Also, as shown with Chatbot, the can be stored

in a database, or a combination of flat files and database, and flat files can be in multiple formats, such as JSON (Java Script Object Notation) or CSV (Comma Separated Values) or XML (eXtensible Markup Language). Extending the NLTK library for querying custom values allows the custom corpus creation. A usage-based corpus is practical, and could be used to identify semantic alignment. The idea of dimensionality pointing to a point in conceptual space is sound because of two things, one dimensions exist, and two semantic change occurs. Finding the evidence will require more research and corpus building.

The YouTube corpus that I created was not conclusive, however, because of the missing classifiers i.e. if the closed-caption files were to contain who said what, then a classifier of who was speaking could be attributed to the spoken token. NLTK lacks access to a complete FrameNet, which also leaves holes in the data. WordNet does have a more complete corpus for the English language and could be used for obtaining semantic domains, but that was not the path I chose. Visualization of the data is lacking because most of the discourses were long, and who said what was indistinguishable. YouTube has a useful source of linguistic data, but not the closed-captions. Perhaps using the comments regarding a video that were contributed by viewers would yield better results. If I were to start all over I would create the Chatbot (4.7) application, and agnostically approach corpus creation by finding sources of data with identifiable classifiers, and I would load them into the Chatbot database using the same processes (Figure 17 - Chatbot Data Flow). The most important piece of data that was lacking in this study, but should be required, is the classifier both at the chat session level and at the token level. The Chatbot data model that I proposed earlier provides a relational storage location for classification.

Classifiers are essentially contextual reference points that point to semantic domain in conceptual space, they are schematically more abstract, and therefore provide target for the dimensional values that bias activation.

The linguistic value system of an individual allows them to participate in the coordination of meaning, which is achieved through common lexical knowledge shared between the individual and others who are engaged in conversation. Their value system prods the activation of memories in language processing, and maybe alignment or un-alignment of meaning occurs; linguistic categorization (memory, learning and performance) is biased dynamically by a set of one or more value systems (dimensions). The value systems hold the linguistic criteria for language processing, activation and selection of the best adaptive meaning. With the activated data the individual can construct a response from construction patterns in memory because data is retained for all linguistic experiences as exemplars of usage theoretically. Although the pilot study doesn't show evidence of this, it does suggest further research in corpora development.

In summary, there were a few things that were evident in this study. Closed-caption text is not the best source of data due to the lack of classification features. The data does however support the idea that language is shared, and when two or more people are having a conversation there are missing parts which are pointed to with the shared context, shared human experiences. The under-determinacy is evident because mutual understanding does occur, and not just with the participants in the conversation, but with the viewer of the YouTube video also.

6 Figures

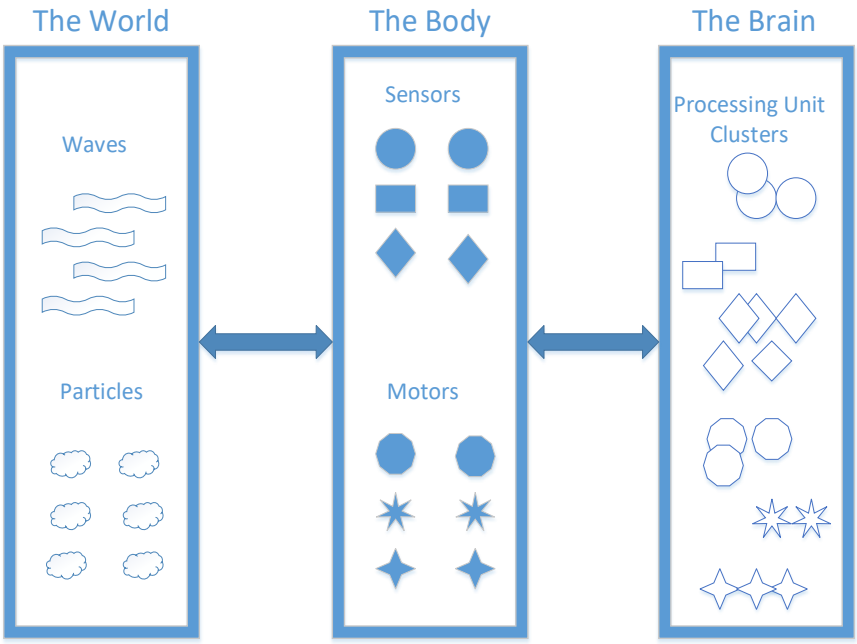

Figure 1 - Somatic Processing Model

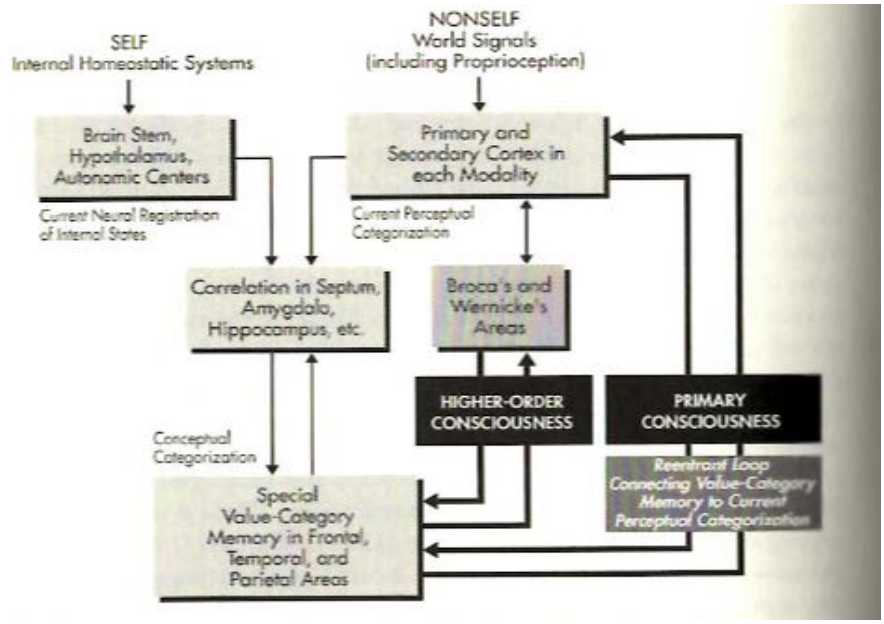

Figure 2 - A Scheme for higher-order consciousness (Edelman & Tononi, 2000, p. 194)

Here, I define some terms that I will use in describing dynamical systems and models thereof. I will use them repeatedly throughout the book, so you might want to mark this page. All of these definitions are standard.

1. The *state space* of a system is the space defined by the set of all possible states the system could ever be in.
2. A *trajectory* or *path* is a set of positions in the state space through which the system might pass successively. The behavior of the system is often described by trajectories through the state space.
3. An *attractor* is a point of state space to which the system will tend when in the surrounding region.
4. A *repeller* is a point of state space away from which the system will tend when in the surrounding region.
5. The *topology* of a state space is the layout of attractors and repellors in the state space.
6. A *control parameter* is some parameter of a system whose continuous quantitative change leads to a noncontinuous, qualitative change in the topology of a state space.
7. A differential equation $dx/dt = F(x)$ for variables $x_1 \ldots x_n$ is *linear* if none of $x_1 \ldots x_n$ or functions of $x_1 \ldots x_n$ are among the coefficients of F. Otherwise, the equation is *nonlinear*.
8. Systems that can be modeled with linear differential equations are called *linear systems*. Systems that can only be modeled with nonlinear differential equations are called *nonlinear systems*.
9. Only linear systems are *decomposable*; that is, only linear systems can be modeled as collections of separable components. Nonlinear systems are *nondecomposable*.
10. Nondecomposable, nonlinear systems can only be characterized using global *collective variables* and/or *order parameters*, variables or parameters of the system that summarize the behavior of the system's components.

Figure 3 - Dynamical Systems Terminology

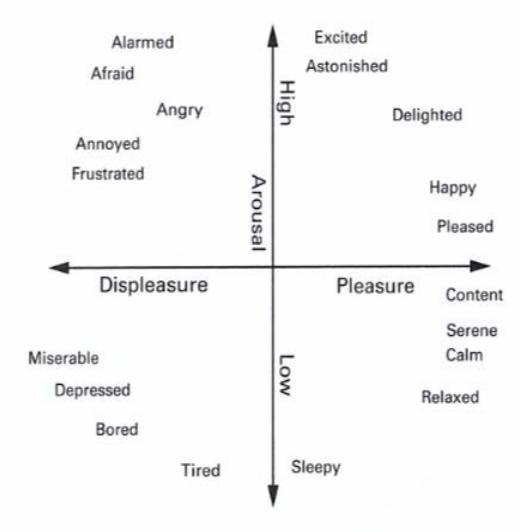

Figure 4 - A two-dimensional emotion space (Russell, 1980)

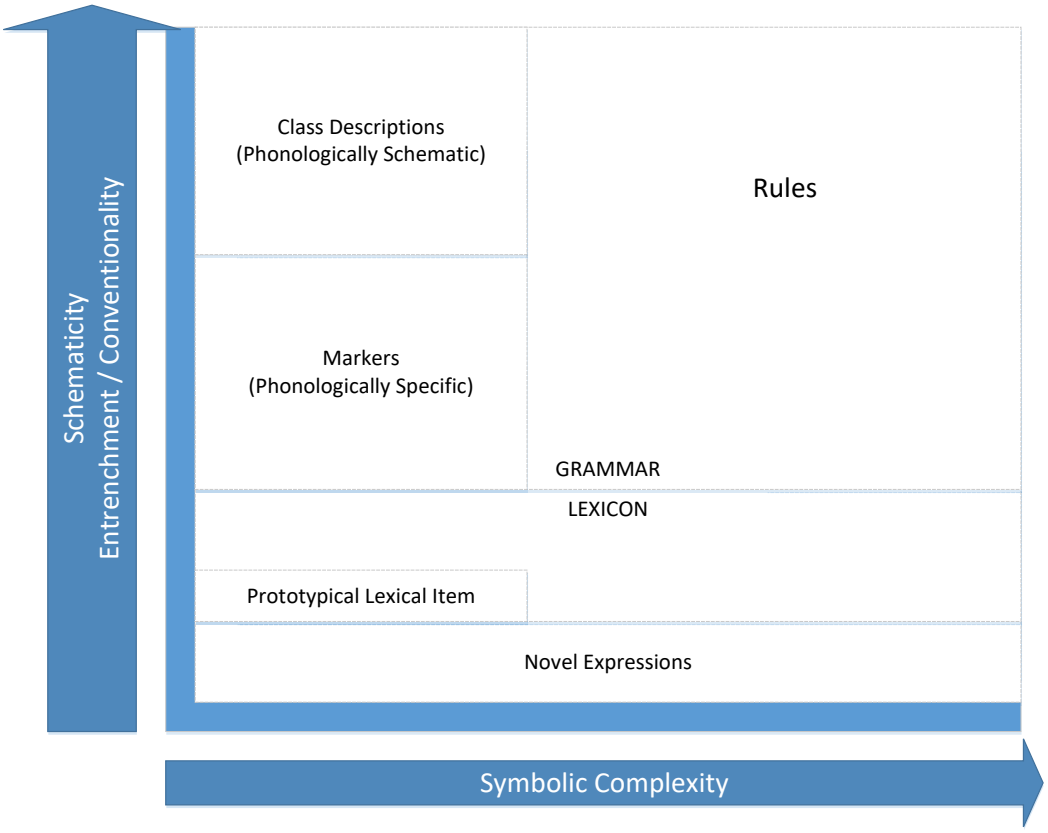

Figure 5 Conventional Unit Status

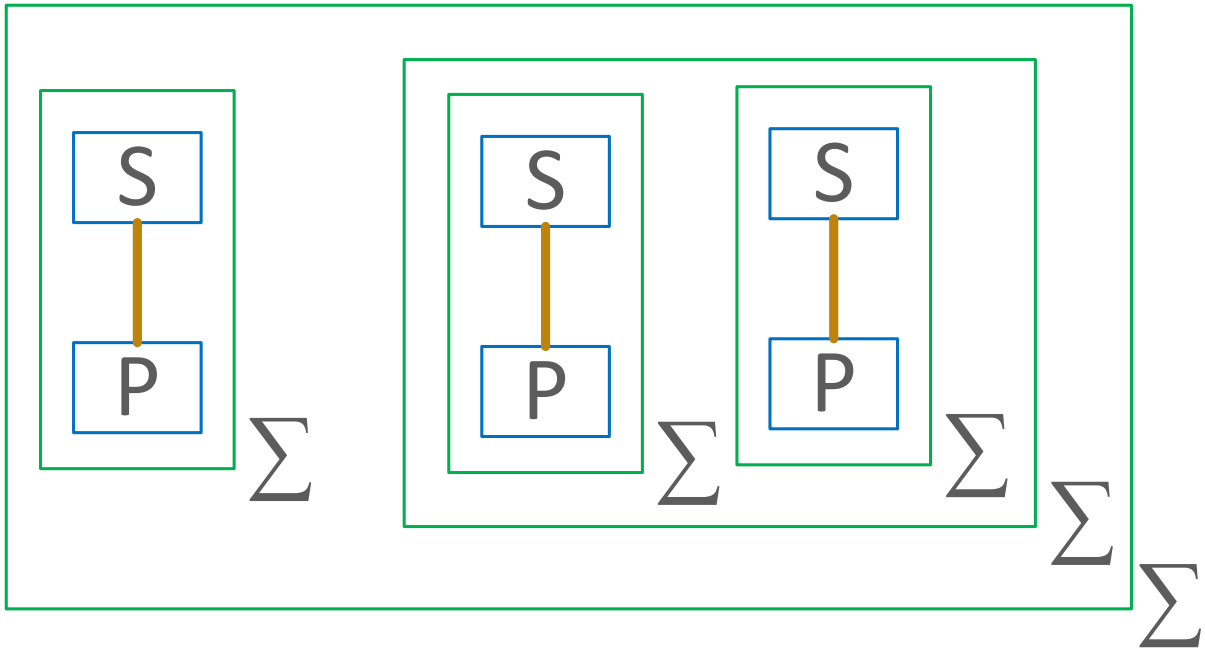

Figure 6 Symbolic Assembly

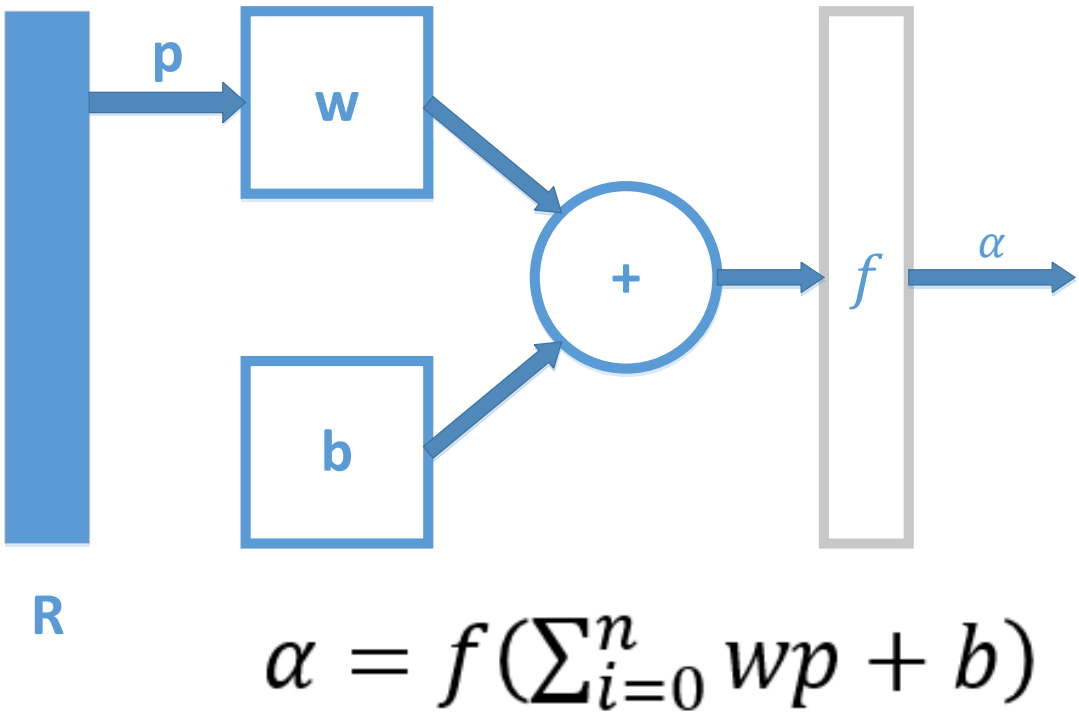

Figure 7 Forward-Propagation

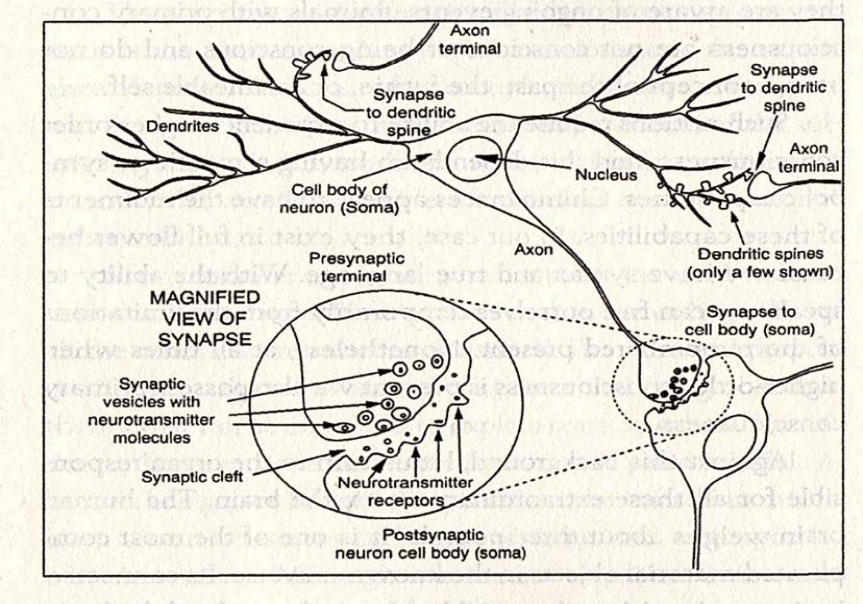

Figure 8 Neuron

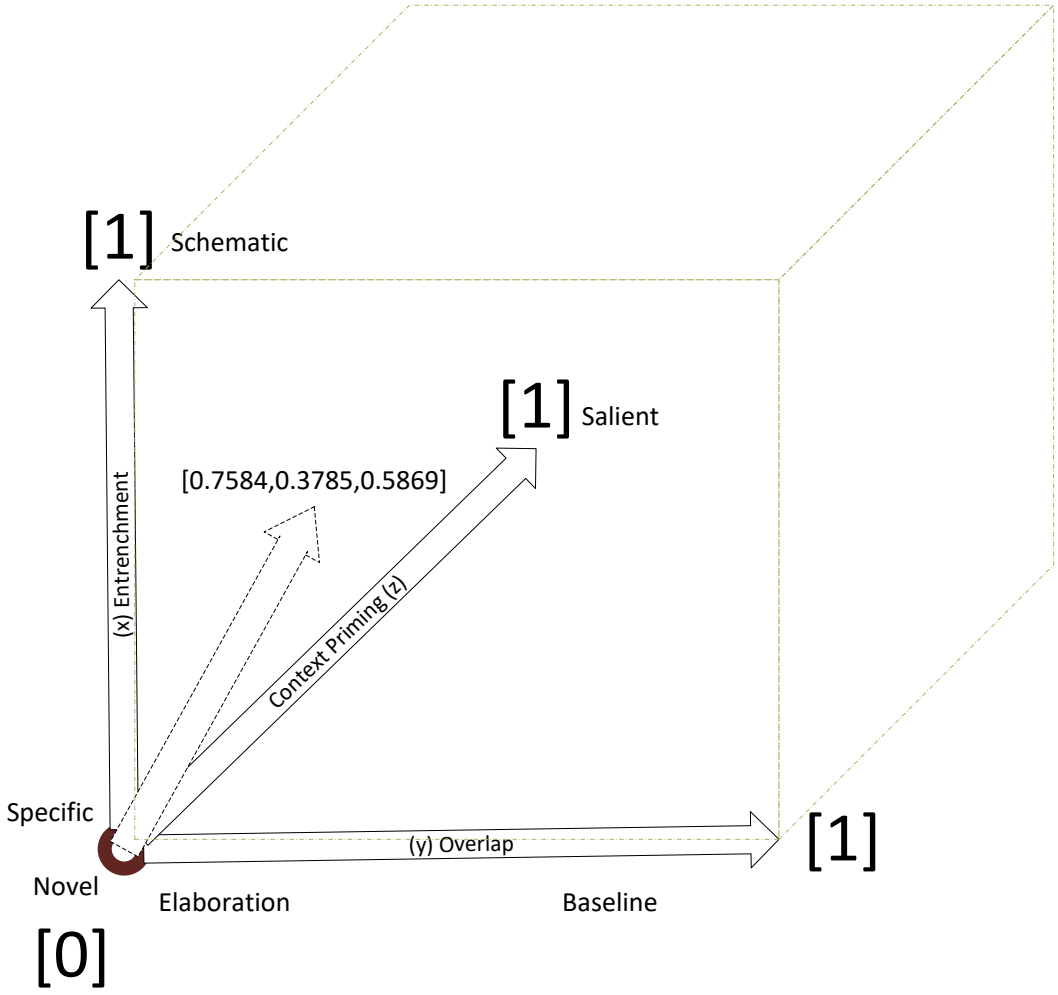

Figure 9 Symbolic Unit Bias Vector

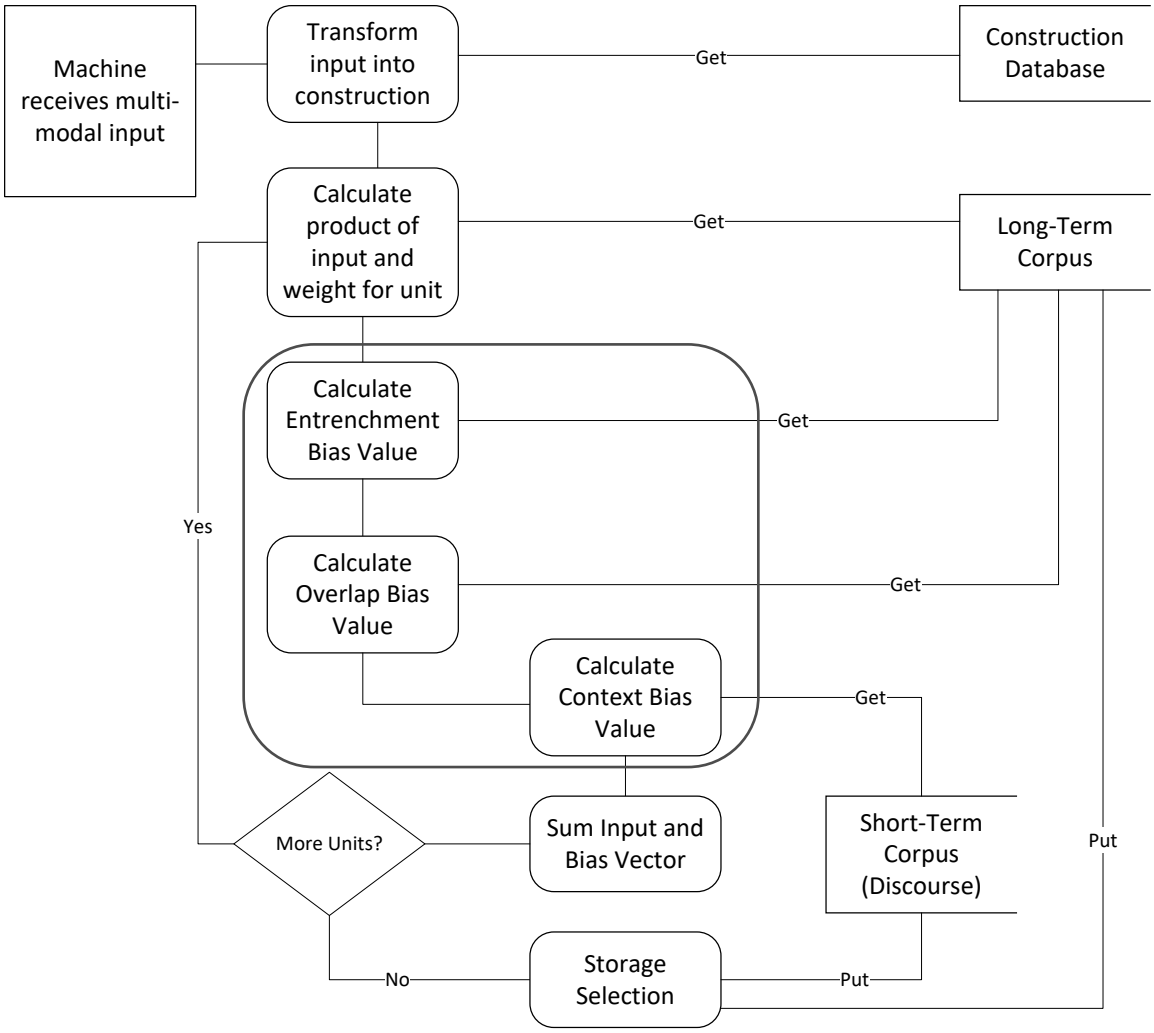

Figure 10 Dynamic System Matching Process

```
WEBVTT
Kind: captions
Language: en

00:00:01.170 --> 00:00:15.650
Hey, welcome back to TED.

00:00:15.650 --> 00:00:17.490
It's great to have you here.

00:00:17.490 --> 00:00:18.710
Thanks for having me.

00:00:18.710 --> 00:00:25.210
So, in the next half hour or so we're going
to spend some time exploring your vision for
```

Figure 11 - Closed Caption File Sample

```
["guess", "VBP",
[["Synset('think.v.02')", "[Synset('expect.v.01')]", 0.8571428571428571],
["Synset('guess.v.02')", "[Synset('speculate.v.02')]", 0.8571428571428571],
["Synset('estimate.v.01')", "[Synset('calculate.v.01')]", 0.8571428571428571],
["Synset('guess.v.04')", "[Synset('solve.v.01')]", 0.8]]]
```

Figure 12 - Synsets for target word "guess"

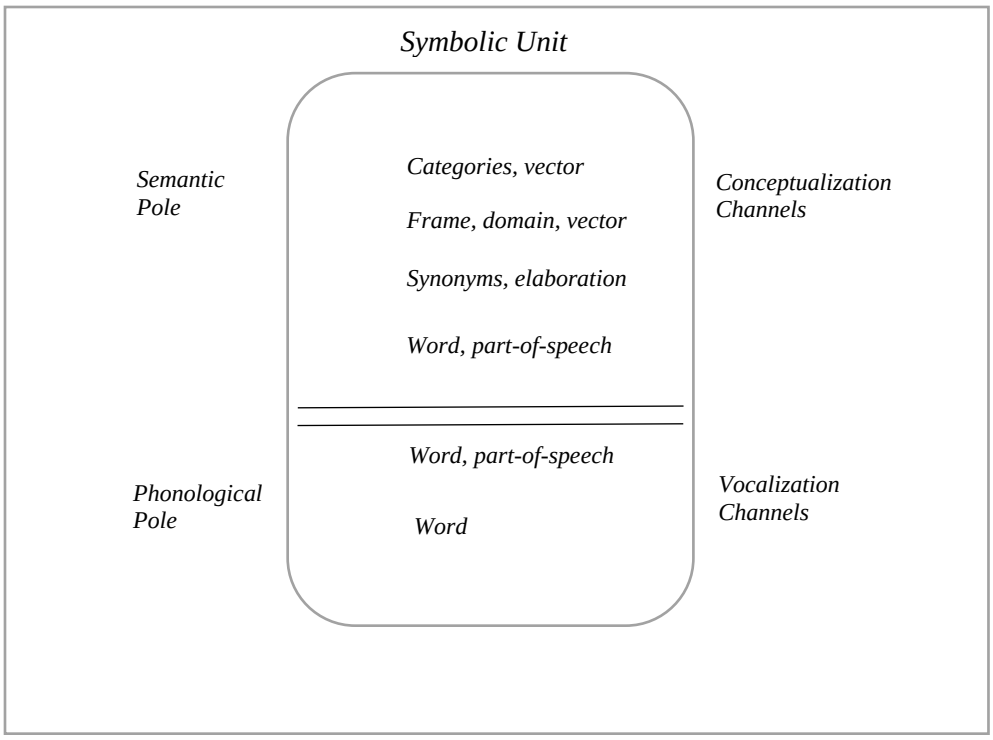

Figure 13 - YouTube corpus Symbolic Unit

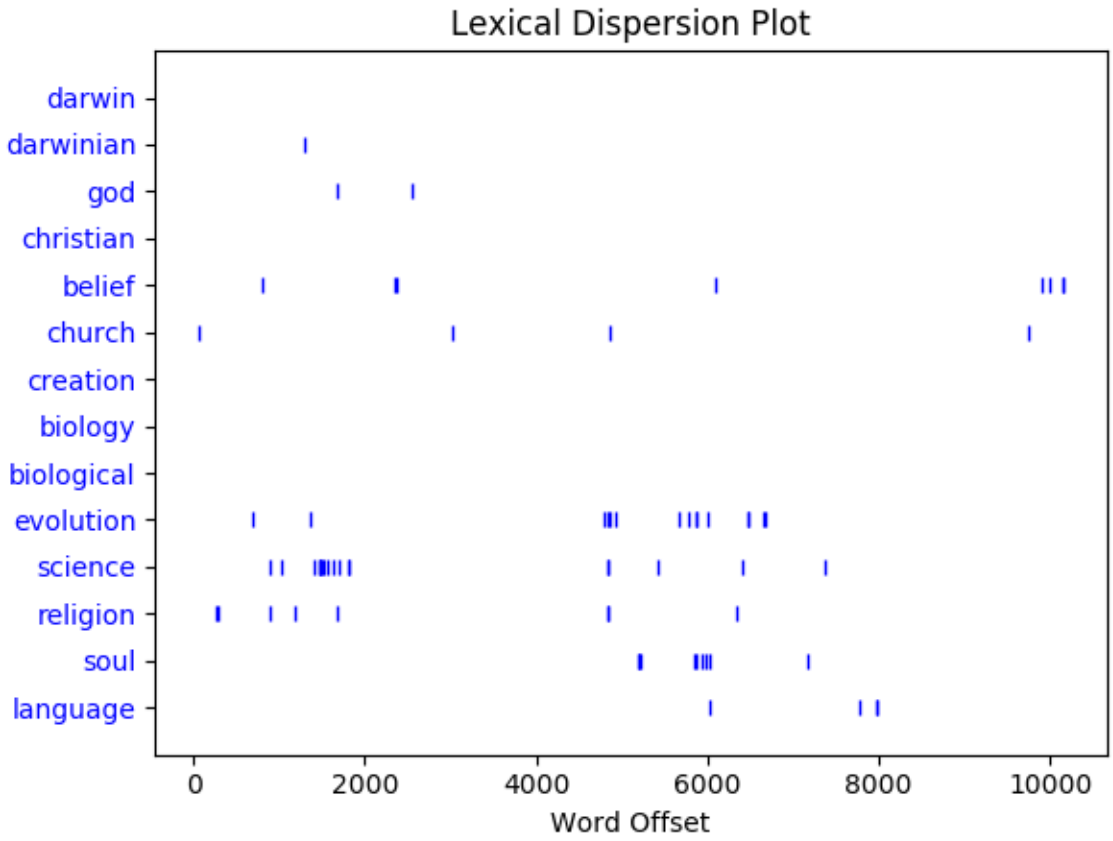

Figure 14 - Dispersion Plot of discourse Creationism vs. Evolution

1.

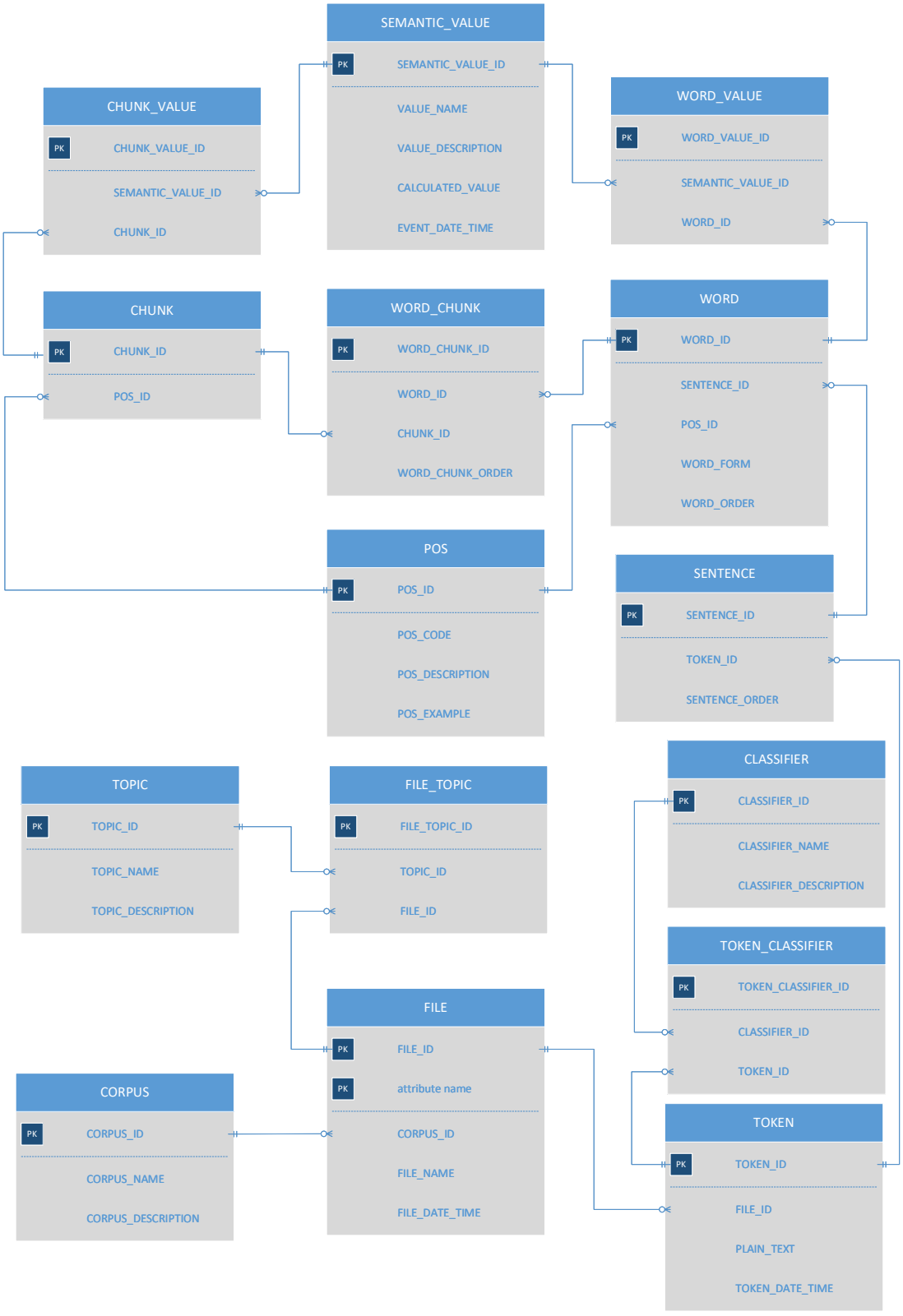

Figure 15 - Memory Corpora Data Model

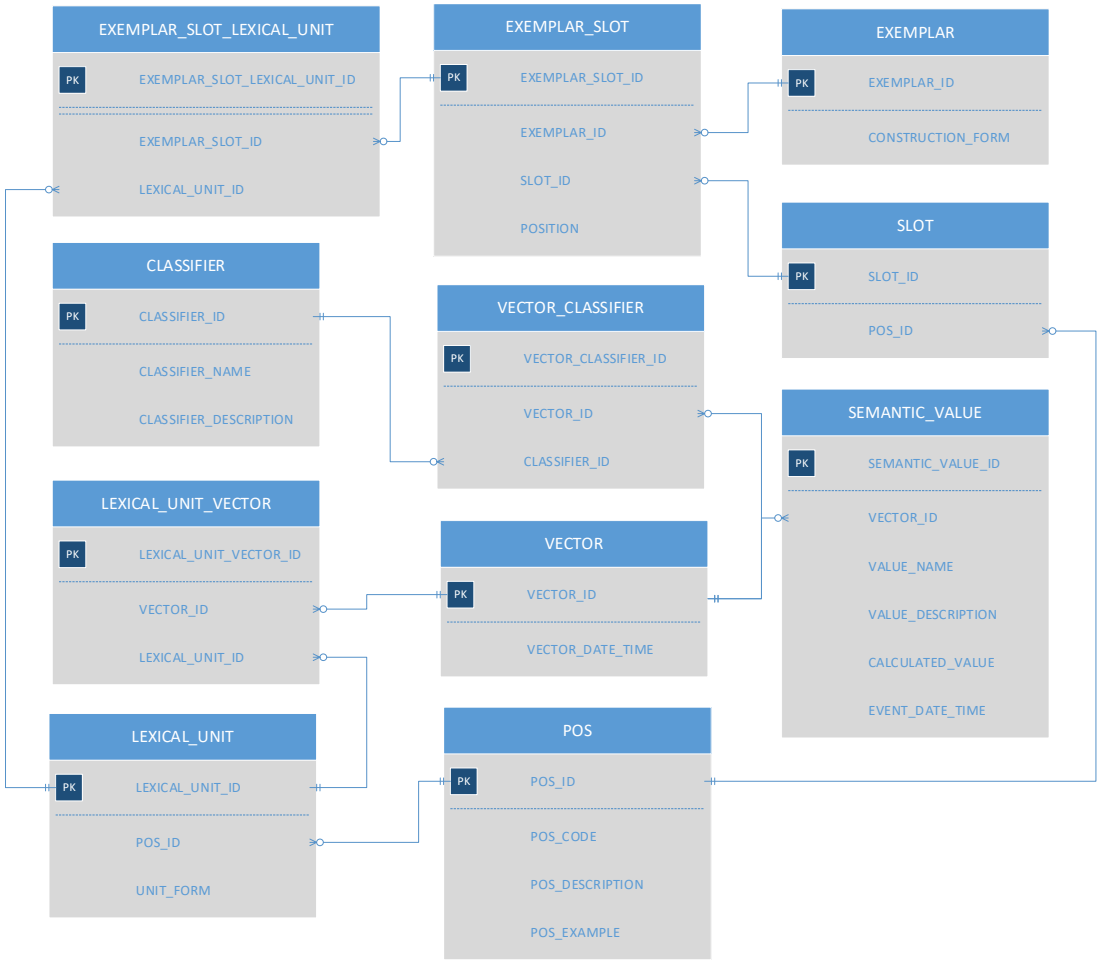

Figure 16 – Exemplar Construction Data Model

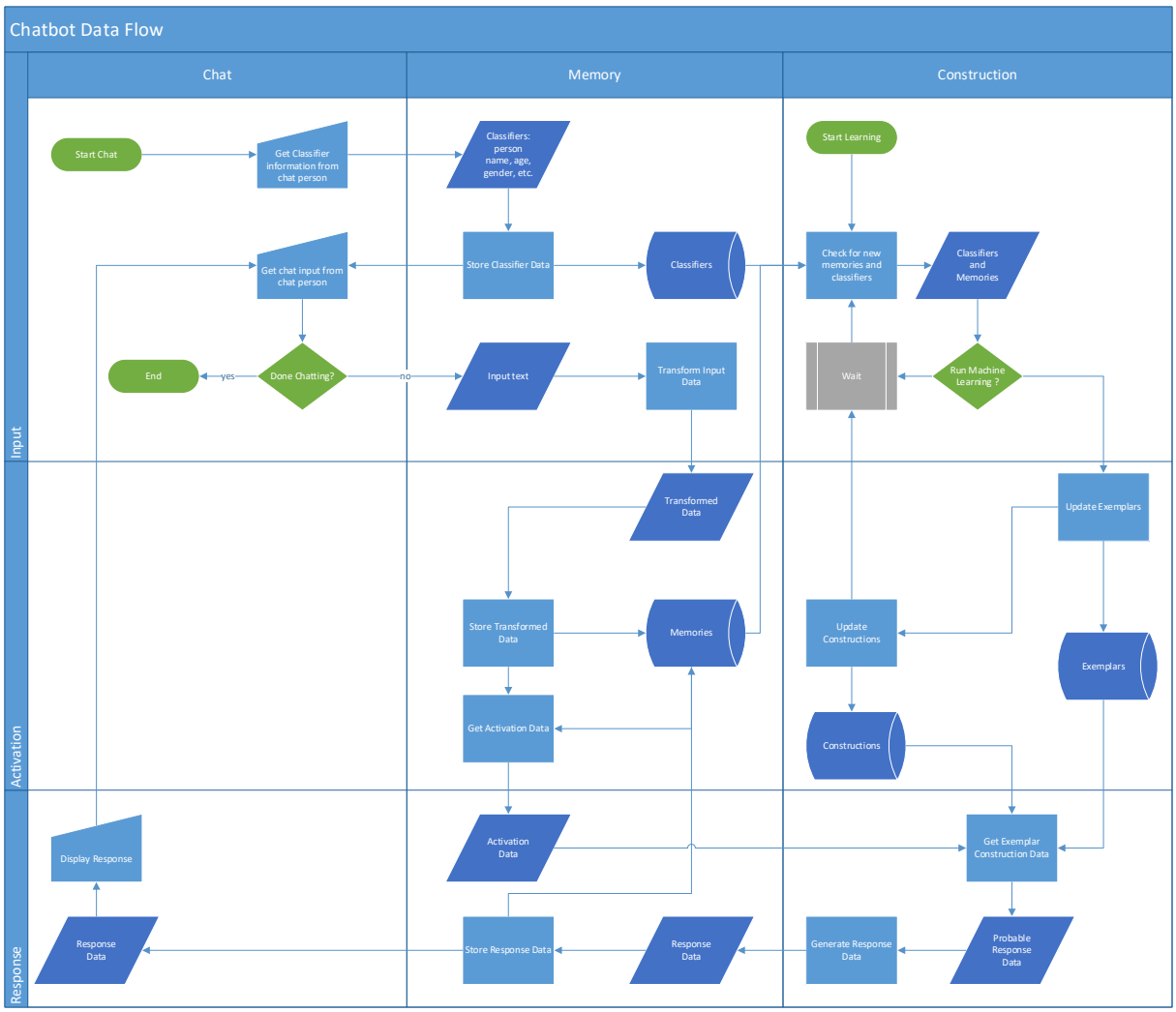

Figure 17 - Chatbot Data Flow

7 Tables

-

Table 1 - Modal Verbs Frequency by Category

	would	could	should	will	can	shall	must
Nonprofits	384	214	132	189	514	8	18
People	485	241	209	265	537	3	9
News	639	321	233	416	789	2	31
Politics	639	321	233	416	789	2	31
Style	3	1	0	1	9	0	0
Sports	59	31	14	11	54	0	3
Events	57	45	12	19	29	0	0
Autos	10	13	9	26	36	0	0
Education	848	516	292	437	1174	14	74
Howto	3	1	0	1	9	0	0
Comedy	25	9	17	1	18	0	0
Animation	53	34	10	17	36	1	4
Blogs	485	241	209	265	537	3	9
Activism	384	214	132	189	514	8	18
Music	157	107	34	49	195	2	7
Travel	57	45	12	19	29	0	0
Technology	142	93	53	132	247	2	6
Vehicles	10	13	9	26	36	0	0
Science	142	93	53	132	247	2	6
Entertainment	195	83	84	94	260	1	10
Film	53	34	10	17	36	1	4
all	3096	1750	1105	1681	3973	33	165

Table 2 - YouTube Videos for Corpus

Video Id	Words	Set	Time	Description
-XFaEFNALqU	9,779	1,613	01:07:28.560	Dawn Eden - Courage 2013 #5 - CONF 222 - Education
0mJXSnpKqJc	5,447	1,457	00:44:13.006	Art Talks: Dr Loretta Würtenberger and Melanie Gerlis discuss 'The Artist's Estate' - Nonprofits Activism
0Ttrb-97tFA	13,293	2,672	01:19:38.945	Naomi Wolf and Jim Pfaus talk sex - Education
0wGsXO1vzNI	18,600	2,509	01:21:41.144	SOFREP Radio: Green Beret Terry Schappert in studio talking "Hollywood Weapons" - Travel Events

0xPlja9a2Cs	12,338	1,881	01:00:30.659	PRO/CON at The Pier: The Return of Civil Discourse - Public Education: Is It Broken? - Nonprofits Activism
1LLbCU6QxoM	10,106	1,552	00:54:07.285	Steve Jobs 1983 Gives A Talk About The Future - Science Technology
2jtdrIcEXus	1,584	482	00:10:58.002	Jackie Chan On Why We Seeks Variety In His Roles: 'I Want To Be Like Robert De Niro' - Entertainment
2lgvd5wsWG0	16,903	2,254	01:30:25.430	#037: David & Anna Discuss Amanda Palmer: The art of asking - Music
2LwaVr_OgZE	10,539	1,537	01:11:15.835	'Women in the Arts' – Siri Hustvedt, Katharina Grosse, Nicola Graef at me Collectors Room Berlin - Education
2ZVA0vwHMQs	15,928	2,153	01:39:02.356	Same-Sex Marriage Debate: Gallagher vs. Corvino - Education
3eVclwNQHJo	8,852	1,313	00:57:35.883	Marcus and Anthony Discuss Polyamory - People Blogs
3PFrdfxdYPI	1,537	478	00:08:09.685	Richard Dawkins Interview - Sky News - Entertainment
4-4F95jtcxI	6,742	1,283	00:34:15.716	Warren Buffett Candid Interview 2015 - People Blogs
4C_MLzjb0bI	10,156	1,595	00:52:55.459	Intelligence to Protect the Homeland and the Way Ahead - Autos Vehicles
4e2kJhAGPCE	4,006	791	00:24:12.615	PEACE, LOVE AND MISUNDERSTANDING | indieWIRE | TIFF Industry 2011 - Film Animation
5ir1hhpkwbo	23,994	660	00:19:06.395	Jimmy Kimmel's FULL INTERVIEW with President George W. Bush - Comedy
6F4M2tDlAAQ	6,806	1,269	00:32:04.896	Mark Cuban Interview 2017 - Talks Tech, Business, Investing - Science Technology
6HffK1ZxVZ4	2,059	687	00:10:36.846	much ado group teaching project - Music
6NOSD0XK0r8	18,504	2,162	01:35:26.826	Fired Google Engineer James Damore (Live Interview) - News Politics
7M5c71l9tno	1,988	556	00:14:42.085	The Tango Cafe Legacy - Conversation between friends - Education
7MlGiL2lgKA	3,703	734	00:19:35.471	Talking With Tea & Bee: Real Conversation between Friends :) - Tianna Thompson - People Blogs
7NMOMscsA2c	1,243	409	00:06:13.187	Heated debate on gun control - People Blogs
7PxFnKFFvK4	3,748	822	00:20:58.781	Women Discuss Being Pro Casual Sex - People Blogs
8IiDgZK-Fz4	21,197	3,015	02:04:44.722	Sex and Speech on the College Campus - Roundtable Discussion - Education
9HKlzk4xKl8	2,437	716	00:13:11.555	Funniest Local News interviews 😀😀😀 - People Blogs

ad9L3zWcWIo	1,882	613	00:10:25.871	Sarah Huckabee Heated Exchange vs CNN Jim Acosta, San Juan mayor, private jet, Rex Tillerson 10/5/17 - People Blogs
Alzqh8x9OpI	5,159	953	00:26:43.826	Caroline Kennedy | CONVERSATIONS AT KCTS 9 - People Blogs
APC2jnOSfhQ	4,207	823	00:21:58.152	REAL CONVERSATIONS: I'm Pro-Gun | Change My Mind - Comedy
avlAa4KUm-Y	454	190	00:03:31.035	#7 Hilarious Language Barrier Misunderstanding! : Yurgei Meets Monica - Entertainment
B14uaSxLong	14,609	2,474	01:28:39.428	What is the Use of Ornament in Contemporary Art and Architecture? - Nonprofits Activism
B1EhafsWudQ	13,317	1,913	01:18:02.111	VIDEO: 8/1/2017 - The Chief talks Climate Change with Bailey Hall for Climate Corps - Entertainment
BgBs5BNHYNQ	9,654	1,378	00:47:14.525	Andy Stanley, Michael Leahy and ex-wife talk about Michael's sex addiction - Nonprofits Activism
bmovaPIsHa0	7990	1,524	00:52:41.762	EDWARD SNOWDEN EXPOSES DONALD TRUMP FULL INTERVIEW 2017 - News Politics
byT2P4OxaBE	8,538	1,618	00:49:19.078	The Future of Economy | Panel Discussion - Education
bZYwZDqdsas	4,576	731	00:23:45.847	Advanced English Conversation About Travel [The Fearless Fluency Club] - Education
cEProM1NcvU	9805	1741	01:17:34.775	TRAC2014: Roger Scruton and Odd Nerdrum - Contemporary Representational Aesthetics - Education
CkObh3RZKXU	16,368	2,096	01:47:39.051	CLE812 - Various Panelists - Life After Death Life After Life Panel - Entertainment
cOzSYkk3ZM0	12,915	2,069	01:23:26.437	Point-Counterpoint Discussion on Rail Transit - Education
cYhjo5O-nfg	11,000	2,273	01:15:46.290	Mastering Style: The Learning and Teaching of Writing - Education
dLmcZ9dGBk4	5,587	1,057	00:26:44.815	Colin Goddard & Kristina Anderson | CONVERSATIONS AT KCTS 9 - Nonprofits Activism
dmbSENYk-KI	5,482	1,146	00:40:16.936	Peter, Paul and Mary's Peter Yarrow candid feature interview - Music
dqq4TMXxq1E	2,913	657	00:20:02.922	Real English Conversation: My Wife and I Answer Your Questions! - Entertainment
e5MD-2GTqto	483	208	00:02:19.829	Chicken Connoisseur tv interview - Entertainment
EG3Y8Cp-9NA	16,847	2,413	01:48:52.933	Drinking Water: A Crisis in Every State - News Politics
Erkp675dLrM	167	102	00:00:56.812	Hurricane Harvey - Awkward News Interview Question - Comedy

eymykfdIIpc	1,545	432	00:07:09.955	Chatting with Friends: Wakfu - Entertainment
fe9fZxfsqwM	3,496	773	00:20:38.388	Connie and Samuel Johnson discuss Love Your Sister, unicycles, and mortality - Entertainment
fLcHfHZ1k9A	16,292	2,437	01:38:03.223	Death at SeaWorld Panel Discussion: Author David Kirby, Dr Naomi Rose and Dr Lori Marino - Nonprofits Activism
FnrJ3jPWG68	2,167	568	00:12:08.437	We're Cursed w/ Keith Lemelin | Karla's Car Conversations - People Blogs
g--WzSUmkdk	13,978	1,690	01:11:42.449	A conversation with Hemant Mehta (ex-Jain, editor of the Friendly Äthïest) - Nonprofits Activism
gd0oSNjHf1A	9,247	1,385	00:38:50.052	Joe Rogan vs Steven Crowder: Heated Argument over Marijuana - People Blogs
GDF-8PiM_vg	3,373	729	00:17:34.006	James O'Keefe Uncovers Evil Machinations Of Mainstream Media | Cerno News Interview - News Politics
GhVzaEGxTw4	11,317	1,479	00:58:45.941	Coach Unplugged Interview (Mike McGivern) - Sports
gWT-EWKIR3M	2,114	673	00:10:37.047	Climate Realist Marc Morano Debates Bill Nye the Science Guy on Global Warming - News Politics
gZKDInabaPM	2,305	590	00:11:05.878	I debate with Dietitian on LIVE TV this morning - My reaction - People Blogs
h0962biiZa4	10891	1751	01:00:02.110	Superintelligence: Science or Fiction? | Elon Musk & Other Great Minds - People Blogs
h4cSZLP8cwA	1,070	284	00:03:34.293	Tim and Eric Get Into a Heated Discussion | THE ULTIMATE FIGHTER - Sports
H4h44yN_QTg	10,708	1,933	01:10:24.799	39 Leston Havens MD: #3 Patient Interview and Discussion: War Neurosis or Malingering? - Education
hpDHwfXbpfg	6,863	1,409	00:40:40.581	Elon Musk Interview 2017 | TEDTalk - Science Technology
hRniRF2BAus	12,048	1,533	01:03:11.000	HOW TO CHANGE YOUR PAST & YOUR FUTURE - CONVERSATION WITH BRIDGET NIELSEN - People Blogs
Hz9FqepcRUM	17,866	2,358	01:22:37.990	FULL Steve Bannon Interview with Charlie Rose - News Politics
IhqDbLPvKsM	10,992	1,595	00:54:14.037	Buffy Panel Discussion - Film Animation
IiqAVxT0FUw	1,126	331	00:11:38.112	Conversations with Curl Friends at Curl Fest! - Howto Style
iOKePIvoNcI	9,591	1,416	01:08:29.578	Sex Talk: Group Discussion on Sex, Purity, and Holiness - People Blogs
j1xDizRZw3I	5,367	903	00:31:04.787	Interview with Vanessa [Featuring Jack from ToFluency] - Education
Jk0PndXxSoQ	14,670	1,966	01:32:48.613	Dont Call Me Crazy: How we Fell in Love With Outsider Art - Nonprofits Activism

jsJeE9suuBc	70,635	2,481	02:20:25.687	A Public Discussion About Race in Boston - News Politics
jX86JCbkSUI	13,552	1,622	01:15:01.850	Fireside Chat #6: Yakov Boyko with Khepra Anu. Raw Food. Nutrition. Health. - Education
KCDVEn5Wzmg	12,466	1,860	01:08:00.483	Every Bot Is a Critic: 03.06.17 at New Lab - Science Technology
KKjKCec8i6c	9,392	1,581	00:52:48.843	Jeff Bezos - A Candid Interview About The Amazon Story - Science Technology
KncbeLE9HS0	28,342	3,548	02:54:53.013	Amazing Interview With Astrophysicist Neil deGrasse Tyson - News Politics
Kp-wGkzDLSw	14,565	2,202	01:22:08.283	Jonathan Haidt - Panel Discussion - Future of Cities - NYU - People Blogs
KRLI42qd_wQ	13,869	1,929	01:13:31.662	Tech Talk: Marketing to Today's Diner - Education
LduioANhxlA	13,902	2,184	01:25:12.149	Watch This! Artist Panel Discussion - Education
lgjqYRdgkTw	8,786	1,216	00:42:10.285	Raising Kids With or Without Religion: The Mom's View Live - People Blogs
lgK6qLBLVD0	10,539	2,054	01:13:37.539	Counter Terrorism Discussion - Education
Lp9zuo52Njo	986	327	00:06:16.238	Spontaneous Road Trips w/Dayviideo| Karla's Car Conversations - People Blogs
ltbADstPdek	12,468	2,054	01:16:56.682	Richard Dawkins & Neil deGrasse Tyson - Education
mhvw0Jrevqk	12,733	2,367	01:24:42.993	Presidential Leadership Scholars 2017 Graduation - Nonprofits Activism
N4aXeJ3Z29w	2,799	649	00:15:15.622	Conversations on Death with Kim Mooney - People Blogs
N6m7pWEMPlA	2,682	198	00:03:12.720	Obamacare vs. Affordable Care Act #2 - Entertainment
n7IHU28aR2E	21,339	3,100	01:57:12.239	The Four Horseman - Hitchens, Dawkins, Dennet, Harris [2007] - Education
N8zIiSaETqk	11,343	1,600	01:01:34.369	Nerd HQ 2016: A Conversation with Zac and Friends - Entertainment
nEA0oW9TSjw	11,497	1,845	01:00:31.632	Lowkey In Discussion - #FlipLifeRadio - PyroRadio - (15/09/2017) - Music
nHHDoywUfCQ	1,481	418	00:08:19.883	Jimmy Fallon Interview on Live with Kelly and Ryan - People Blogs
NWdc7PyZNLA	5,023	718	00:18:24.304	Guest Host Jennifer Lawrence Interviews Kim Kardashian West - Comedy
oG804t0WG-c	12,921	1,959	01:06:30.359	Dan Savage vs. Brian Brown: The Dinner Table Debate - News Politics
ovN6ntFy0zI	5,344	775	00:23:37.703	SELF LOVE or SELFISH? | The Mom's View - People Blogs
PJIvBeVKoQA	173	111	00:01:22.818	Anchorman can't stop laughing! - World's funniest live news interview - People Blogs
QgJPYJ0Jn04	12,815	2,080	01:06:57.547	Fixing the System: VICE on HBO Special Report (Full Episode) - News Politics

qKnBIiIUv9A	7,684	1,421	00:43:41.014	Bill Maher On The Messy Truth With Van Jones - Full Show - News Politics
R0HpE1yh_sw	11,529	1,765	01:12:12.366	GENERAL SESSION: How Transit Agencies are Integrating Rideshare and Public Transportation - Nonprofits Activism
rmpi3uYK3A4	7,223	1,234	00:50:13.586	Artist and Curator: A Conversation - Education
RSFNe0AowwM	4,865	1,223	00:25:49.055	Kieser Report: Cryptocurrency Taking Russia By Storm? (E1146) - News Politics
rtIE5evvHGg	12,098	1,900	01:10:00.899	Panel Discussion Sustaining Relationships - Education
sT40DH6hdKs	10,401	1,638	01:03:12.109	Discussion Panel | Blockchain and Bitcoin will Disrupt Entire Industries and Governments - News Politics
t4jUQGbHhUw	16,527	2,595	01:42:56.702	St. John's College Special Panel Discussion - "Implications of the Death of Osama Bin Laden" - Education
tAZGxRmxMH0	14,934	1,968	01:23:44.749	Bernie Sanders VS. Ted Cruz on The GOP Tax Plan. #Breaking #TaxReform #BernieSanders #TedCruz - News Politics
ThHzIJUm-So	9,627	1,682	00:54:40.883	Warren Buffett And Bill Gates - January 2017 - People Blogs
Tm6ESsMlvYE	13,042	1,889	01:24:06.929	Gloria Steinem and Emma Watson in Conversation - People Blogs
tzzoVNR5wkM	11,635	1,778	01:18:20.689	The Trouble With Sculpture - Nonprofits Activism
uBz8uzJEJxI	17,171	1,971	01:27:50.449	The Trouble With Painting - Nonprofits Activism
UKM3ac_6CVs	173	113	00:01:30.581	UPDATE #7 Sgnt. Hurricane Harvey The Hunkered Down Hurricane Hawk - People Blogs
Un0ohUagTWo	6,756	1,118	00:41:05.681	First Ladies Laura Bush and Michelle Obama at Investing in Our Future - Nonprofits Activism
v9wzw5nFVfc	9,600	1,885	00:55:09.008	Let’s talk about men sexual abuse – The Urban Debate (July 6) - News Politics
vln9D81eO60	2,065	583	00:10:06.179	Ben Affleck, Sam Harris and Bill Maher Debate Radical Islam | Real Time with Bill Maher (HBO) - Entertainment
vqJM80MZQQ8	10,647	1,499	00:49:59.605	Danny Meyer & Michael Romano, "Family Table" | Talks at Google - Music
Vv1x4GHpU7E	269	148	00:02:52.625	UH Thea 1331 "much ado about nothing soap opera" - People Blogs
W1Sa6CzHpiE	13,044	1,666	01:30:43.659	My Friends Are Gonna Be Strangers: A Conversation with Merle Haggard, Norm Hamlet, Don Markham. - Music
wLx4DjeMtz8	8,014	1,296	00:55:49.710	Apple CEO, Tim Cook Interview On Steve Jobs, AR, Heros, The Future - Science Technology

xvOtab2vn2Y	299	139	00:02:19.275	Jurgen Klopp's disagreement with a journalist - Sports
y2KwVRiuILk	10,300	1,965	01:04:09.140	Artist panel discussion on Form & Story - Education
y8hy8NxZvFY	9,938	1,887	00:59:56.774	DEBATE: Atheist vs Christian (Richard Dawkins vs Cardinal George Pell) - Education
YCuxezFZUVg	4,615	715	00:20:10.933	Cheating?!? - Entertainment
yI_h0loITWM	11,087	1,446	00:52:58.187	Howard Stern and Bill Maher Discuss His Romanticization of Al-Qaeda's 9/11 Terror Invasion - Entertainment
YsQ6Relf7bw	10,586	1,613	00:49:13.168	The GORUCK Show: Travel Like a Green Beret - People Blogs
yYeS3gNQnaQ	14,221	2,652	01:40:46.229	Imagining Queer Justice: Prison Abolition and LGBT Hate Crime Legislation - Education
z6gB3gA9UZg	43,182	3,715	03:58:51.004	Debate: Socialism vs Capitalism - News Politics
ZWE8HUjYf8Q	6,068	1,361	00:47:31.852	Obama's Full Speech and Q&A with Bill and Melinda Gates - News Politics
zylMee7xJ0c	2,123	532	00:13:47.706	Conversations with Friends: The Gender Spectrum and Sexuality - Film Animation
120 Videos	1, 159,8 43	34,948	105:13:41.000	

Table 3 - NLTK Part of Speech Tags

POS	DESCRIPTION	EXAMPLE
$	dollar	$ -$ --$ A$ C$ HK$ M$ NZ$ S$ U.S.$ US$
''	closing quotation mark	' ''
(	opening parenthesis	([{
)	closing parenthesis	)] }
,	comma	,
--	dash	--
.	sentence terminator	. ! ?
:	colon or ellipsis	; ...
CC	conjunction, coordinating	& 'n and both but either et for less minus neither nor or plus so therefore times v. versus vs. whether yet

CD	numeral, cardinal	mid-1890 nine-thirty forty-two one-tenth ten million 0.5 one forty- seven 1987 twenty '79 zero two 78-degrees eighty-four IX '60s .025 fifteen 271,124 dozen quintillion DM2,000 ...
DT	determiner	all an another any both del each either every half la many much nary neither no some such that the them these this those
EX	existential there	there
FW	foreign word	gemeinschaft hund ich jeux habeas Haementeria Herr K'ang-si vous lutihaw alai je jour objets salutaris fille quibusdam pas trop Monte terram fiche oui corporis ...
IN	preposition or conjunction, subordinating	astride among uppon whether out inside pro despite on by throughout below within for towards near behind atop around if like until below next into if beside ...
JJ	adjective or numeral, ordinal	third ill-mannered pre-war regrettable oiled calamitous first separable ectoplasmic battery-powered participatory fourth still-to-be-named multilingual multi-disciplinary ...
JJR	adjective, comparative	bleaker braver breezier briefer brighter brisker broader bumper busier calmer cheaper choosier cleaner clearer closer colder commoner costlier cozier creamier crunchier cuter ...
JJS	adjective, superlative	calmest cheapest choicest classiest cleanest clearest closest commonest corniest costliest crassest creepiest crudest cutest darkest deadliest dearest deepest densest dinkiest ...

LS	list item marker	A A. B B. C C. D E F First G H I J K One SP-44001 SP-44002 SP-44005 SP-44007 Second Third Three Two * a b c d first five four one six three two
MD	modal auxiliary	can cannot could couldn't dare may might must need ought shall should shouldn't will would
NN	noun, common, singular or mass	common-carrier cabbage knuckle-duster Casino afghan shed thermostat investment slide humour falloff slick wind hyena override subhumanity machinist ...
NNP	noun, proper, singular	Motown Venneboerger Czestochwa Ranzer Conchita Trumplane Christos Oceanside Escobar Kreisler Sawyer Cougar Yvette Ervin ODI Darryl CTCA Shannon A.K.C. Meltex Liverpool ...
NNPS	noun, proper, plural	Americans Americas Amharas Amityvilles Amusements Anarcho-Syndicalists Andalusians Andes Andruses Angels Animals Anthony Antilles Antiques Apache Apaches Apocrypha ...
NNS	noun, common, plural	undergraduates scotches bric-a-brac products bodyguards facets coasts divestitures storehouses designs clubs fragrances averages subjectivists apprehensions muses factory-jobs ...
PDT	pre-determiner	all both half many quite such sure this
POS	genitive marker	' 's
PRP	pronoun, personal	hers herself him himself hisself it itself me myself one oneself ours ourselves ownself self she thee theirs them themselves they thou thy us

PRP $	pronoun, possessive	her his mine my our ours their thy your
RB	adverb	occasionally unabatingly maddeningly adventurously professedly stirringly prominently technologically magisterially predominately swiftly fiscally pitilessly ...
RBR	adverb, comparative	further gloomier grander graver greater grimmer harder harsher healthier heavier higher however larger later leaner lengthier less- perfectly lesser lonelier longer louder lower more ...
RBS	adverb, superlative	best biggest bluntest earliest farthest first furthest hardest heartiest highest largest least less most nearest second tightest worst
RP	particle	aboard about across along apart around aside at away back before behind by crop down ever fast for forth from go high i.e. in into just later low more off on open out over per pie raising start teeth that through under unto up up-pp upon whole with you
SYM	symbol	% & ' '' ''.)). * + ,. < = > @ A[fj] U.S U.S.S.R * ** ***
TO	to as preposition or infinitive marker	to
UH	interjection	Goodbye Goody Gosh Wow Jeepers Jee-sus Hubba Hey Kee-reist Oops amen huh howdy uh dammit whammo shucks heck anyways whodunnit honey golly man baby diddle hush sonuvabitch ...

VB	verb, base form	ask assemble assess assign assume atone attention avoid bake balkanize bank begin behold believe bend benefit bevel beware bless boil bomb boost brace break bring broil brush build ...
VBD	verb, past tense	dipped pleaded swiped regummed soaked tidied convened halted registered cushioned exacted snubbed strode aimed adopted belied figgered speculated wore appreciated contemplated ...
VBG	verb, present participle or gerund	telegraphing stirring focusing angering judging stalling lactating hankerin' alleging veering capping approaching traveling besieging encrypting interrupting erasing wincing ...
VBN	verb, past participle	multihulled dilapidated aerosolized chaired languished panelized used experimented flourished imitated reunifed factored condensed sheared unsettled primed dubbed desired ...
VBP	verb, present tense, not 3rd person singular	predominate wrap resort sue twist spill cure lengthen brush terminate appear tend stray glisten obtain comprise detest tease attract emphasize mold postpone sever return wag ...
VBZ	verb, present tense, 3rd person singular	bases reconstructs marks mixes displeases seals carps weaves snatches slumps stretches authorizes smolders pictures emerges stockpiles seduces fizzes uses bolsters slaps speaks pleads ...
WDT	WH-determiner	that what whatever which whichever
WP	WH-pronoun	that what whatever whatsoever which who whom whosoever

WP$	WH-pronoun, possessive	whose
WRB	Wh-adverb	how however whence whenever where whereby wherever wherein whereof why
``	opening quotation mark	` ``

Table 4 - Stages of Corpus Data

Stage	Data
Raw	00:00:00.000 --> 00:00:05.069 align:start position:19% I've<00:00:00.149><c> had</c><00:00:00.420><c> obese</c><c.colorE5E5E5><00:00:01.199><c> nutritionist</c><00:00:02.159><c> try</c><00:00:02.490><c> and</c><00:00:02.639><c> tell</c></c> 00:00:02.850 --> 00:00:06.120 align:start position:19% me<c.colorCCCCCC><00:00:02.879><c> that</c><00:00:03.120><c> my</c></c><c.colorE5E5E5><00:00:03.629><c> diet</c><00:00:04.170><c> is</c><00:00:04.380><c> wrong</c><00:00:04.620><c> and</c><00:00:04.830><c> I'm</c><00:00:04.890><c> like</c></c> 00:00:05.069 --> 00:00:07.680 align:start position:19% hey<c.colorE5E5E5><00:00:05.279><c> just</c><00:00:05.549><c> because</c></c><c.colorCCCCCC><00:00:05.700><c> you've</c><00:00:05.790><c> got</c><00:00:05.819><c> a</c><00:00:05.910><c> bit</c></c><c.colorE5E5E5><00:00:06.029><c> of</c></c> 00:00:06.120 --> 00:00:09.360 align:start position:19% paper<c.colorE5E5E5><00:00:06.359><c> doesn't</c><00:00:07.109><c> mean</c><00:00:07.200><c> I'm</c><00:00:07.290><c> going</c><00:00:07.440><c> to</c><00:00:07.500><c> take</c></c>

```
00:00:07.680 --> 00:00:17.640 align:start position:19%
your<c.colorE5E5E5><00:00:07.830><c>
 advice</c></c><c.colorCCCCCC><00:00:07.919><c> I</c><00:00:08.370><c>
 want</c><00:00:08.580><c> to</c><00:00:08.670><c>
 see</c><00:00:08.820><c> the</c><00:00:08.940><c> results</c></c>
```


Plain

```
I 've had obese nutritionist try and tell me that my diet is wrong and I
 'm like hey just because you 've got a bit of paper does n't mean I 'm
 going to take your advice I want to see the results
```

Tagged

```
I/PRP 've/VBP had/VBN obese/JJ nutritionist/JJ try/NN and/CC tell/VB
 me/PRP that/IN my/PRP$ diet/NN is/VBZ wrong/JJ and/CC I/PRP 'm/VBP
 like/IN hey/NN just/RB because/IN you/PRP 've/VBP got/VBN a/DT bit/NN
 of/IN paper/NN does/VBZ n't/RB mean/VB I/PRP 'm/VBP going/VBG to/TO
 take/VB your/PRP$ advice/NN I/PRP want/VBP to/TO see/VB the/DT
 results/NNS
```

Frames


```
["I", "Compliance"], ["'ve", ""], ["had", ""], ["obese",
 "Body_description_holistic"], ["nutritionist", ""], ["try",
 "Isolated_places"], ["and", ""], ["tell", "Omen"], ["me", ""], ["that",
 ""], ["my", ""], ["diet", ""], ["is", ""], ["wrong",
 "Morality_evaluation"], ["and", ""], ["I", "Compliance"], ["'m", ""],
 ["like", "Similarity"], ["hey", ""], ["just", ""], ["because", ""],
 ["you", ""], ["'ve", ""], ["got", "Wearing"], ["a", ""], ["bit",
 "Quantified_mass"], ["of", ""], ["paper", "Text"], ["does", ""], ["n't",
 ""], ["mean", "Linguistic_meaning"], ["I", "Compliance"], ["'m", ""],
 ["going", "Getting_underway"], ["to", ""], ["take",
 "Getting_vehicle_underway"], ["your", ""], ["advice", ""], ["I",
 "Compliance"], ["want", "Possession"], ["to", ""], ["see", "Request"],
 ["the", ""], ["results", ""]
```


Word Nets


```
["I", "PRP", []], ["'ve", "VBP", []], ["had", "VBN",
 [["Synset('have.v.01')", "{}", 0], ["Synset('have.v.02')", "{}", 0],
 ["Synset('experience.v.03')", "[Synset('undergo.v.01')]", 0.8],
 ["Synset('own.v.01')", "{}", 0], ["Synset('get.v.03')",
 "[Synset('make.v.02')]", 0.8], ["Synset('consume.v.02')", "{}", 0],
 ["Synset('have.v.07')", "[Synset('interact.v.01')]", 0.8],
 ["Synset('hold.v.03')", "[Synset('direct.v.04')]", 0.8571428571428571],
 ["Synset('have.v.09')", "{}", 0], ["Synset('have.v.10')", "{}", 0],
```

["Synset('have.v.11')", "[Synset('change.v.02')]", 0.4],
["Synset('have.v.12')", "[Synset('suffer.v.06')]", 0.8],
["Synset('induce.v.02')", "{}", 0], ["Synset('accept.v.02')",
"[Synset('get.v.01')]", 0.4], ["Synset('receive.v.01')",
"[Synset('get.v.01')]", 0.4], ["Synset('suffer.v.02')",
"[Synset('experience.v.03')]", 0.8571428571428571],
["Synset('have.v.17')", "[Synset('score.v.01')]", 0.8],
["Synset('give_birth.v.01')", "[Synset('produce.v.01')]", 0.8],
["Synset('take.v.35')", "[Synset('sleep_together.v.01')]",
0.8888888888888888]]], ["obese", "JJ", [["Synset('corpulent.s.01')",
"{}", 0]]], ["nutritionist", "JJ", []], ["try", "NN",
[["Synset('attempt.n.01')", "[Synset('activity.n.01')]",
0.9230769230769231]]], ["and", "CC", []], ["tell", "VB",
[["Synset('state.v.01')", "[Synset('express.v.02')]", 0.4],
["Synset('tell.v.02')", "[Synset('inform.v.01')]", 0.8888888888888888],
["Synset('tell.v.03')", "[Synset('inform.v.01')]", 0.8888888888888888],
["Synset('order.v.01')", "[Synset('request.v.02')]", 0.9333333333333333],
["Synset('tell.v.05')", "[Synset('guess.v.04')]", 0.8571428571428571],
["Synset('assure.v.02')", "[Synset('affirm.v.02')]", 0.8888888888888888],
["Synset('tell.v.07')", "[Synset('inform.v.03')]", 0.9090909090909091],
["Synset('distinguish.v.01')", "[Synset('identify.v.01')]", 0.8]]],
["me", "PRP", []], ["that", "IN", []], ["my", "PRP$", []], ["diet", "NN",
[["Synset('diet.n.01')", "[Synset('fare.n.04')]", 0.9230769230769231],
["Synset('diet.n.02')", "[Synset('legislature.n.01')]",
0.9333333333333333], ["Synset('diet.n.03')", "[Synset('fare.n.04')]",
0.9230769230769231], ["Synset('diet.n.04')", "[Synset('fast.n.01')]",
0.9523809523809523]]], ["is", "VBZ", [["Synset('be.v.01')", "{}", 0],
["Synset('be.v.02')", "{}", 0], ["Synset('be.v.03')", "{}", 0],
["Synset('exist.v.01')", "{}", 0], ["Synset('be.v.05')", "{}", 0],
["Synset('equal.v.01')", "{}", 0], ["Synset('constitute.v.01')", "{}",
0], ["Synset('be.v.08')", "{}", 0], ["Synset('embody.v.02')",
"[Synset('typify.v.02')]", 0.9473684210526315], ["Synset('be.v.10')",
"[Synset('take.v.02')]", 0.8], ["Synset('be.v.11')", "{}", 0],
["Synset('be.v.12')", "[Synset('stay.v.01')]", 0.8],
["Synset('cost.v.01')", "[Synset('be.v.01')]", 0.4]]], ["wrong", "JJ",

[["Synset('incorrect.a.01')", "{}", 0], ["Synset('wrong.a.02')", "{}",
0], ["Synset('improper.s.03')", "{}", 0], ["Synset('amiss.s.01')", "{}",
0], ["Synset('wrong.a.05')", "{}", 0], ["Synset('wrong.s.06')", "{}", 0],
["Synset('wrong.s.07')", "{}", 0], ["Synset('ill-timed.s.01')", "{}", 0],
["Synset('faulty.s.02')", "{}", 0]]], ["and", "CC", []], ["I", "PRP",
[]], ["'m", "VBP", []], ["like", "IN", []], ["hey", "NN", []], ["just",
"RB", [["Synset('merely.r.01')", "{}", 0], ["Synset('precisely.r.01')",
"{}", 0], ["Synset('just.r.03')", "{}", 0], ["Synset('just.r.04')", "{}",
0], ["Synset('barely.r.01')", "{}", 0], ["Synset('just.r.06')", "{}",
0]]], ["because", "IN", []], ["you", "PRP", []], ["'ve", "VBP", []],
["got", "VBN", [["Synset('get.v.01')", "{}", 0],
["Synset('become.v.01')", "[Synset('change_state.v.01')]", 0.8],
["Synset('get.v.03')", "[Synset('make.v.02')]", 0.8],
["Synset('receive.v.02')", "[Synset('change.v.02')]", 0.4],
["Synset('arrive.v.01')", "{}", 0], ["Synset('bring.v.04')",
"[Synset('transmit.v.04')]", 0.8], ["Synset('experience.v.03')",
"[Synset('undergo.v.01')]", 0.8], ["Synset('pay_back.v.02')",
"[Synset('get_even.v.02')]", 0.8571428571428571], ["Synset('have.v.17')",
"[Synset('score.v.01')]", 0.8], ["Synset('induce.v.02')", "{}", 0],
["Synset('get.v.11')", "[Synset('seize.v.01')]", 0.8],
["Synset('grow.v.08')", "[Synset('change.v.02')]", 0.4],
["Synset('contract.v.04')", "[Synset('sicken.v.02')]",
0.8888888888888888], ["Synset('get.v.14')",
"[Synset('communicate.v.02')]", 0.8571428571428571],
["Synset('make.v.02')", "[Synset('change.v.01')]", 0.4],
["Synset('drive.v.11')", "[Synset('mean.v.01')]", 0.9411764705882353],
["Synset('catch.v.18')", "[Synset('understand.v.01')]", 0.4],
["Synset('catch.v.07')", "[Synset('attract.v.01')]", 0.8571428571428571],
["Synset('get.v.19')", "[Synset('hit.v.03')]", 0.8],
["Synset('get.v.20')", "{}", 0], ["Synset('get.v.21')",
"[Synset('get.v.01')]", 0.4], ["Synset('get.v.22')",
"[Synset('buy.v.01')]", 0.8], ["Synset('catch.v.21')",
"[Synset('hear.v.01')]", 0.8], ["Synset('catch.v.22')",
"[Synset('hurt.v.06')]", 0.8], ["Synset('get.v.25')", "{}", 0],
["Synset('scram.v.01')", "[Synset('leave.v.01')]", 0.4],

["Synset('get.v.27')", "[Synset('catch.v.09')]", 0.8571428571428571],
["Synset('get.v.28')", "[Synset('annoy.v.01')]", 0.8],
["Synset('get.v.29')", "[Synset('touch.v.03')]", 0.8],
["Synset('catch.v.24')", "[Synset('reproduce.v.03')]",
0.8888888888888888], ["Synset('draw.v.15')", "[Synset('effect.v.01')]",
0.8571428571428571], ["Synset('get.v.32')", "[Synset('destroy.v.02')]",
0.4], ["Synset('perplex.v.01')", "[Synset('confuse.v.02')]", 0.8],
["Synset('get_down.v.07')", "{}", 0], ["Synset('suffer.v.02')",
"[Synset('experience.v.03')]", 0.8571428571428571],
["Synset('beget.v.01')", "[Synset('make.v.03')]", 0.4]]], ["a", "DT",
[]], ["bit", "NN", [["Synset('spot.n.10')",
"[Synset('small_indefinite_quantity.n.01')]", 0.9090909090909091],
["Synset('bit.n.02')", "[Synset('fragment.n.01')]", 0.9090909090909091],
["Synset('moment.n.02')", "[Synset('time.n.03')]", 0.9230769230769231],
["Synset('piece.n.05')", "[Synset('case.n.01')]", 0.9230769230769231],
["Synset('bit.n.05')", "[Synset('stable_gear.n.01')]",
0.9473684210526315], ["Synset('bit.n.06')",
"[Synset('unit_of_measurement.n.01')]", 0.9090909090909091],
["Synset('morsel.n.02')", "[Synset('taste.n.05')]", 0.9230769230769231],
["Synset('snatch.n.01')", "[Synset('fragment.n.03')]",
0.9411764705882353], ["Synset('act.n.04')",
"[Synset('performance.n.01')]", 0.9333333333333333],
["Synset('bit.n.10')", "[Synset('part.n.02')]", 0.8888888888888888],
["Synset('bit.n.11')", "[Synset('cutting_implement.n.01')]",
0.9473684210526315]]], ["of", "IN", []], ["paper", "NN",
[["Synset('paper.n.01')", "[Synset('material.n.01')]",
0.9230769230769231], ["Synset('composition.n.08')",
"[Synset('essay.n.01')]", 0.9230769230769231],
["Synset('newspaper.n.01')", "[Synset('press.n.02')]",
0.9473684210526315], ["Synset('paper.n.04')", "[Synset('medium.n.01')]",
0.9333333333333333], ["Synset('paper.n.05')", "[Synset('article.n.01')]",
0.9411764705882353], ["Synset('newspaper.n.02')",
"[Synset('publisher.n.01')]", 0.9473684210526315],
["Synset('newspaper.n.03')", "[Synset('product.n.02')]",
0.9333333333333333]]], ["does", "VBZ", [["Synset('make.v.01')", "{}", 0],

["Synset('perform.v.01')", "{}", 0], ["Synset('do.v.03')",
"[Synset('carry_through.v.01')]", 0.8888888888888888],
["Synset('do.v.04')", "[Synset('proceed.v.04')]", 0.8],
["Synset('cause.v.01')", "[Synset('make.v.03')]", 0.4],
["Synset('practice.v.01')", "{}", 0], ["Synset('suffice.v.01')",
"[Synset('satisfy.v.01')]", 0.8888888888888888], ["Synset('do.v.08')",
"[Synset('create.v.05')]", 0.8], ["Synset('act.v.02')", "{}", 0],
["Synset('serve.v.09')", "[Synset('spend.v.01')]", 0.4],
["Synset('do.v.11')", "{}", 0], ["Synset('dress.v.16')",
"[Synset('groom.v.03')]", 0.8888888888888888], ["Synset('do.v.13')",
"[Synset('travel.v.01')]", 0.4]]], ["n't", "RB", []], ["mean", "VB",
[["Synset('mean.v.01')", "[Synset('convey.v.01')]", 0.9333333333333333],
["Synset('entail.v.01')", "[Synset('necessitate.v.02')]",
0.9090909090909091], ["Synset('mean.v.03')", "{}", 0],
["Synset('intend.v.01')", "{}", 0], ["Synset('mean.v.05')", "{}", 0],
["Synset('think_of.v.04')", "[Synset('associate.v.01')]", 0.8],
["Synset('mean.v.07')", "[Synset('intend.v.02')]", 0.8571428571428571]]],
["I", "PRP", []], ["'m", "VBP", []], ["going", "VBG",
[["Synset('travel.v.01')", "{}", 0], ["Synset('go.v.02')",
"[Synset('act.v.01')]", 0.4], ["Synset('go.v.03')",
"[Synset('exit.v.01')]", 0.8], ["Synset('become.v.01')",
"[Synset('change_state.v.01')]", 0.8], ["Synset('go.v.05')", "{}", 0],
["Synset('run.v.05')", "[Synset('be.v.01')]", 0.4],
["Synset('run.v.03')", "[Synset('be.v.03')]", 0.4],
["Synset('proceed.v.04')", "[Synset('happen.v.01')]", 0.4],
["Synset('go.v.09')", "[Synset('disappear.v.01')]", 0.4],
["Synset('go.v.10')", "[Synset('be.v.01')]", 0.4],
["Synset('sound.v.02')", "[Synset('cause_to_be_perceived.v.01')]", 0.4],
["Synset('function.v.01')", "{}", 0], ["Synset('run_low.v.01')",
"[Synset('end.v.01')]", 0.4], ["Synset('move.v.13')",
"[Synset('change.v.02')]", 0.4], ["Synset('survive.v.01')", "{}", 0],
["Synset('go.v.16')", "{}", 0], ["Synset('die.v.01')",
"[Synset('change_state.v.01')]", 0.8], ["Synset('belong.v.03')",
"[Synset('be.v.03')]", 0.4], ["Synset('go.v.19')",
"[Synset('compare.v.02')]", 0.8], ["Synset('start.v.09')", "{}", 0],

["Synset('move.v.15')", "{}", 0], ["Synset('go.v.22')", "{}", 0],
["Synset('go.v.23')", "{}", 0], ["Synset('blend.v.02')",
"[Synset('harmonize.v.01')]", 0.8571428571428571], ["Synset('go.v.25')",
"[Synset('be.v.03')]", 0.4], ["Synset('fit.v.02')",
"[Synset('fit.v.07')]", 0.8], ["Synset('rifle.v.02')",
"[Synset('search.v.04')]", 0.8], ["Synset('go.v.28')", "{}", 0],
["Synset('plump.v.04')", "[Synset('choose.v.01')]", 0.8],
["Synset('fail.v.04')", "[Synset('change.v.02')]", 0.4]]], ["to", "TO",
[]], ["take", "VB", [["Synset('take.v.01')", "[Synset('act.v.01')]",
0.4], ["Synset('take.v.02')", "[Synset('use.v.03')]", 0.4],
["Synset('lead.v.01')", "{}", 0], ["Synset('take.v.04')", "{}", 0],
["Synset('assume.v.03')", "[Synset('change.v.02')]", 0.4],
["Synset('take.v.06')", "[Synset('interpret.v.01')]", 0.8],
["Synset('bring.v.01')", "[Synset('transport.v.02')]", 0.8],
["Synset('take.v.08')", "{}", 0], ["Synset('take.v.09')",
"[Synset('use.v.01')]", 0.4], ["Synset('choose.v.01')",
"[Synset('decide.v.01')]", 0.4], ["Synset('accept.v.02')",
"[Synset('get.v.01')]", 0.4], ["Synset('fill.v.04')",
"[Synset('work.v.02')]", 0.4], ["Synset('consider.v.03')",
"[Synset('think_about.v.01')]", 0.8], ["Synset('necessitate.v.01')",
"{}", 0], ["Synset('take.v.15')", "[Synset('experience.v.03')]",
0.8571428571428571], ["Synset('film.v.01')", "[Synset('record.v.01')]",
0.8888888888888888], ["Synset('remove.v.01')", "{}", 0],
["Synset('consume.v.02')", "{}", 0], ["Synset('take.v.19')",
"[Synset('undergo.v.01')]", 0.8], ["Synset('take.v.20')", "{}", 0],
["Synset('take.v.21')", "{}", 0], ["Synset('assume.v.05')",
"[Synset('move.v.03')]", 0.4], ["Synset('accept.v.05')",
"[Synset('accept.v.02')]", 0.8], ["Synset('take.v.24')",
"[Synset('receive.v.02')]", 0.8], ["Synset('learn.v.04')", "{}", 0],
["Synset('claim.v.05')", "[Synset('necessitate.v.01')]", 0.4],
["Synset('take.v.27')", "[Synset('head.v.01')]", 0.8571428571428571],
["Synset('aim.v.01')", "[Synset('position.v.01')]", 0.8571428571428571],
["Synset('take.v.29')", "[Synset('become.v.01')]", 0.8571428571428571],
["Synset('carry.v.02')", "[Synset('have.v.02')]", 0.4],
["Synset('lease.v.04')", "[Synset('get.v.01')]", 0.4],

["Synset('subscribe.v.05')", "[Synset('buy.v.01')]", 0.8],
["Synset('take.v.33')", "[Synset('buy.v.01')]", 0.8],
["Synset('take.v.34')", "{}", 0], ["Synset('take.v.35')",
"[Synset('sleep_together.v.01')]", 0.8888888888888888],
["Synset('claim.v.04')", "[Synset('affirm.v.02')]", 0.8888888888888888],
["Synset('accept.v.08')", "[Synset('be.v.01')]", 0.4],
["Synset('contain.v.05')", "[Synset('be.v.01')]", 0.4],
["Synset('take.v.39')", "{}", 0], ["Synset('drive.v.16')",
"[Synset('traverse.v.01')]", 0.8571428571428571], ["Synset('take.v.41')",
"[Synset('win.v.01')]", 0.4], ["Synset('contract.v.04')",
"[Synset('sicken.v.02')]", 0.8888888888888888]]], ["your", "PRP$", []],
["advice", "NN", [["Synset('advice.n.01')", "[Synset('proposal.n.01')]",
0.9090909090909091]]], ["I", "PRP", []], ["want", "VBP",
[["Synset('desire.v.01')", "{}", 0], ["Synset('want.v.02')",
"[Synset('be.v.01')]", 0.4], ["Synset('want.v.03')",
"[Synset('search.v.01')]", 0.4], ["Synset('want.v.04')",
"[Synset('demand.v.01')]", 0.9230769230769231], ["Synset('want.v.05')",
"[Synset('miss.v.06')]", 0.4]]], ["to", "TO", []], ["see", "VB",
[["Synset('see.v.01')", "[Synset('perceive.v.01')]", 0.4],
["Synset('understand.v.02')", "{}", 0], ["Synset('witness.v.02')",
"[Synset('experience.v.01')]", 0.8571428571428571],
["Synset('visualize.v.01')", "[Synset('imagine.v.01')]",
0.8571428571428571], ["Synset('see.v.05')", "[Synset('think.v.01')]",
0.8571428571428571], ["Synset('learn.v.02')", "{}", 0],
["Synset('watch.v.03')", "[Synset('watch.v.01')]", 0.4],
["Synset('meet.v.01')", "{}", 0], ["Synset('determine.v.08')", "{}", 0],
["Synset('see.v.10')", "[Synset('verify.v.01')]", 0.8],
["Synset('see.v.11')", "[Synset('visit.v.03')]", 0.8],
["Synset('see.v.12')", "[Synset('visit.v.03')]", 0.8],
["Synset('visit.v.01')", "[Synset('tour.v.01')]", 0.8571428571428571],
["Synset('attend.v.02')", "[Synset('care.v.02')]", 0.8571428571428571],
["Synset('see.v.15')", "[Synset('receive.v.05')]", 0.4],
["Synset('go_steady.v.01')", "[Synset('consort.v.01')]",
0.8571428571428571], ["Synset('see.v.17')", "[Synset('see.v.01')]", 0.8],
["Synset('see.v.18')", "[Synset('consider.v.05')]", 0.9333333333333333],

["Synset('see.v.19')", "[Synset('detect.v.01')]", 0.8571428571428571], ["Synset('examine.v.02')", "{}", 0], ["Synset('experience.v.01')", "[Synset('undergo.v.01')]", 0.8], ["Synset('see.v.22')", "[Synset('accompany.v.02')]", 0.8], ["Synset('see.v.23')", "[Synset('bet.v.02')]", 0.8888888888888888], ["Synset('interpret.v.01')", "[Synset('understand.v.01')]", 0.4]]], ["the", "DT", []], ["results", "NNS", [["Synset('consequence.n.01')", "[Synset('phenomenon.n.01')]", 0.8888888888888888], ["Synset('solution.n.02')", "[Synset('statement.n.01')]", 0.9090909090909091], ["Synset('result.n.03')", "[Synset('ending.n.04')]", 0.9230769230769231], ["Synset('resultant_role.n.01')", "[Synset('semantic_role.n.01')]", 0.9333333333333333]]]

Vectors

{"word": "I", "pos": "PRP", "domain": "Compliance", "time": ["I", "00:00:00.000"], "vector": ["1.01219512195", "1.0", "1.01219512195"]}, {"word": "'ve", "pos": "VBP", "domain": "", "time": ["'ve", "00:00:02.005"], "vector": ["0.0", "1.0", "0.0"]}, {"word": "had", "pos": "VBN", "domain": "", "time": ["had", "00:00:03.333"], "vector": ["0.0", "0.111111111111", "0.0"]}, {"word": "obese", "pos": "JJ", "domain": "Body_description_holistic", "time": ["obese", "00:00:03.075"], "vector": ["0.0243902439024", "1", "0.0487804878049"]}, {"word": "nutritionist", "pos": "JJ", "domain": "", "time": ["nutritionist", "00:00:04.000"], "vector": ["0.0731707317073", "1.0", "0.0"]}, {"word": "try", "pos": "NN", "domain": "Isolated_places", "time": ["try", "00:00:04.166"], "vector": ["0.0243902439024", "0.0769230769231", "0.0487804878049"]}, {"word": "and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, {"word": "tell", "pos": "VB", "domain": "Omen", "time": ["tell", "00:00:04.375"], "vector": ["0.0243902439024", "0.0666666666667", "0.0243902439024"]}, {"word": "me", "pos": "PRP", "domain": "", "time": ["me", "00:00:02.085"], "vector": ["0.0", "1.0", "0.0"]}, {"word": "that", "pos": "IN", "domain": "", "time": ["that", "00:00:05.275"], "vector": ["0.0", "1.0", "0.0"]}, {"word": "my", "pos": "PRP$", "domain": "", "time": ["my", "00:00:06.833"], "vector": ["0.0", "1.0", "0.0"]}, {"word": "diet", "pos": "NN", "domain": "", "time": ["diet", "00:00:06.487"], "vector": ["0.0853658536585", "0.047619047619", "0.0"]},

{"word": "is", "pos": "VBZ", "domain": "", "time": ["is",
"00:00:06.073"], "vector": ["0.0", "0.0526315789474", "0.0"]}, {"word":
"wrong", "pos": "JJ", "domain": "Morality_evaluation", "time": ["wrong",
"00:00:06.891"], "vector": ["0.0243902439024", "1", "0.0365853658537"]},
{"word": "and", "pos": "CC", "domain": "", "time": ["and",
"00:00:07.714"], "vector": ["0.0", "1.0", "0.0"]}, {"word": "I", "pos":
"PRP", "domain": "Compliance", "time": ["I", "00:00:07.937"], "vector":
["1.01219512195", "1.0", "1.01219512195"]}, {"word": "'m", "pos": "VBP",
"domain": "", "time": ["'m", "00:00:07.161"], "vector": ["0.0", "1.0",
"0.0"]}, {"word": "like", "pos": "IN", "domain": "Similarity", "time":
["like", "00:00:07.215"], "vector": ["0.19512195122", "1.0",
"0.158536585366"]}, {"word": "hey", "pos": "NN", "domain": "", "time":
["hey", "00:00:05.069"], "vector": ["0.0853658536585", "1.0", "0.0"]},
{"word": "just", "pos": "RB", "domain": "", "time": ["just",
"00:00:06.103"], "vector": ["0.0", "1", "0.0"]}, {"word": "because",
"pos": "IN", "domain": "", "time": ["because", "00:00:06.448"], "vector":
["0.0", "1.0", "0.0"]}, {"word": "you", "pos": "PRP", "domain": "",
"time": ["you", "00:00:06.620"], "vector": ["0.0", "1.0", "0.0"]},
{"word": "'ve", "pos": "VBP", "domain": "", "time": ["'ve",
"00:00:06.724"], "vector": ["0.0", "1.0", "0.0"]}, {"word": "got", "pos":
"VBN", "domain": "Wearing", "time": ["got", "00:00:06.793"], "vector":
["0.0731707317073", "0.0588235294118", "0.0731707317073"]}, {"word": "a",
"pos": "DT", "domain": "", "time": ["a", "00:00:06.842"], "vector":
["0.0", "1.0", "0.0"]}, {"word": "bit", "pos": "NN", "domain":
"Quantified_mass", "time": ["bit", "00:00:06.879"], "vector":
["0.0243902439024", "0.0526315789474", "0.280487804878"]}, {"word": "of",
"pos": "IN", "domain": "", "time": ["of", "00:00:06.908"], "vector":
["0.0", "1.0", "0.0"]}, {"word": "paper", "pos": "NN", "domain": "Text",
"time": ["paper", "00:00:06.012"], "vector": ["0.0609756097561",
"0.0526315789474", "0.0975609756098"]}, {"word": "does", "pos": "VBZ",
"domain": "", "time": ["does", "00:00:07.068"], "vector": ["0.0",
"0.111111111111", "0.0"]}, {"word": "n't", "pos": "RB", "domain": "",
"time": ["n't", "00:00:08.002"], "vector": ["0.0", "1.0", "0.0"]},
{"word": "mean", "pos": "VB", "domain": "Linguistic_meaning", "time":
["mean", "00:00:08.046"], "vector": ["0.134146341463", "0.0666666666667",

"0.0975609756098"]}, {"word": "I", "pos": "PRP", "domain": "Compliance", "time": ["I", "00:00:08.616"], "vector": ["1.01219512195", "1.0", "1.01219512195"]}, {"word": "'m", "pos": "VBP", "domain": "", "time": ["'m", "00:00:08.072"], "vector": ["0.0", "1.0", "0.0"]}, {"word": "going", "pos": "VBG", "domain": "Getting_underway", "time": ["going", "00:00:08.794"], "vector": ["0.0853658536585", "0.142857142857", "0.0853658536585"]}, {"word": "to", "pos": "TO", "domain": "", "time": ["to", "00:00:08.085"], "vector": ["0.0", "1.0", "0.0"]}, {"word": "take", "pos": "VB", "domain": "Getting_vehicle_underway", "time": ["take", "00:00:08.893"], "vector": ["0.0731707317073", "0.111111111111", "0.0731707317073"]}, {"word": "your", "pos": "PRP$", "domain": "", "time": ["your", "00:00:07.068"], "vector": ["0.0", "1.0", "0.0"]}, {"word": "advice", "pos": "NN", "domain": "", "time": ["advice", "00:00:13.002"], "vector": ["0.0731707317073", "0.0909090909091", "0.0"]}, {"word": "I", "pos": "PRP", "domain": "Compliance", "time": ["I", "00:00:14.008"], "vector": ["1.01219512195", "1.0", "1.01219512195"]}, {"word": "want", "pos": "VBP", "domain": "Possession", "time": ["want", "00:00:15.069"], "vector": ["0.0853658536585", "0.0769230769231", "0.0853658536585"]}, {"word": "to", "pos": "TO", "domain": "", "time": ["to", "00:00:16.224"], "vector": ["0.0", "1.0", "0.0"]}, {"word": "see", "pos": "VB", "domain": "Request", "time": ["see", "00:00:16.058"], "vector": ["0.121951219512", "0.0666666666667", "0.121951219512"]}, {"word": "the", "pos": "DT", "domain": "", "time": ["the", "00:00:16.834"], "vector": ["0.0", "1.0", "0.0"]}, {"word": "results", "pos": "NNS", "domain": "", "time": ["results", "00:00:17.025"], "vector": ["0.121951219512", "0.0666666666667", "0.0"]}

K-Nearest Neighbor

{"current": {"word": "I", "pos": "PRP", "domain": "Compliance", "time": ["I", "00:00:00.000"], "vector": ["1.01219512195", "1.0", "1.01219512195"]}, "neighbor": {"word": "I", "pos": "PRP", "domain": "Compliance", "time": ["I", "00:00:07.937"], "vector": ["1.01219512195", "1.0", "1.01219512195"]}, "distance": 0.0}, {"current": {"word": "'ve", "pos": "VBP", "domain": "", "time": ["'ve", "00:00:02.005"], "vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word": "and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word": "had", "pos": "VBN",

"domain": "", "time": ["had", "00:00:03.333"], "vector": ["0.0",
"0.111111111111", "0.0"]}, "neighbor": {"word": "does", "pos": "VBZ",
"domain": "", "time": ["does", "00:00:07.068"], "vector": ["0.0",
"0.111111111111", "0.0"]}, "distance": 0.0}, {"current": {"word":
"obese", "pos": "JJ", "domain": "Body_description_holistic", "time":
["obese", "00:00:03.075"], "vector": ["0.0243902439024", "1",
"0.0487804878049"]}, "neighbor": {"word": "obese", "pos": "JJ", "domain":
"Body_description_holistic", "time": ["obese", "00:08:49.201"], "vector":
["0.0243902439024", "1", "0.0487804878049"]}, "distance": 0.0},
{"current": {"word": "nutritionist", "pos": "JJ", "domain": "", "time":
["nutritionist", "00:00:04.000"], "vector": ["0.0731707317073", "1.0",
"0.0"]}, "neighbor": {"word": "nutritionist", "pos": "JJ", "domain": "",
"time": ["nutritionist", "00:02:32.011"], "vector": ["0.0731707317073",
"1.0", "0.0"]}, "distance": 0.0}, {"current": {"word": "try", "pos":
"NN", "domain": "Isolated_places", "time": ["try", "00:00:04.166"],
"vector": ["0.0243902439024", "0.0769230769231", "0.0487804878049"]},
"neighbor": {"word": "experience", "pos": "NN", "domain": "Expertise",
"time": ["experience", "00:04:42.065"], "vector": ["0.0243902439024",
"0.0769230769231", "0.0487804878049"]}, "distance": 0.0}, {"current":
{"word": "and", "pos": "CC", "domain": "", "time": ["and",
"00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word":
"and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"],
"vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word":
"tell", "pos": "VB", "domain": "Omen", "time": ["tell", "00:00:04.375"],
"vector": ["0.0243902439024", "0.0666666666667", "0.0243902439024"]},
"neighbor": {"word": "fact", "pos": "NN", "domain": "Artifact", "time":
["fact", "00:00:40.521"], "vector": ["0.0243902439024",
"0.0666666666667", "0.0243902439024"]}, "distance": 0.0}, {"current":
{"word": "me", "pos": "PRP", "domain": "", "time": ["me",
"00:00:02.085"], "vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word":
"and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"],
"vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word":
"that", "pos": "IN", "domain": "", "time": ["that", "00:00:05.275"],
"vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word": "and", "pos":
"CC", "domain": "", "time": ["and", "00:00:04.285"], "vector": ["0.0",

"1.0", "0.0"]}, "distance": 0.0}, {"current": {"word": "my", "pos": "PRP$", "domain": "", "time": ["my", "00:00:06.833"], "vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word": "and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word": "diet", "pos": "NN", "domain": "", "time": ["diet", "00:00:06.487"], "vector": ["0.0853658536585", "0.047619047619", "0.0"]}, "neighbor": {"word": "diet", "pos": "NN", "domain": "", "time": ["diet", "00:08:50.093"], "vector": ["0.0853658536585", "0.047619047619", "0.0"]}, "distance": 0.0}, {"current": {"word": "is", "pos": "VBZ", "domain": "", "time": ["is", "00:00:06.073"], "vector": ["0.0", "0.0526315789474", "0.0"]}, "neighbor": {"word": "been", "pos": "VBN", "domain": "", "time": ["been", "00:00:28.393"], "vector": ["0.0", "0.0526315789474", "0.0"]}, "distance": 0.0}, {"current": {"word": "wrong", "pos": "JJ", "domain": "Morality_evaluation", "time": ["wrong", "00:00:06.891"], "vector": ["0.0243902439024", "1", "0.0365853658537"]}, "neighbor": {"word": "wrong", "pos": "JJ", "domain": "Morality_evaluation", "time": ["wrong", "00:08:52.085"], "vector": ["0.0243902439024", "1", "0.0365853658537"]}, "distance": 0.0}, {"current": {"word": "and", "pos": "CC", "domain": "", "time": ["and", "00:00:07.714"], "vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word": "and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word": "I", "pos": "PRP", "domain": "Compliance", "time": ["I", "00:00:07.937"], "vector": ["1.01219512195", "1.0", "1.01219512195"]}, "neighbor": {"word": "I", "pos": "PRP", "domain": "Compliance", "time": ["I", "00:00:07.937"], "vector": ["1.01219512195", "1.0", "1.01219512195"]}, "distance": 0.0}, {"current": {"word": "'m", "pos": "VBP", "domain": "", "time": ["'m", "00:00:07.161"], "vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word": "and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word": "like", "pos": "IN", "domain": "Similarity", "time": ["like", "00:00:07.215"], "vector": ["0.19512195122", "1.0", "0.158536585366"]}, "neighbor": {"word": "like", "pos": "IN", "domain": "Similarity", "time": ["like", "00:00:23.855"], "vector": ["0.19512195122", "1.0", "0.158536585366"]}, "distance": 0.0},

{"current": {"word": "hey", "pos": "NN", "domain": "", "time": ["hey", "00:00:05.069"], "vector": ["0.0853658536585", "1.0", "0.0"]}, "neighbor": {"word": "hey", "pos": "NN", "domain": "", "time": ["hey", "00:00:17.648"], "vector": ["0.0853658536585", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word": "just", "pos": "RB", "domain": "", "time": ["just", "00:00:06.103"], "vector": ["0.0", "1", "0.0"]}, "neighbor": {"word": "and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word": "because", "pos": "IN", "domain": "", "time": ["because", "00:00:06.448"], "vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word": "and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word": "you", "pos": "PRP", "domain": "", "time": ["you", "00:00:06.620"], "vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word": "and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word": "'ve", "pos": "VBP", "domain": "", "time": ["'ve", "00:00:06.724"], "vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word": "and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word": "got", "pos": "VBN", "domain": "Wearing", "time": ["got", "00:00:06.793"], "vector": ["0.0731707317073", "0.0588235294118", "0.0731707317073"]}, "neighbor": {"word": "got", "pos": "VBD", "domain": "Wearing", "time": ["got", "00:08:52.095"], "vector": ["0.0731707317073", "0.0588235294118", "0.0731707317073"]}, "distance": 0.0}, {"current": {"word": "a", "pos": "DT", "domain": "", "time": ["a", "00:00:06.842"], "vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word": "and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word": "bit", "pos": "NN", "domain": "Quantified_mass", "time": ["bit", "00:00:06.879"], "vector": ["0.0243902439024", "0.0526315789474", "0.280487804878"]}, "neighbor": {"word": "bit", "pos": "NN", "domain": "Quantified_mass", "time": ["bit", "00:08:53.426"], "vector": ["0.0243902439024", "0.0526315789474", "0.280487804878"]}, "distance": 0.0}, {"current": {"word": "of", "pos": "IN", "domain": "", "time": ["of", "00:00:06.908"], "vector": ["0.0",

"1.0", "0.0"]}, "neighbor": {"word": "and", "pos": "CC", "domain": "",
"time": ["and", "00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]},
"distance": 0.0}, {"current": {"word": "paper", "pos": "NN", "domain":
"Text", "time": ["paper", "00:00:06.012"], "vector": ["0.0609756097561",
"0.0526315789474", "0.0975609756098"]}, "neighbor": {"word": "paper",
"pos": "NN", "domain": "Text", "time": ["paper", "00:06:51.802"],
"vector": ["0.0609756097561", "0.0526315789474", "0.0975609756098"]},
"distance": 0.0}, {"current": {"word": "does", "pos": "VBZ", "domain":
"", "time": ["does", "00:00:07.068"], "vector": ["0.0", "0.111111111111",
"0.0"]}, "neighbor": {"word": "does", "pos": "VBZ", "domain": "", "time":
["does", "00:00:07.068"], "vector": ["0.0", "0.111111111111", "0.0"]},
"distance": 0.0}, {"current": {"word": "n't", "pos": "RB", "domain": "",
"time": ["n't", "00:00:08.002"], "vector": ["0.0", "1.0", "0.0"]},
"neighbor": {"word": "and", "pos": "CC", "domain": "", "time": ["and",
"00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0},
{"current": {"word": "mean", "pos": "VB", "domain": "Linguistic_meaning",
"time": ["mean", "00:00:08.046"], "vector": ["0.134146341463",
"0.0666666666667", "0.0975609756098"]}, "neighbor": {"word": "mean",
"pos": "VBP", "domain": "Linguistic_meaning", "time": ["mean",
"00:03:19.665"], "vector": ["0.134146341463", "0.0666666666667",
"0.0975609756098"]}, "distance": 0.0}, {"current": {"word": "I", "pos":
"PRP", "domain": "Compliance", "time": ["I", "00:00:08.616"], "vector":
["1.01219512195", "1.0", "1.01219512195"]}, "neighbor": {"word": "I",
"pos": "PRP", "domain": "Compliance", "time": ["I", "00:00:07.937"],
"vector": ["1.01219512195", "1.0", "1.01219512195"]}, "distance": 0.0},
{"current": {"word": "'m", "pos": "VBP", "domain": "", "time": ["'m",
"00:00:08.072"], "vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word":
"and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"],
"vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word":
"going", "pos": "VBG", "domain": "Getting_underway", "time": ["going",
"00:00:08.794"], "vector": ["0.0853658536585", "0.142857142857",
"0.0853658536585"]}, "neighbor": {"word": "going", "pos": "VBG",
"domain": "Getting_underway", "time": ["going", "00:01:25.949"],
"vector": ["0.0853658536585", "0.142857142857", "0.0853658536585"]},
"distance": 0.0}, {"current": {"word": "to", "pos": "TO", "domain": "",

"time": ["to", "00:00:08.085"], "vector": ["0.0", "1.0", "0.0"]},
"neighbor": {"word": "and", "pos": "CC", "domain": "", "time": ["and",
"00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0},
{"current": {"word": "take", "pos": "VB", "domain":
"Getting_vehicle_underway", "time": ["take", "00:00:08.893"], "vector":
["0.0731707317073", "0.111111111111", "0.0731707317073"]}, "neighbor":
{"word": "take", "pos": "VB", "domain": "Getting_vehicle_underway",
"time": ["take", "00:02:55.683"], "vector": ["0.0731707317073",
"0.111111111111", "0.0731707317073"]}, "distance": 0.0}, {"current":
{"word": "your", "pos": "PRP$", "domain": "", "time": ["your",
"00:00:07.068"], "vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word":
"and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"],
"vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word":
"advice", "pos": "NN", "domain": "", "time": ["advice", "00:00:13.002"],
"vector": ["0.0731707317073", "0.0909090909091", "0.0"]}, "neighbor":
{"word": "advice", "pos": "NN", "domain": "", "time": ["advice",
"00:04:14.618"], "vector": ["0.0731707317073", "0.0909090909091",
"0.0"]}, "distance": 0.0}, {"current": {"word": "I", "pos": "PRP",
"domain": "Compliance", "time": ["I", "00:00:14.008"], "vector":
["1.01219512195", "1.0", "1.01219512195"]}, "neighbor": {"word": "I",
"pos": "PRP", "domain": "Compliance", "time": ["I", "00:00:07.937"],
"vector": ["1.01219512195", "1.0", "1.01219512195"]}, "distance": 0.0},
{"current": {"word": "want", "pos": "VBP", "domain": "Possession",
"time": ["want", "00:00:15.069"], "vector": ["0.0853658536585",
"0.0769230769231", "0.0853658536585"]}, "neighbor": {"word": "want",
"pos": "VBP", "domain": "Possession", "time": ["want", "00:03:33.022"],
"vector": ["0.0853658536585", "0.0769230769231", "0.0853658536585"]},
"distance": 0.0}, {"current": {"word": "to", "pos": "TO", "domain": "",
"time": ["to", "00:00:16.224"], "vector": ["0.0", "1.0", "0.0"]},
"neighbor": {"word": "and", "pos": "CC", "domain": "", "time": ["and",
"00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0},
{"current": {"word": "see", "pos": "VB", "domain": "Request", "time":
["see", "00:00:16.058"], "vector": ["0.121951219512", "0.0666666666667",
"0.121951219512"]}, "neighbor": {"word": "see", "pos": "VB", "domain":
"Request", "time": ["see", "00:00:57.537"], "vector": ["0.121951219512",

"0.0666666666667", "0.121951219512"]}, "distance": 0.0}, {"current": {"word": "the", "pos": "DT", "domain": "", "time": ["the", "00:00:16.834"], "vector": ["0.0", "1.0", "0.0"]}, "neighbor": {"word": "and", "pos": "CC", "domain": "", "time": ["and", "00:00:04.285"], "vector": ["0.0", "1.0", "0.0"]}, "distance": 0.0}, {"current": {"word": "results", "pos": "NNS", "domain": "", "time": ["results", "00:00:17.025"], "vector": ["0.121951219512", "0.0666666666667", "0.0"]}, "neighbor": {"word": "results", "pos": "NNS", "domain": "", "time": ["results", "00:03:48.355"], "vector": ["0.121951219512", "0.0666666666667", "0.0"]}, "distance": 0.0}

Table 5 - K-Nearest Neighbor for Discourse 6NOSD0XK0r8

Temporal Flow				Nearest Neighbor				Distance
Time	Word	Domain	POS	Time	Word	Domain	POS	
00:00:24.200	report	Statement	NN	00:25:55.613	statement	Statement	NN	0.001626016
00:00:48.125	room	Building_subparts	NN	01:02:22.989	AT	Calendric_unit	NNP	0.003575798
00:01:05.998	talk	Discussion	NN	00:20:56.051	show	Hostile_encounter	NN	0.004436408
00:01:39.512	hang	Cause_change	VBP	00:24:40.429	trait	Natural_features	NN	0.001626016
00:02:46.008	biography	Text	NN	01:04:03.515	Island	Natural_features	NNP	0.001592357
00:03:19.089	fun	Contingency	NN	00:36:21.982	cycle	Vehicle	NN	0.001592357
00:04:39.443	guys		VBP	00:07:38.085	groups		NNS	0.003252033
00:06:46.418	corporation	Businesses	NN	00:38:38.575	friends	Personal_relationship	NNS	0.001592357
00:07:07.017	sit	Placing	VB	00:48:57.355	regret	Experiencer_focus	VB	0.003575798
00:07:31.522	sort	Hedging	RB	00:06:01.005	something	Hedging	NN	0.009756098
00:10:05.483	outrage	Emotion_directed	NN	00:56:30.448	US	Calendric_unit	NNP	0.003184713
00:11:02.431	point	Appointing	VB	00:28:24.803	public	Public_services	NN	0.005778937
00:12:49.131	hop	Cause_harm	VB	01:10:01.882	fix	Attaching	VB	0.001592357
00:12:55.032	knows		NNS	00:20:00.188	situation		NN	0.001626016
00:14:15.431	buddy	Attention_getting	NN	00:15:19.026	cousin	Kinship	NN	0.011146497
00:14:20.666	night	Calendric_unit	NN	01:33:41.949	cell	Building_subparts	NN	0.003184713
00:14:46.924	na	Size	JJ	00:21:56.629	interesting	Mental_stimulus_stimulus_focus	JJ	0.001592357
00:15:02.108	plead	Request	VB	00:47:18.639	request	Request	VB	0.01025641
00:15:19.654	lawyer	People_by_vocation	NN	01:13:16.862	girlfriend	Personal_relationship	NN	0.036016413
00:15:19.026	cousin	Kinship	NN	00:49:09.716	Terminator	Death	NNP	0.003184713

00:15:42.094	harassment	Offenses	NN	00:14:19.004	Monday	Calendric_unit	NNP	0.006369427
00:17:02.649	got	Discussion	NNS	00:14:46.924	na	Size	JJ	0.008126126
00:17:04.839	little	Degree	RB	00:14:56.047	public	Secrecy_status	JJ	0.007151595
00:17:52.420	compare	Evaluative_comparison	VB	01:30:08.421	screw	Bungling	VB	0.001592357
00:18:13.823	crap	Desirability	NN	01:20:22.008	read	Desirability	NN	0.008130081
00:18:14.558	citizen	People_by_jurisdiction	NNS	00:22:17.341	nationalist		NN	0.001592357
00:20:19.061	invites		NNS	00:14:24.036	perpetuating		VBG	0.038095238
00:20:31.014	Business	Documents	NNP	01:17:08.629	patron	Being_named	NN	0.004436408
00:20:43.002	cut	Change_operational_state	VB	00:00:17.008	turned		VBD	0.001592357
00:21:10.007	act	Theft	VBP	01:13:33.316	bring	Cause_to_start	VBP	0.001592357
00:21:10.406	behave	Conduct	VBP	01:06:46.855	meditate	Cogitation	VBP	0.006369427
00:22:03.089	disagree	Quarreling	NN	00:01:21.999	ideological		JJ	0.001592357
00:23:11.396	shape	Cause_change	VB	01:31:02.151	smear	Filling	VB	0.003184713
00:23:22.469	fan	Body_parts	NN	00:55:41.416	market	Buildings	NN	0.001592357
00:24:09.246	fear	Experiencer_focus	VBP	00:29:11.314	set	Attack	VBN	0.001592357
00:24:18.957	cut	Leadership	NN	00:27:34.783	rule	Leadership	NN	0.001626016
00:24:40.429	trait	Natural_features	NN	00:52:06.009	infiltrate	Attack	VB	0.001592357
00:25:01.035	glad	Biological_area	NN	00:23:49.004	names	Namesake	NNS	0.001626016
00:27:06.438	study	Scrutiny	NN	00:03:46.610	puzzle	Emotion_directed	NN	0.00477707
00:27:29.065	sure	Telling	VB	00:09:32.007	feedback		RB	0.001592357
00:27:58.514	cares	Manipulation	VBZ	00:48:57.355	regret	Experiencer_focus	VB	0.002275858
00:28:43.318	fight	Hostile_encounter	NN	00:35:37.097	punch	Containers	NN	0.002275858
00:29:40.842	decent	Offenses	NN	00:21:19.036	usually	Frequency	RB	0.001592357
00:29:49.056	progressive		NN	00:14:40.348	seen		VBN	0.001626016
00:30:23.087	ratio	Leadership	NN	00:00:14.091	engineer	People_by_vocation	NN	0.007151595
00:31:08.208	evolve	Coming_to_be	VB	00:39:21.395	depend	Contingency	VB	0.001592357
00:31:27.262	defense	Defending	NN	00:21:57.745	phrase		NN	0.001592357
00:33:48.028	suspect	Suspicion	NN	01:25:24.484	keep		NN	0.001592357
00:34:53.414	buck	Containers	NN	01:10:24.541	machine	Weapon	NN	0.001592357
00:35:37.097	punch	Containers	NN	00:28:43.318	fight	Hostile_encounter	NN	0.002275858
00:36:22.148	spin	Biological_area	NN	00:31:27.262	defense	Defending	NN	0.001592357
00:36:29.617	accord	Documents	NN	00:20:10.844	News	Organization	NNP	0.001592357
00:36:54.223	tweet	Sounds	NN	00:16:07.633	employment	Being_employed	NN	0.001592357
00:39:44.546	ok	Omen	VBP	00:32:02.238	publicly	Secrecy_status	RB	0.001592357
00:41:33.357	plenty	Sufficiency	NN	00:41:59.084	require	Needing	VBP	0.001592357

00:41:40.099	host	Aggregate	NN	00:06:50.052	population	Aggregate	NN	0.004761724
00:41:52.084	allow	Statement	VB	00:00:24.200	report	Statement	NN	0.01025641
00:41:59.084	require	Needing	VBP	00:41:33.357	plenty	Sufficiency	NN	0.001592357
00:42:08.000	hole	Medical_conditions	NN	00:13:51.039	band	Abandonment	NN	0.001592357
00:42:21.034	training		VBG	00:18:50.643	made		VBD	0.001626016
00:42:55.006	correct	Prison	NN	00:01:09.811	glad	Emotions_by_stimulus	JJ	0.001592357
00:43:15.416	gas	Accoutrements	NN	01:16:21.873	owner	Intoxicants	NN	0.002275858
00:43:29.008	move	Intentionally_act	NN	00:30:55.395	feelings	Feeling	NNS	0.001626016
00:43:41.456	na	Being_named	NNS	00:03:42.427	got	Wearing	VBD	0.00816666
00:47:05.083	dish	Experiencer_obj	VB	00:24:40.429	trait	Natural_features	NN	0.00477707
00:47:18.639	request	Request	VB	00:16:13.243	treat	Request	VBP	0.003252033
00:47:56.931	function	Contingency	NN	01:12:40.073	challenge	Competition	NN	0.001592357
00:48:11.853	throw	Weapon	NN	00:26:38.135	treatment	Abusing	NN	0.002275858
00:48:36.983	Super	Leadership	NNP	00:27:34.783	rule	Leadership	NN	0.004140787
00:48:41.624	courage	Emotion_directed	NN	01:07:08.830	summer	Calendric_unit	NN	0.003184713
00:48:57.355	regret	Experiencer_focus	VB	00:27:58.514	cares	Manipulation	VBZ	0.002275858
00:49:02.092	need	Needing	NN	00:01:04.769	story	Individual_history	NN	0.001626016
00:49:09.716	Terminator	Death	NNP	00:18:14.558	citizen	People_by_jurisdiction	NNS	0.001592357
00:50:50.011	average		NN	00:12:22.046	called		VBN	0.001626016
00:50:55.001	train	Hindering	VB	00:20:03.245	repeated	Event_instance	VBN	0.001592357
00:50:55.808	kid	Attention_getting	NN	01:22:46.480	sheep	Animals	NN	0.008198302
00:52:06.009	infiltrate	Attack	VB	00:24:40.429	trait	Natural_features	NN	0.001592357
00:52:18.039	tricky	Difficulty	JJ	01:15:17.425	single	Personal_relationship	JJ	0.001592357
00:52:34.981	field	Locale_by_use	NN	00:31:23.616	ER	Medical_conditions	NNP	0.001592357
00:52:34.255	leans	Grooming	VBZ	00:03:57.465	apply	Using	VB	0.001592357
00:52:37.829	libertarian		NN	00:03:25.716	biology		NN	0.003252033
00:53:25.018	Jersey		NNP	01:14:13.423	grades		NNS	0.001544402
00:55:03.609	elsewhere	Locative_relation	RB	00:17:39.025	San	Expertise	NNP	0.001592357
00:56:30.448	US	Calendric_unit	NNP	00:10:05.483	outrage	Emotion_directed	NN	0.003184713
00:57:19.000	load	Filling	VBZ	00:52:06.009	infiltrate	Attack	VB	0.001592357
00:57:30.228	drama	Text	NN	00:21:41.308	hate	Buildings	NN	0.002275858
00:57:36.638	cool	Containers	NN	00:43:55.322	la	Commemorative	NN	0.001592357
00:58:57.043	lists		VBZ	00:18:50.643	made		VBD	0.001626016
00:59:00.001	bill	Body_parts	NN	00:38:07.551	Bill	Body_parts	NNP	0.001626016
00:59:13.435	foundation	Body_decoration	NN	00:16:45.448	claim	Judgment_communication	NN	0.002275858

00:59:14.007	education	Education_teaching	NN	01:14:11.003	professor	Education_teaching	NN	0.003478261
00:59:44.455	ability	Capability	NN	00:07:07.017	sit	Placing	VB	0.00477707
01:00:24.652	second	Measure_duration	NN	00:23:22.662	Trump	Sounds	NNP	0.001592357
01:01:17.386	role	Containers	NN	01:10:17.583	weapon	Weapon	NN	0.001592357
01:02:27.774	arms	Weapon	NNS	01:00:24.652	second	Measure_duration	NN	0.001592357
01:03:20.011	drug	Leadership	NN	00:21:52.562	alt	Building_subparts	NN	0.005046219
01:03:28.581	increase	Change_position_on_a_scale	VB	01:11:37.705	choose	Choosing	VB	0.003184713
01:04:03.515	Island	Natural_features	NNP	00:02:46.008	biography	Text	NN	0.001592357
01:04:03.613	suck	Desirability	VBD	00:12:24.756	read	Time_vector	VBP	0.029469127
01:04:09.252	Reb	Kinship	NNP	00:02:30.013	explode	Change_position_on_a_scale	VB	0.001592357
01:04:59.583	attempt	Attempt	NN	00:10:50.001	fine	Inhibit_movement	NN	0.001592357
01:05:58.569	action	Theft	NN	00:30:49.636	heart	Part_orientational	NN	0.001592357
01:06:02.581	avenue	Roadways	NN	00:54:44.801	representative	Leadership	NN	0.001592357
01:06:46.855	meditate	Cogitation	VBP	01:21:14.779	cause	Causation	VB	0.005778937
01:07:26.269	visible	Obviousness	JJ	00:42:40.915	obvious	Obviousness	JJ	0.003252033
01:08:34.749	attack	Attack	VB	00:55:06.007	Valley	Natural_features	NNP	0.001592357
01:08:39.065	strike	Attack	NN	00:09:27.920	phone	Word_relations	NN	0.001592357
01:10:01.882	fix	Attaching	VB	00:12:49.131	hop	Cause_harm	VB	0.001592357
01:10:17.583	weapon	Weapon	NN	01:01:17.386	role	Containers	NN	0.001592357
01:10:24.541	machine	Weapon	NN	00:34:53.414	buck	Containers	NN	0.001592357
01:10:43.450	fair	Fairness_evaluation	VB	00:17:13.179	according		VBG	0.002275858
01:10:47.650	rate	Intoxicants	NN	01:08:35.367	mine	Intoxicants	NN	0.003252033
01:11:19.811	control	Control	VBP	00:03:16.769	took		VBD	0.001592357
01:11:37.705	choose	Choosing	VB	01:01:58.232	National	Military	NNP	0.002275858
01:12:09.083	agree	Documents	NN	00:31:46.000	exactly	Proportional_quantity	RB	0.005825612
01:12:40.073	challenge	Competition	NN	00:02:21.262	describe	Communicate_categorization	VB	0.001592357
01:13:16.862	girlfriend	Personal_relationship	NN	00:15:19.654	lawyer	People_by_vocation	NN	0.036016413
01:13:33.316	bring	Cause_to_start	VBP	00:21:10.007	act	Theft	VBP	0.001592357
01:13:42.380	bit	Degree	VB	00:34:06.332	come	Suitability	VB	0.006827573
01:14:06.062	kids		NNS	01:14:13.423	grades		NNS	0.001731602
01:14:08.048	say		RB	00:24:47.218	oh		IN	0.006504065
01:14:13.423	grades		NNS	00:53:25.018	Jersey		NNP	0.001544402
01:14:11.003	professor	Education_teaching	NN	00:59:14.007	education	Education_teaching	NN	0.003478261
01:14:32.493	CTO	Calendric_unit	NNP	00:01:46.428	ish	Emotion_directed	JJ	0.003184713
01:15:17.425	single	Personal_relationship	JJ	00:52:18.039	tricky	Difficulty	JJ	0.001592357

01:15:49.760	harm	Stimulus_focus	NN	01:08:35.367	mine	Intoxicants	NN	0.0065737
01:16:02.306	steam	Apply_heat	NN	01:05:56.474	income	Earnings_and_losses	NN	0.001592357
01:16:22.154	Emory	Memory	NNP	00:01:00.229	away	Time_vector	RB	0.014423159
01:16:43.226	community	Aggregate	NN	00:15:42.094	harassment	Offenses	NN	0.008003149
01:17:08.629	patron	Being_named	NN	00:20:31.014	Business	Documents	NNP	0.004436408
01:18:25.044	push	Cause_change_of_position_on_a_scale	NN	01:18:01.614	fire	Setting_back_burn	NN	0.002275858
01:18:33.045	well		IN	00:24:47.218	oh		IN	0.003252033
01:20:12.411	stop	Preventing_or_letting	NN	00:12:25.086	attack	Attack	NN	0.002275858
01:20:22.008	read	Desirability	NN	00:18:13.823	crap	Desirability	NN	0.008130081
01:20:26.947	name	Cause_emotion	VB	00:04:37.000	working		VBG	0.002275858
01:20:49.077	know		RB	01:02:42.058	know	Being_named	JJ	0.047770701
01:21:07.626	removal	Removing	NN	00:58:52.494	exchange	Discussion	NN	0.002275858
01:21:14.629	pens	Containers	NNS	00:34:53.414	buck	Containers	NN	0.003478261
01:21:14.779	cause	Causation	VB	00:27:28.170	join	Adjacency	VB	0.001592357
01:22:34.217	past	Locative_relation	IN	00:26:13.598	biological	Weapon	JJ	0.001592357
01:22:46.480	sheep	Animals	NN	01:02:29.056	monitor	Information_display	NN	0.007962925
01:22:54.834	attempt	Attempt	VB	00:51:10.838	force	Enforcing	VB	0.001592357
01:22:59.694	shadow	Omen	VBP	00:47:05.083	dish	Experiencer_obj	VB	0.006369427
01:23:38.964	Image	Physical_artworks	NNP	00:02:46.008	biography	Text	NN	0.001592357
01:24:05.165	infer	Coming_to_believe	VB	00:51:00.625	engineer	People_by_vocation	NN	0.003620955
01:24:47.206	thought	Awareness	NN	00:27:38.956	believe	Taking_sides	VBP	0.003575798
01:24:45.061	attention	Attention	NN	00:34:27.965	sides	Avoiding	NNS	0.001592357
01:25:24.484	keep		NN	00:33:48.028	suspect	Suspicion	NN	0.001592357
01:25:33.004	mind	Evoking	VB	01:08:56.330	remind	Evoking	VB	0.003252033
01:25:37.018	religion	Co-association	NN	00:05:33.523	X	Commemorative	NNP	0.001592357
01:25:42.689	house	Locale_by_use	NN	00:52:27.556	IT	People_by_origin	NNP	0.001592357
01:26:42.632	answer	Communication_response	VB	00:09:57.876	month	Measure_duration	NN	0.003620955
01:28:16.035	explore	Scrutiny	VB	00:07:51.395	start	Activity_start	VBP	0.042535243
01:28:27.806	century	Measure_duration	NN	00:43:11.065	electricity	Electricity	NN	0.001592357
01:28:45.497	recognize	Becoming_aware	VB	01:01:17.386	role	Containers	NN	0.001592357
01:29:04.098	country	Isolated_places	NN	00:43:11.065	electricity	Electricity	NN	0.001592357
01:29:47.001	change	Undergo_change	NN	00:33:48.028	suspect	Suspicion	NN	0.001592357
01:30:08.421	screw	Bungling	VB	00:17:52.420	compare	Evaluative_comparison	VB	0.001592357
01:30:57.157	trick	Quantified_mass	NN	01:04:34.299	number	Quantified_mass	NN	0.005265152
01:31:02.151	smear	Filling	VB	00:23:11.396	shape	Cause_change	VB	0.003184713

01:32:10.095	happen	Coincidence	NN	00:13:12.337	important	Importance	JJ	0.001626016
01:32:32.564	feeling	Sensation	NN	00:37:50.414	intention	Purpose	NN	0.001592357
01:33:00.597	fascinating	Stimulus_focus	JJ	00:24:15.402	scary	Stimulus_focus	JJ	0.001626016
01:33:20.034	come	Communication_response	NN	00:01:02.478	saw	Cause_to_move_in_place	VBD	0.004551716
01:33:23.112	thank	Judgment_direct_address	VB	00:12:11.669	ton	Word_relations	NN	0.001626016
01:33:41.949	cell	Building_subparts	NN	00:51:29.225	salon	Building_subparts	NN	0.001626016
01:33:46.322	skill	Expertise	NN	00:28:05.098	conversation	Chatting	NN	0.003575798
01:34:08.320	message	Statement	NN	01:23:47.102	explain	Statement	VB	0.031912165
01:35:11.058	parent	Kinship	NN	01:19:02.705	interference	Hindering	NN	0.001592357
01:35:24.120	Tuesday	Calendric_unit	NNP	00:14:19.004	Monday	Calendric_unit	NNP	0.001626016

8 Appendices

8.1 Corpus Python Code

```
        import argparse
import codecs
import json
import os
import sys
import math
import time
import numpy as np
import unicodedata
from moviepy.video.io.VideoFileClip import VideoFileClip
from scipy.spatial import KDTree
import scipy
import matplotlib.pyplot as plt
from matplotlib.ticker import LinearLocator, FormatStrFormatter
import nltk
import nltk.probability
from nltk.corpus import stopwords
from nltk.corpus import framenet as fn
from nltk.corpus import wordnet as wn
import string
import youtube_dl
from corpus import youtube as yttc
import pygame
from scipy.spatial import convex_hull_plot_2d
from OpenGL.GL import *
from pgu import gui
import re
import random
from six.moves import input

reflections = {
    "i am": "you are",
    "i was": "you were",
    "i": "you",
    "i'm": "you are",
    "i'd": "you would",
    "i've": "you have",
    "i'll": "you will",
    "my": "your",
    "you are": "I am",
    "you were": "I was",
    "you've": "I have",
    "you'll": "I will",
    "your": "my",
    "yours": "mine",
    "you": "me",
    "me": "you"
}


class Chat(object):
    def __init__(self, pairs, reflections={}):
        """
        Initialize the chatbot.  Pairs is a list of patterns and responses.  Each
        pattern is a regular expression matching the user's statement or question,
        e.g. r'I like (.*)'.  For each such pattern a list of possible responses
        is given, e.g. ['Why do you like %1', 'Did you ever dislike %1'].  Material
        which is matched by parenthesized sections of the patterns (e.g. .*) is mapped to
        the numbered positions in the responses, e.g. %1.

        :type pairs: list of tuple
        :param pairs: The patterns and responses
        :type reflections: dict
        :param reflections: A mapping between first and second person expressions
```

```
        :rtype: None
        """

        self._pairs = [(re.compile(x, re.IGNORECASE), y) for (x, y) in pairs]
        self._reflections = reflections
        self._regex = self._compile_reflections()

    def _compile_reflections(self):
        sorted_refl = sorted(self._reflections.keys(), key=len,
                             reverse=True)
        return re.compile(r"\b({0})\b".format("|".join(map(re.escape,
                                                           sorted_refl))), re.IGNORECASE)

    def _substitute(self, str):
        """
        Substitute words in the string, according to the specified reflections,
        e.g. "I'm" -> "you are"

        :type str: str
        :param str: The string to be mapped
        :rtype: str
        """

        return self._regex.sub(lambda mo:
                               self._reflections[mo.string[mo.start():mo.end()]],
                               str.lower())

    def _wildcards(self, response, match):
        pos = response.find('%')
        while pos >= 0:
            num = int(response[pos + 1:pos + 2])
            response = response[:pos] + \
                       self._substitute(match.group(num)) + \
                       response[pos + 2:]
            pos = response.find('%')
        return response

    def respond(self, str):
        """
        Generate a response to the user input.

        :type str: str
        :param str: The string to be mapped
        :rtype: str
        """

        # check each pattern
        for (pattern, response) in self._pairs:
            match = pattern.match(str)

            # did the pattern match?
            if match:
                resp = random.choice(response)  # pick a random response
                resp = self._wildcards(resp, match)  # process wildcards

                # fix munged punctuation at the end
                if resp[-2:] == '?.': resp = resp[:-2] + '.'
                if resp[-2:] == '??': resp = resp[:-2] + '?'
                return resp

    # Hold a conversation with a chatbot
    def converse(self, quit="quit"):
        user_input = ""
        while user_input != quit:
            user_input = quit
            try:
                user_input = input(">")
            except EOFError:
                print(user_input)
            if user_input:
                while user_input[-1] in "!.": user_input = user_input[:-1]
```

```
                print(self.respond(user_input))


class LaunchVizDialog(gui.Dialog):
    def __init__(self, display_id):
        title = gui.Label('Launch Visualization')
        self.value = gui.Form()

        t = gui.Table()

        t.tr()
        t.td(gui.Label("YouTube Display ID: "), align=0, colspan=2)
        t.td(gui.Input(name="display_id", value=display_id, size=25))
        t.tr()
        e = gui.Button("Launch")
        e.connect(gui.CLICK, self.send, gui.CHANGE)
        t.td(e)
        ##

        e = gui.Button("Cancel")
        e.connect(gui.CLICK, self.close, None)
        t.td(e)

        gui.Dialog.__init__(self, title, t)


class ColorDialog(gui.Dialog):
    def __init__(self, value, **params):
        self.value = list(gui.parse_color(value))

        title = gui.Label("Color Picker")

        main = gui.Table()

        main.tr()

        self.color = gui.Color(self.value, width=64, height=64)
        main.td(self.color, rowspan=3, colspan=1)

        ## The sliders CHANGE events are connected to the adjust method.  The
        ## adjust method updates the proper color component based on the value
        ## passed to the method.
        ## ::
        main.td(gui.Label(' Red: '), 1, 0)
        e = gui.HSlider(value=self.value[0], min=0, max=255, size=32, width=128,
height=16)
        e.connect(gui.CHANGE, self.adjust, (0, e))
        main.td(e, 2, 0)
        ##

        main.td(gui.Label(' Green: '), 1, 1)
        e = gui.HSlider(value=self.value[1], min=0, max=255, size=32, width=128,
height=16)
        e.connect(gui.CHANGE, self.adjust, (1, e))
        main.td(e, 2, 1)

        main.td(gui.Label(' Blue: '), 1, 2)
        e = gui.HSlider(value=self.value[2], min=0, max=255, size=32, width=128,
height=16)
        e.connect(gui.CHANGE, self.adjust, (2, e))
        main.td(e, 2, 2)

        gui.Dialog.__init__(self, title, main)

    ##The custom adjust handler.
    ##::
    def adjust(self, value):
        (num, slider) = value
        self.value[num] = slider.value
        self.color.repaint()
        self.send(gui.CHANGE)
```

```
    ##


class PartsOfSpeech:
    def __init__(self):
        self.pos = get_pos_dict()

    def get_description(self, tag):
        if tag in self.pos.keys():
            return self.pos[tag][0]
        else:
            return ''

    def get_example(self, tag):
        if tag in self.pos.keys():
            return self.pos[tag][1]
        else:
            return ''

    def get_color(self, tag):
        if tag in self.pos.keys():
            return self.pos[tag][2]
        else:
            return 255, 255, 255


class Polyhedron:
    def __init__(self, vertices=[]):
        self.points = np.array([[v.x, v.y, v.z] for v in vertices])
        self.volumne = convex_hull_plot_2d(self.points)


class Point3D:
    def __init__(self, x=0, y=0, z=0):
        self.x, self.y, self.z = float(x), float(y), float(z)

    def rotateX(self, angle):
        """ Rotates the point around the X axis by the given angle in degrees. """
        rad = angle * math.pi / 180
        cosa = math.cos(rad)
        sina = math.sin(rad)
        y = self.y * cosa - self.z * sina
        z = self.y * sina + self.z * cosa
        return Point3D(self.x, y, z)

    def rotateY(self, angle):
        """ Rotates the point around the Y axis by the given angle in degrees. """
        rad = angle * math.pi / 180
        cosa = math.cos(rad)
        sina = math.sin(rad)
        z = self.z * cosa - self.x * sina
        x = self.z * sina + self.x * cosa
        return Point3D(x, self.y, z)

    def rotateZ(self, angle):
        """ Rotates the point around the Z axis by the given angle in degrees. """
        rad = angle * math.pi / 180
        cosa = math.cos(rad)
        sina = math.sin(rad)
        x = self.x * cosa - self.y * sina
        y = self.x * sina + self.y * cosa
        return Point3D(x, y, self.z)

    def project(self, win_width, win_height, fov, viewer_distance):
        """ Transforms this 3D point to 2D using a perspective projection. """
        factor = fov / (viewer_distance + self.z)
        x = self.x * factor + win_width / 2
        y = -self.y * factor + win_height / 2
        return Point3D(x, y, 1)
```

```
def get_pos_dict():
    return {'CC': ('coordinating conjunction', '', (240, 248, 255)),
            'CD': ('cardinal digit', '', (240, 248, 255)),
            'DT': ('determiner', '', (245, 255, 250)),
            'EX': ('existential there', '(like: "there is" ... think of it like "there
exists")', (255, 255, 255)),
            'FW': ('foreign word', '', (255, 255, 255)),
            'IN': ('preposition/subordinating conjunction', '', (240, 248, 255)),
            'JJ': ('adjective', 'big', (255, 248, 220)),
            'JJR': ('adjective, comparative', 'bigger', (255, 248, 220)),
            'JJS': ('adjective, superlative', 'biggest', (255, 248, 220)),
            'LS': ('list marker', '1)', (245, 255, 250)),
            'MD': ('modal', 'could, will', (255, 182, 193)),
            'NN': ('noun, singular', 'desk', (224, 255, 255)),
            'NNS': ('noun plural', 'desks', (224, 255, 255)),
            'NNP': ('proper noun, singular', 'Harrison', (224, 255, 255)),
            'NNPS': ('proper noun, plural', 'Americans', (224, 255, 255)),
            'PDT': ('predeterminer', 'all the kids', (245, 255, 250)),
            'POS': ('possessive ending', 'parent\'s', (255, 235, 205)),
            'PRP': ('personal pronoun', 'I, he, she', (224, 255, 255)),
            'PRP$': ('possessive pronoun', 'my, his, hers',
                     (224, 255, 255)),
            'RB': ('adverb', 'very, silently', (255, 240, 245)),
            'RBR': ('adverb, comparative', 'better', (255, 240, 245)),
            'RBS': ('adverb, superlative', 'best', (255, 240, 245)),
            'RP': ('particle', 'give up', (255, 240, 245)),
            'TO': ('to', 'go "to" the store', (240, 248, 255)),
            'UH': ('interjection', 'errrrrrrrm', (255, 255, 255)),
            'VB': ('verb', 'base form  take', (255, 182, 193)),
            'VBD': ('verb, past tense', 'took', (255, 182, 193)),
            'VBG': ('verb, gerund/present participle', 'taking', (255, 182, 193)),
            'VBN': ('verb, past participle', 'taken', (255, 182, 193)),
            'VBP': ('verb, sing. present, non-3d', 'take', (255, 182, 193)),
            'VBZ': ('verb, 3rd person sing. present', 'takes', (255, 182, 193)),
            'WDT': ('wh-determiner', 'which', (245, 255, 250)),
            'WP': ('wh-pronoun', 'who, what', (224, 255, 255)),
            'WP$': ('possessive wh-pronoun', 'whose', (255, 235, 205)),
            'WRB': ('wh-abverb', 'where, when', (255, 240, 245))}


def get_youtube_url(display_id):
    url = 'https://www.youtube.com/watch?v=' + display_id
    return url


def extract_youtube_data(display_id, output, overwrite):
    url = get_youtube_url(display_id)
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    folder_exists = os.path.isdir(folder)
    if not folder_exists or overwrite:
        ydl = youtube_dl.YoutubeDL({'outtmpl': folder + '%(id)s%(ext)s',
                                    'writesubtitles': True,
                                    # 'proxy': 'http://wwwproxy.sandia.gov:80',
                                    'allsubtitles': True,
                                    'writedescription': True,
                                    'writeinfojson': True,
                                    'writeannotations': True,
                                    'writeautomaticsub': True,
                                    'skip_download': False})
        with ydl:
            result = ydl.extract_info(
                url,
                download=True
            )


def write_audio_clip(display_id, output, overwrite):
    movie_path = get_youtube_video_path(display_id, output)
    audio_path = movie_path + '.mp3'
    file_exists = os.path.exists(audio_path)
    if not file_exists or overwrite:
```

```
        print('getting clip', movie_path)
        clip = VideoFileClip(movie_path)
        clip.audio.write_audiofile(audio_path)
    return audio_path


def check_for_cc_files(folder, display_id, extension):
    cc_exists = False
    json_file_path = os.path.join(folder, display_id + extension + '.info.json')
    with open(json_file_path, 'r') as f:
        data = json.load(f)
        if data['automatic_captions']:
            cc_exists = True
    return cc_exists


def get_url_parameters(url):
    parameters = {}
    main_parts = url.split('?')
    parameter_parts = main_parts[1].split('&')
    for kv in parameter_parts:
        kv_parts = kv.split('=')
        parameters[kv_parts[0]] = kv_parts[1]
    return parameters


def get_start_end_times(time_line):
    time_parts = time_line.split('-->')
    clock_time = time_parts[0].strip()
    parts = clock_time.split(':')
    sec_parts = parts[2].split('.')
    start_time = {'time': clock_time, 'hours': parts[0], 'minutes': parts[1],
                  'seconds': sec_parts[0], 'milliseconds': sec_parts[1]}
    clock_time = time_parts[1].strip()
    parts = clock_time.split(':')
    sec_parts = parts[2].split('.')
    end_time = {'time': clock_time, 'hours': parts[0], 'minutes': parts[1],
                'seconds': sec_parts[0], 'milliseconds': sec_parts[1]}
    return start_time, end_time


def get_time_difference(start_time, end_time):
    hours = int(end_time['hours']) - int(start_time['hours'])
    minutes = int(end_time['minutes']) - int(start_time['minutes'])
    seconds = int(end_time['seconds']) - int(start_time['seconds'])
    milliseconds = int(end_time['hours']) - int(start_time['milliseconds'])
    total = format(hours, '02') + ':' + format(minutes, '02') + ':' + format(seconds,
'02') + '.' + format(
        milliseconds,
        '03')
    return {'time': total, 'hours': format(hours, '02'), 'minutes': format(minutes,
'02'),
            'seconds': format(seconds, '02'), 'milliseconds': format(milliseconds, '02')}


def get_clean_line(line):
    remove_chr = False
    clean_line = ''
    for c in line:
        if c == '<':
            remove_chr = True
        elif c == '>':
            remove_chr = False
        if not remove_chr:
            clean_line += c
    clean_line = clean_line.replace('>', '')
    clean_line = clean_line.replace('\r', '')
    clean_line = clean_line.replace('\n', '')
    return clean_line + ' '
```

```
def get_text_lines_from_cc_file(cc_path):
    lines = []
    with codecs.open(cc_path, 'r', 'ISO-8859-2') as c:
        clean_line = ''
        timestamp = ''
        for line in c.readlines():
            if '-->' in line:
                if timestamp != '':
                    lines.append((timestamp, clean_line))
                time_line = line.replace('\n', '')
                start_time, end_time = get_start_end_times(time_line)
                elapsed_time = get_time_difference(start_time, end_time)
                timestamp = {'stamp': line.replace('\n', ''), 'start': start_time, 'end':
end_time,
                             'elapsed': elapsed_time}
                clean_line = ''
            else:
                clean_line += get_clean_line(line)
    return lines


def append_text_line(path, line):
    with codecs.open(path, 'a+', 'utf-16') as f:
        f.write(line.strip() + '\r\n')


def append_text(path, line):
    with codecs.open(path, 'a+', 'ISO-8859-2') as f:
        f.write(line.strip() + " ")


def get_youtube_media_extension(display_id, output):
    extension = 'mp4'
    for file in os.listdir(output + r'/youtube/' + display_id + r'/'):
        if file.endswith(".description"):
            extension = file.replace(display_id, '').replace('.description', '')
            break
    return extension


def extract_cc_plain_text(display_id, output, overwrite):
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    plain_text_path = os.path.join(folder, display_id + r'_plain.txt')
    word_time_stamp_json_path = os.path.join(folder, display_id + r'_time_line.json')
    file_exists = os.path.exists(plain_text_path)
    extension = get_youtube_media_extension(display_id, output)
    cc_exist = check_for_cc_files(folder, display_id, extension)
    text_lines = []
    if cc_exist:
        if not file_exists or overwrite:
            annotations_path = os.path.join(folder, display_id + extension +
r'.info.json')
            with open(annotations_path, 'r') as f:
                data = json.load(f)
                if data['automatic_captions']:
                    youtube_en_auto_url = data['automatic_captions']['en'][0]['url']
                    url_parameters = get_url_parameters(youtube_en_auto_url)
                    lang = url_parameters['lang']
                    cc_path = os.path.join(folder, display_id + extension + r'.' + lang +
'.vtt')
                    text_lines = get_text_lines_from_cc_file(cc_path)
    else:
        plain_text_path = ''
        word_time_stamp_json_path = ''
    return plain_text_path, word_time_stamp_json_path, text_lines


def write_pos_tagged_to_youtube_corpus(display_id, output, overwrite):
    plain_text_path, word_time_stamp_json_path, text_lines =
extract_cc_plain_text(display_id, output, overwrite)
    plain_exists = os.path.exists(plain_text_path)
```

```
    time_file_exists = os.path.exists(word_time_stamp_json_path)
    if plain_text_path == '':
        print('no captions')
        return
    corpora_path = os.path.join(output, r'youtube/' + display_id + r'/' + display_id +
r'.pos')
    corpus_file_exists = os.path.exists(corpora_path)
    if not corpus_file_exists or overwrite:
        if plain_exists:
            os.remove(plain_text_path)
        if time_file_exists:
            os.remove(word_time_stamp_json_path)
        if corpus_file_exists:
            os.remove(corpora_path)
        count = 0
        word_time_lines = []
        for timestamp, line in text_lines:
            words = nltk.tokenize.word_tokenize(line)
            tagged_words = nltk.pos_tag(words)
            tagged_line = ''
            plain_line = ''
            word_time_line = []
            for i in range(len(tagged_words)):
                tagged_line += tagged_words[i][0] + '/' + tagged_words[i][1] + ' '
                plain_line += tagged_words[i][0] + ' '
                word_time_line.append((tagged_words[i][0],
get_time_from_timestamp(timestamp, i)))
                count += 1
            tagged_line = tagged_line.strip() + '\r\n'
            append_text(corpora_path, tagged_line)
            append_text(plain_text_path, plain_line)
            word_time_lines.append(word_time_line)
        with codecs.open(word_time_stamp_json_path, 'w+', 'ISO-8859-2') as j:
            json.dump(word_time_lines, j)
        print(corpora_path, plain_text_path, word_time_stamp_json_path, count)


def get_convex_hull_delaunay_from_vertices(vertices):
    points = np.array(vertices)
    ch = None
    d = None
    try:
        if len(points) > 3:
            ch = scipy.spatial.ConvexHull(points)
            d = scipy.spatial.Delaunay(points)
    except:
        print("Oops!", sys.exc_info()[0], "occurred.")
    return ch, d


def write_domains_to_youtube_corpus(output):
    folder = os.path.join(output, r'youtube/')
    display_ids = [sub for sub in os.listdir(folder) if
os.path.isdir(os.path.join(folder, sub))]
    domains = {}
    domain_count = 0
    total_vertices = 0
    for display_id in display_ids:
        vectors = get_vectors(display_id, output)
        for vector in vectors:
            domain = vector['domain']
            v = vector['vector']
            vertex = [float(v[0]), float(v[1]), float(v[2])]
            if len(domain) > 0:
                if domain in domains:
                    if vertex not in domains[domain]['vertices']:
                        domains[domain]['vertices'].append(vertex)
                        domain_count += 1
                else:
                    domains[domain] = {'vertices': [vertex]}
                    domain_count += 1
```

```
        print(display_id, domain_count, 'vertices added')
        total_vertices += domain_count
        domain_count = 0
    # write domains file to corpus
    domains_path = os.path.join(folder, r'_domains.json')
    file_exists = os.path.exists(domains_path)
    if file_exists:
        os.remove(domains_path)
    with codecs.open(domains_path, 'w+', 'ISO-8859-2') as j:
        json.dump(domains, j)
        print(total_vertices, 'total vertices')


def write_topics_to_youtube_corpus(display_id, output, overwrite):
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    topics_json_path = os.path.join(folder, display_id + r'_topics.json')
    file_exists = os.path.exists(topics_json_path)
    extension = get_youtube_media_extension(display_id, output)
    cc_exist = check_for_cc_files(folder, display_id, extension)
    if cc_exist:
        if not file_exists or overwrite:
            annotations_path = os.path.join(folder, display_id + extension +
r'.info.json')
            with open(annotations_path, 'r') as f:
                data = json.load(f)
                title = ''
                categories = []
                description = ''
                tags = ''
                if data['title']:
                    title = data['title']
                if data['categories']:
                    for cat in data['categories']:
                        if '&' in cat:
                            cat_parts = cat.split('&')
                            for cat2 in cat_parts:
                                categories.append(cat2.strip())
                        else:
                            categories.append(cat.strip())
                if data['description']:
                    description = data['description']
                if data['tags']:
                    tags = data['tags']
                if overwrite and file_exists:
                    os.remove(topics_json_path)
                with codecs.open(topics_json_path, 'w+', 'ISO-8859-2') as j:
                    json.dump({"title": title, "categories": categories, "description":
description, "tags": tags}, j)
                print(topics_json_path)


def get_corpus_domains(output):
    domains = None
    folder = os.path.join(output, r'youtube/')
    domains_path = os.path.join(folder, r'_domains.json')
    file_exists = os.path.exists(domains_path)
    if file_exists:
        with codecs.open(domains_path, 'r', 'ISO-8859-2') as j:
            domains = json.load(j)
    return domains


def get_lexical_unit_pos(tag):
    pos = ''
    if tag.startswith('V') or tag.startswith('MD'):
        pos = 'v'  # v - verb
    elif tag.startswith('N') or tag.startswith('PRP') or tag.startswith('WP'):
        pos = 'n'  # n - noun
    elif tag.startswith('J'):
        pos = 'a'  # a - adjective
    elif tag.startswith('R') or tag == 'WRB':
```

```
        pos = 'adv'  # adv - adverb
    elif tag.startswith('IN'):
        pos = 'prep'  # prep - preposition
    elif tag.startswith('CD'):
        pos = 'num'  # num - numbers
    elif tag.startswith('UH'):
        pos = 'intj'  # intj - interjection
    elif tag.endswith('DT'):
        pos = 'art'  # art - article
    elif tag.startswith('CC'):
        pos = 'c'  # c - conjunction
    elif tag.startswith('IN'):
        pos = 'scon'  # scon - subordinating conjunction
    return pos


def penn2morphy(penntag, returnNone=False):
    morphy_tag = {'NN': wn.NOUN, 'JJ': wn.ADJ,
                  'VB': wn.VERB, 'RB': wn.ADV}
    try:
        return morphy_tag[penntag[:2]]
    except:
        return None if returnNone else ''


def get_word_frames_from_tagged_words(tagged_words):
    stopwords = nltk.corpus.stopwords.words('english')
    word_frames = []
    frame_counts = {}
    for tagged_word in tagged_words:
        if len(tagged_word) > 1:
            word = tagged_word[0]
            pos = get_lexical_unit_pos(tagged_word[1])
            frame = ''
            if word not in stopwords and word.isalpha() and len(pos) > 0:
                lexical_units = fn.lus(r'(?i)' + word)
                frames = [lu.frame.name for lu in lexical_units if lu.name.endswith(pos)]
                if frames and len(frames) > 0:
                    frame = frames[0]
            word_frames.append((word, frame))
            if frame not in frame_counts.keys():
                frame_counts[frame] = 1
            else:
                frame_counts[frame] += 1
    return word_frames, frame_counts


def write_word_frames_to_youtube_corpus(display_id, output, overwrite):
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    word_frame_path = os.path.join(folder, display_id + r'_frames.json')
    file_exists = os.path.exists(word_frame_path)
    if not file_exists or overwrite:
        fileid = display_id + r'/' + display_id + r'.pos'
        tagged_words = yttc.tagged_words([fileid])
        frames, counts = get_word_frames_from_tagged_words(tagged_words)
        if overwrite and file_exists:
            os.remove(word_frame_path)
        with codecs.open(word_frame_path, 'w+', 'ISO-8859-2') as j:
            json.dump({"frames": frames, "counts": counts}, j)
        print(word_frame_path)


def get_word_net_synsets_from_tagged_words(tagged_words):
    syns = []
    counts = {}
    for word, pos in tagged_words:
        ss = {}
        syn = wn.synsets(word, pos=penn2morphy(pos))
        sss = []
        for s in syn:
            abstract = {}
```

```
            similarity = 0
            if s:
                hyps = {}
                if isinstance(s, list):
                    hyps = s[0].hypernyms()
                    if hyps:
                        if isinstance(hyps, list):
                            abstract = hyps[-1:]
                            similarity = s[0].wup_similarity(abstract[0])
                        else:
                            abstract = hyps
                            similarity = s[0].wup_similarity(abstract)
                else:
                    hyps = s.hypernyms()
                    if hyps:
                        if isinstance(hyps, list):
                            abstract = hyps[-1:]
                            similarity = s.wup_similarity(abstract[0])
                        else:
                            abstract = hyps
                            similarity = s.wup_similarity(abstract)
            sss.append((str(s), str(abstract), similarity))
        syns.append((word, pos, sss))
        if word not in counts.keys():
            counts[word] = len(sss)
        else:
            counts[word] += len(sss)
    return syns, counts


def write_word_nets_to_youtube_corpus(display_id, output, overwrite):
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    word_net_path = os.path.join(folder, display_id + r'_word_nets.json')
    file_exists = os.path.exists(word_net_path)
    if not file_exists or overwrite:
        fileid = display_id + r'/' + display_id + r'.pos'
        tagged_words = yttc.tagged_words([fileid])
        syns, counts = get_word_net_synsets_from_tagged_words(tagged_words)
        if overwrite and file_exists:
            os.remove(word_net_path)
        with codecs.open(word_net_path, 'w+', 'ISO-8859-2') as j:
            json.dump({"syns": syns, "counts": counts}, j)
        print(word_net_path)


def get_seconds_from_time_string(time_stamp):
    tp = time_stamp.split(':')
    seconds = 0
    if tp and len(tp) > 2:
        tps = tp[2].split('.')
        seconds = (int(tp[0]) * 60 * 60) + (int(tp[1]) * 60) + int(tps[0])
    return seconds


def get_time_string_from_seconds(word_seconds):
    t = ''
    ts = str(word_seconds).split('.')
    h = 60 * 60
    m = 60
    hms = int(ts[0])
    if hms >= h:
        hs = int(hms / h) * h
        t += str(int(hms / h)).zfill(2) + ':'
        hms = hms - hs
    else:
        t += '00:'
    if hms >= m:
        hm = int(hms / m) * m
        t += str(int(hms / m)).zfill(2) + ':'
        hms = hms - hm
    else:
```

```
        t += '00:'
    if hms > 0:
        t += str(hms).zfill(2)
    else:
        t += '00'
    if len(ts) > 1:
        t += '.' + str(int(ts[1])).zfill(3)[:3]
    else:
        t += '.000'
    return t


def get_time_from_timestamp(timestamp, i):
    start_hours = int(timestamp['start']['hours'])
    start_minutes = int(timestamp['start']['minutes'])
    start_seconds = int(timestamp['start']['seconds'])
    start_milliseconds = abs(int(timestamp['start']['milliseconds']))
    word_seconds = (start_seconds + (start_hours * 60 * 60) + (start_minutes * 60) +
(start_milliseconds / 1000))
    hours = int(timestamp['elapsed']['hours'])
    minutes = int(timestamp['elapsed']['minutes'])
    seconds = int(timestamp['elapsed']['seconds'])
    milliseconds = abs(int(timestamp['elapsed']['milliseconds']))
    word_seconds += ((seconds + (hours * 60 * 60) + (minutes * 60) + (milliseconds /
1000)) / (i + 1)) * i
    t = get_time_string_from_seconds(word_seconds)
    return t


def get_word_times(display_id, output):
    word_times = []
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    word_time_stamp_json_path = os.path.join(folder, display_id + r'_time_line.json')
    file_exists = os.path.exists(word_time_stamp_json_path)
    if file_exists:
        with codecs.open(word_time_stamp_json_path, 'r', 'ISO-8859-2') as j:
            word_time_lines = json.load(j)
            for word_time_line in word_time_lines:
                for word, timestamp in word_time_line:
                    t = (word, timestamp)
                    word_times.append(t)
    return word_times


def get_topic(display_id, output):
    topic = None
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    word_topic_json_path = os.path.join(folder, display_id + r'_topics.json')
    file_exists = os.path.exists(word_topic_json_path)
    if file_exists:
        with codecs.open(word_topic_json_path, 'r', 'ISO-8859-2') as j:
            topic = json.load(j)
    return topic


def get_corpus_fileids(output):
    folder = os.path.join(output, r'youtube/')
    ids = [item for item in os.listdir(folder) if os.path.isdir(os.path.join(folder,
item))]
    fileids = []
    for id in ids:
        fileid = id + '/' + id + '.pos'
        fileids.append(fileid)
    return fileids


def get_discourse_categories(output):
    folder = os.path.join(output, r'youtube/')
    ids = [item for item in os.listdir(folder) if os.path.isdir(os.path.join(folder,
item))]
    discourse_categories = {}
```

```
    for id in ids:
        topic = get_topic(id, output)
        discourse_categories[id] = topic['categories']
    return discourse_categories


def get_category_discourses(output):
    dcs = get_discourse_categories(output)
    cs = []
    for d in dcs.keys():
        for c in dcs[d]:
            cs.append(c)
    cats = set(cs)
    category_discourses = {}
    for c in cats:
        category_discourses[c] = [d + '/' + d + '.pos' for d in dcs.keys() if c in
dcs[d]]
    return category_discourses


def get_tabular_frequency_distribution_by_category(words, output, export=None):
    discourse_categories = get_discourse_categories(output)
    category_discourses = get_category_discourses(output)
    tfdbc = {}
    for c in category_discourses.keys():
        dws = yttc.words(category_discourses[c])
        dts = nltk.Text(dws)
        fds = nltk.FreqDist(dts)
        wfs = {}
        for word in words:
            wfs[word] = (fds[word], fds.freq(word))
        tfdbc[c] = wfs
    all_words = yttc.words()
    all_text = nltk.Text(all_words)
    all_fd = nltk.FreqDist(all_text)
    all_word_frequencies = {}
    for word in words:
        all_word_frequencies[word] = (all_fd[word], all_fd.freq(word))
        tfdbc['all'] = all_word_frequencies
    if export:
        header = ''
        for word in words:
            header += '\t\t' + word
        if os.path.exists(export):
            os.remove(export)
        append_text_line(export, header)
        for c in tfdbc.keys():
            line = c
            for w in tfdbc[c].keys():
                line += '\t{}\t{}'.format(tfdbc[c][w][0], tfdbc[c][w][1])
            append_text_line(export, line)
    return tfdbc


def get_frames(display_id, output):
    frames = {}
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    word_frames_json_path = os.path.join(folder, display_id + r'_frames.json')
    file_exists = os.path.exists(word_frames_json_path)
    if file_exists:
        with codecs.open(word_frames_json_path, 'r', 'ISO-8859-2') as j:
            frames = json.load(j)
    return frames


def get_knns(display_id, output):
    knns = {}
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    word_frames_json_path = os.path.join(folder, display_id + r'_knn.json')
    file_exists = os.path.exists(word_frames_json_path)
    if file_exists:
```

```
        with codecs.open(word_frames_json_path, 'r', 'ISO-8859-2') as j:
            knns = json.load(j)
    return knns


def get_word_nets(display_id, output):
    word_nets = []
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    word_word_nets_json_path = os.path.join(folder, display_id + r'_word_nets.json')
    file_exists = os.path.exists(word_word_nets_json_path)
    if file_exists:
        with codecs.open(word_word_nets_json_path, 'r', 'ISO-8859-2') as j:
            word_nets = json.load(j)
    return word_nets


def get_entrenchment(tagged_word, word_freq_dists):
    entrenchment = 0.0
    c = len(word_freq_dists)
    sfd = word_freq_dists.most_common()
    wfd = [(sfd[i], i) for i in range(0, len(sfd) - 1) if sfd[i][0] in [tagged_word[0]]]
    arr = [sfd[i][1] for i in range(len(sfd) - 1)]
    min = np.min(arr)
    max = np.max(arr)
    if wfd and len(wfd) > 0:
        dst = wfd[0][0][1]
        entrenchment = dst / (max - min)
    return entrenchment


def get_overlap(word_net, i):
    overlap = 0.0
    syns = word_net['syns']
    syn = syns[i]
    if len(syn[2]) > 0:
        arr = [syn[2][i][2] for i in range(0, len(syn[2]))]
        overlap = np.max(arr)
    return 1 - overlap


def get_salience(frames, i):
    salience = 0.0
    frame = frames['frames'][i]
    domain = frame[1]
    if len(domain) > 0:
        counts = frames['counts']
        domain_counts = []
        for key in counts.keys():
            if len(key) > 0:
                domain_counts.append((key, counts[key]))
        arr = [domain_counts[i][1] for i in range(len(domain_counts) - 1)]
        min = np.min(arr)
        max = np.max(arr)
        d = [domain_counts[i][1] for i in range(0, len(domain_counts) - 1) if domain ==
domain_counts[i][0]]
        if d and len(d) > 0:
            dct = d[0]
            salience = dct / (max - min)
    return salience, frame[1]


def get_clean_frequency_distributions_from_tagged_words(tagged_words):
    stop_words = set(stopwords.words('english'))
    punctuations = str.maketrans('', '', string.punctuation)
    words = [word for word in [w.translate(punctuations) for w, p in tagged_words] if
             word not in stop_words and word.isalpha()]
    freq_dist_list = nltk.FreqDist(words)
    return freq_dist_list


def get_word_vectors_from_tagged_words(display_id, output, tagged_words):
```

```
    vectors = []
    word_times = get_word_times(display_id, output)
    frames = get_frames(display_id, output)
    word_nets = get_word_nets(display_id, output)
    clean_word_freq_dists =
get_clean_frequency_distributions_from_tagged_words(tagged_words)
    for i in range(0, len(tagged_words) - 1):
        # get entrenchment as frequency of word in entire corpus
        entrenchment = get_entrenchment(tagged_words[i], clean_word_freq_dists)
        # get overlap as similarity of synset to abstract hypernym
        overlap = get_overlap(word_nets, i)
        # get salience as usage of frames in discourse
        salience, frame = get_salience(frames, i)
        vectors.append({'word': tagged_words[i][0], 'pos': tagged_words[i][1], 'domain':
frame, 'time': word_times[i],
                        'vector': (str(entrenchment), str(overlap), str(salience))})
    return vectors


def write_word_vectors_to_youtube_corpus(display_id, output, overwrite):
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    word_vectors_path = os.path.join(folder, display_id + r'_vectors.json')
    file_exists = os.path.exists(word_vectors_path)
    if not file_exists or overwrite:
        fileid = display_id + r'/' + display_id + r'.pos'
        tagged_words = yttc.tagged_words([fileid])
        vectors = get_word_vectors_from_tagged_words(display_id, output, tagged_words)
        if overwrite and file_exists:
            os.remove(word_vectors_path)
        with codecs.open(word_vectors_path, 'w+', 'ISO-8859-2') as j:
            json.dump(vectors, j)
        print(word_vectors_path)


def get_vectors(display_id, output):
    vectors = []
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    word_vectors_path = os.path.join(folder, display_id + r'_vectors.json')
    file_exists = os.path.exists(word_vectors_path)
    if file_exists:
        with codecs.open(word_vectors_path, 'r', 'ISO-8859-2') as j:
            vectors = json.load(j)
    return vectors


def get_knns_from_vectors(vectors):
    knns = []
    data = [[float(d['vector'][0]), float(d['vector'][1]), float(d['vector'][2])] for d
in vectors]
    point_array = np.array(data)
    kdt = KDTree(point_array, 100000)
    for i in range(len(vectors)):
        distances, indices = kdt.query(point_array[i], 2)
        neighbors = {'current': vectors[i], 'neighbor': vectors[indices[1]], 'distance':
float(distances[1])}
        knns.append(neighbors)
    return knns


def write_k_nn_word_vectors_to_youtube_corpus(display_id, output, overwrite):
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    word_knn_path = os.path.join(folder, display_id + r'_knn.json')
    file_exists = os.path.exists(word_knn_path)
    if not file_exists or overwrite:
        vectors = get_vectors(display_id, output)
        knns = get_knns_from_vectors(vectors)
        if overwrite and file_exists:
            os.remove(word_knn_path)
        with codecs.open(word_knn_path, 'w+', 'ISO-8859-2') as j:
            json.dump(knns, j)
        print(word_knn_path)
```

```
def print_file_id_and_topic_information(display_id, output, overwrite, export):
    fileid = display_id + r'/' + display_id + r'.pos'
    words = yttc.words(fileid)
    t = len(words)
    s = len(set(words))
    title = display_id
    description = ''
    vl = "unknown"
    vls = 0
    word_times = get_word_times(display_id, output)
    if word_times and len(word_times) > 0:
        wt = word_times[-1]
        vl = wt[1]
        vls = get_seconds_from_time_string(vl)
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    topics_json_path = os.path.join(folder, display_id + r'_topics.json')
    file_exists = os.path.exists(topics_json_path)
    if file_exists:
        with open(topics_json_path, 'r') as f:
            data = json.load(f)
            if data['title'] and len(data['title']) > 0:
                title = data['title']
            if data['categories'] and len(data['categories']) > 0:
                title += ' - '
                for cat in data['categories']:
                    title += ' ' + cat
            print('{:<15} {:<10} {:<10} {:<10} {}'.format(display_id, t, s, vl, title))
            if export and len(export) > 4:
                if data['description'] and len(data['description']) > 0:
                    description = data['description']
                append_text_line(export, '{}\t{}\t{}\t{}\t{}'.format(display_id, t, s,
vl, title))
    return t, s, vls


def print_vector_data(display_id, output, overwrite, start, stop, export):
    folder = output + r'/youtube/' + display_id + r'/'
    word_vectors_path = folder + display_id + r'_vectors.json'
    file_exists = os.path.exists(word_vectors_path)
    if file_exists:
        with codecs.open(word_vectors_path, 'r', 'ISO-8859-2') as j:
            word_vectors = json.load(j)
            r = range(0, len(word_vectors))
            if start != '':
                seconds = get_seconds_from_time_string(start)
                for i in r:
                    i_seconds = get_seconds_from_time_string(word_vectors[i]['time'][1])
                    if seconds >= i_seconds:
                        r = range(i, len(word_vectors))
                        break
            if stop != '':
                seconds = get_seconds_from_time_string(stop)
                for i in r:
                    i_seconds = get_seconds_from_time_string(word_vectors[i]['time'][1])
                    if seconds <= i_seconds:
                        r = range(i, len(word_vectors))
                        break
            for i in r:
                word = word_vectors[i]['word']
                frame = word_vectors[i]['domain']
                pos = word_vectors[i]['pos']
                time = word_vectors[i]['time'][1]
                entrenchment = word_vectors[i]['vector'][0]
                overlap = word_vectors[i]['vector'][1]
                salience = word_vectors[i]['vector'][2]
                print(
                    '{:<15} {:<30} {:<30} {:<5} <{:<15}, {:<15}, {:<15}> '.format(time,
word, frame, pos, entrenchment,
```

```
overlap,

salience))
                if export and len(export) > 4:
                    append_text_line(export,
                                     '{}\t{}\t{}\t{}\t{}\t{}\t{}'.format(time, word,
frame, pos, entrenchment, overlap,
                                                                        salience))


def plot_3d_surface_from_corpus_vectors(display_id, output, overwrite):
    vectors = get_vectors(display_id, output)
    fig = plt.figure()
    ax = fig.gca(projection='3d')
    X = []
    Y = []
    Z = []
    max_t = float(get_seconds_from_time_string(vectors[-1]['time'][1]))
    min_t = float(get_seconds_from_time_string(vectors[0]['time'][1]))
    for v in vectors:
        t = float(get_seconds_from_time_string(v['time'][1])) / (max_t - min_t)
        x = [float(v['vector'][0]), t]
        y = [float(v['vector'][1]), t]
        z = [float(v['vector'][2]), t]
        X.append(x)
        Y.append(y)
        Z.append(z)
    cmhot = plt.get_cmap('coolwarm')
    surf = ax.plot_surface(X, Y, Z, cmap=cmhot, linewidth=0, antialiased=False)
    ax.set_zlim(0.00, 1.00)
    ax.zaxis.set_major_locator(LinearLocator(10))
    ax.zaxis.set_major_formatter(FormatStrFormatter('%.02f'))
    fig.colorbar(surf, shrink=0.5, aspect=5)
    plt.show()


def draw_vector_viz_surface(vertices, edges, surfaces):
    glBegin(GL_QUADS)
    for surface in surfaces:
        glColor3fv(vertices[surface[0]])
        for vertex in surface:
            glVertex3fv(vertices[vertex])
    glEnd()
    glBegin(GL_LINES)
    for edge in edges:
        for vertex in edge:
            glVertex3fv(vertices[vertex])
    glEnd()


def draw_knn_viz(screen, knns):
    vertices = []
    edges = []
    for i in range(len(knns)):
        if len(knns[i]['current']['domain']) > 0 and len(knns[i]['neighbor']['domain']) >
0:
            v1 = knns[i]['current']['vector']
            v2 = knns[i]['neighbor']['vector']
            vertices.append((float(v1[0]), float(v1[1]), float(v1[2])))
            vertices.append((float(v2[0]), float(v2[1]), float(v2[2])))
    for i in range(0, len(vertices), 2):
        edges.append((i, i + 1))
    # draw
    glBegin(GL_LINES)
    for edge in edges:
        glColor3fv(vertices[edge[0]])
        for vertex in edge:
            glVertex3fv(vertices[vertex])
    glEnd()
```

```
def draw_vector_viz(screen, cubes, edges, labels, colors, surfaces):
    # draw cubes
    glBegin(GL_QUADS)
    for i in range(len(cubes)):
        glColor3fv(colors[i])
        for surface in surfaces:
            for vertex in surface:
                glVertex3fv(cubes[i][vertex])
    glEnd()
    glBegin(GL_LINES)
    for i in range(len(cubes)):
        glColor3fv(colors[i])
        for edge in edges:
            for vertex in edge:
                glVertex3fv(cubes[i][vertex])
    glEnd()
    glBegin(GL_LINES)
    for i in range(len(cubes)):
        glColor3fv(colors[i])
        glVertex3fv((cubes[i][0][0], 0, 0))
        glVertex3fv(cubes[i][0])
    glEnd()
    # draw labels
    for i in range(len(cubes)):
        x = cubes[i][0][0]
        y = cubes[i][0][1] + 0.02
        z = cubes[i][0][2]
        gl_draw_text((x, y, z), labels[i][0])
        gl_draw_text((x, y - 0.04, z), labels[i][1])


def get_cubes_edges_labels_colors_surfaces_from_vectors(vectors, domains):
    labels = []
    colors = []
    surfaces = []
    cubes = []
    vertices = []
    wait_times = []
    edges = [(0, 1), (1, 3), (3, 2), (2, 6), (6, 7), (7, 5), (5, 4), (4, 0), (0, 2), (4,
6), (3, 7), (1, 5)]
    surfaces = [(0, 1, 2, 3), (1, 5, 7, 3), (2, 3, 7, 6), (0, 2, 6, 4), (0, 1, 5, 4), (4,
6, 7, 5)]
    vecs = [vectors[i] for i in range(len(vectors))]
    # if vectors[i]['domain'] in domains
    if len(vecs) > 0:
        inc = 0
        for i in range(len(vecs)):
            if i < len(vecs) - 1:
                it = get_seconds_from_time_string(vecs[i]['time'][1])
                nt = get_seconds_from_time_string(vecs[i + 1]['time'][1])
                wait_times.append((nt - it) * 1000)
                inc = (nt - it) / 1000
            # corner vertices
            v = vecs[i]['vector']
            x = float(v[0])
            xi = (float(v[0]) + (inc * i))
            y = float(v[1])
            z = float(v[2])
            vertices.append((xi, y, z))
            colors.append((x, y, z))
            # labels
            if vectors[i]['domain'] in domains:
                label = (
                    '[' + vecs[i]['domain'].upper() + ' / ' + vecs[i]['word'] + ']',
                    '<' + str(round(x, 4)) + ', ' + str(round(y, 4)) + ', ' +
str(round(z, 4)) + '>')
            else:
                label = ('', '')
            labels.append(label)
        # cubes and edges
        for i in range(len(vertices)):
```

```
            x = vertices[i][0]
            y = vertices[i][1]
            z = vertices[i][2]
            d = 0.01
            cube = ((x, y, z), (x, y, z - d), (x, y + d, z), (x, y + d, z - d), (x + d,
y, z), (x + d, y, z - d),
                    (x + d, y + d, z), (x + d, y + d, z - d))
            cubes.append(cube)
    return cubes, edges, labels, colors, surfaces, wait_times


def get_conceptual_spaces(vectors, x_scale=100, y_scale=100, z_scale=100):
    spaces = {}
    for vector in vectors:
        domain = vector['domain']
        v = vector['vector']
        x = int(float(v[0]) * x_scale)
        y = int(float(v[1]) * y_scale)
        z = int(float(v[2]) * z_scale)
        vertex = Point3D(x, y, z)
        if len(domain) > 0:
            if domain in spaces:
                spaces[domain]['vertices'].append(vertex)
            else:
                spaces[domain] = {'vertices': [vertex]}
    return spaces


def gl_draw_text(position, textString):
    font = pygame.font.Font(None, 12)
    textSurface = font.render(textString, True, (0, 0, 0, 255), (245, 245, 220, 255))
    textData = pygame.image.tostring(textSurface, "RGBA", True)
    glRasterPos3d(*position)
    glDrawPixels(textSurface.get_width(), textSurface.get_height(), GL_RGBA,
GL_UNSIGNED_BYTE, textData)


def draw_single_vector_viz(screen, edges, cube, label, vec_color, surfaces):
    # draw cubes
    glBegin(GL_QUADS)
    glColor3fv(vec_color)
    for surface in surfaces:
        for vertex in surface:
            glVertex3fv(cube[vertex])
    glEnd()
    glBegin(GL_LINES)
    glColor3fv(vec_color)
    for edge in edges:
        for vertex in edge:
            glVertex3fv(cube[vertex])
    glEnd()
    glBegin(GL_LINES)
    glColor3fv(vec_color)
    glVertex3fv((cube[0][0], 0, 0))
    glVertex3fv(cube[0])
    glEnd()
    # draw labels
    x = cube[0][0]
    y = cube[0][1] + 0.02
    z = cube[0][2]
    gl_draw_text((x, y, z), label[0])
    gl_draw_text((x, y - 0.04, z), label[1])


def draw_conceptual_spaces(screen, vectors):
    conceptual_spaces = get_conceptual_spaces(vectors)
    angleX, angleY, angleZ = 0, 0, 0
    for d in conceptual_spaces.keys():
        f = len(conceptual_spaces[d]['vertices'])
        if f > 2:
            t = []
```

```
            vs = conceptual_spaces[d]['vertices']
            for i in range(f):
                # Rotate the point around X axis, then around Y axis, and finally around
Z axis.
                r = vs[i].rotateX(angleX).rotateY(angleY).rotateZ(angleZ)
                # Transform the point from 3D to 2D
                p = r.project(screen.get_width(), screen.get_height(), 256, 4)
                # Put the point in the list of transformed vertices
                t.append(p)
            for j in range(0, len(t), 2):
                if j == len(t) - 1:
                    pygame.draw.line(screen, (255, 255, 255), (t[j].x, vs[j].y), (t[0].x,
t[0].y))
                else:
                    pygame.draw.line(screen, (255, 255, 255), (t[j].x, t[j].y), (t[j +
1].x, t[j + 1].y))


def draw_vector_word_domain_text(surface, offset, position, word, domain, part_of_speech,
vector_string, timestamp,
                                 fore_color=(0, 0, 0), back_color=(255, 255, 255)):
    indent = 5
    basic_font = pygame.font.SysFont(None, 24)
    word_text = basic_font.render(word, True, fore_color, back_color)
    domain_text = basic_font.render(domain, True, fore_color, back_color)
    pos_text = basic_font.render(part_of_speech, True, fore_color, back_color)
    vector_text = basic_font.render(vector_string, True, fore_color, back_color)
    time_text = basic_font.render(timestamp, True, fore_color, back_color)
    time_text_rect = time_text.get_rect()
    time_text_rect.left += (offset * position) + indent
    time_text_rect.centery = surface.get_height() - time_text_rect.height
    surface.blit(time_text, time_text_rect)
    pos_text_rect = pos_text.get_rect()
    pos_text_rect.left += (offset * position) + indent
    pos_text_rect.centery = surface.get_height() - pos_text_rect.height -
time_text_rect.height
    surface.blit(pos_text, pos_text_rect)
    vector_text_rect = vector_text.get_rect()
    vector_text_rect.left += (offset * position) + indent
    vector_text_rect.centery = surface.get_height() - vector_text_rect.height -
pos_text_rect.height - time_text_rect.height
    surface.blit(vector_text, vector_text_rect)
    domain_text_rect = domain_text.get_rect()
    domain_text_rect.left += (offset * position) + indent
    domain_text_rect.centery = surface.get_height() - domain_text_rect.height -
vector_text_rect.height - pos_text_rect.height - time_text_rect.height
    surface.blit(domain_text, domain_text_rect)
    word_text_rect = word_text.get_rect()
    word_text_rect.left += (offset * position) + indent
    word_text_rect.centery = surface.get_height() - word_text_rect.height -
domain_text_rect.height - vector_text_rect.height - pos_text_rect.height -
time_text_rect.height
    surface.blit(word_text, word_text_rect)
    return word_text_rect.left, word_text_rect.top


def get_vertex_from_string_vector(v):
    vertex = (float(v[0]), float(v[1]), float(v[2]))
    return vertex


def draw_origin_vector_line(surface, origin, vertex, offset, colors):
    w_scale = offset - 10
    h_scale = origin[1] - 30
    salience_scale = len(colors)
    start_pos = (origin[0], origin[1])
    end_pos = (round(vertex[0] * w_scale) + origin[0], origin[1] - round(vertex[1] *
h_scale))
    line_color = get_color_from_vertex(vertex, colors)
    line_width = round(vertex[2] * (salience_scale / 5)) + 1
    pygame.draw.line(surface, line_color, start_pos, end_pos, line_width)
```

```
    radius = round(vertex[2] * salience_scale) + 1
    end_circle = pygame.draw.circle(surface, line_color, end_pos, radius)


def get_color_from_vertex(vertex, colors):
    c = int(vertex[2] * len(colors)) - 1
    if c < 0:
        c = 0
    if c > len(colors) - 1:
        c = len(colors) - 1
    return colors[c]


def draw_domain_vertices_scatter(surface, origin, vertices, offset, colors):
    w_scale = offset - 10
    h_scale = origin[1] - 30
    salience_scale = len(colors)
    for vertex in vertices:
        point = (round(vertex[0] * w_scale) + origin[0], origin[1] - round(vertex[1] *
h_scale))
        circle_color = get_color_from_vertex(vertex, colors)
        radius = round(vertex[2] * salience_scale) + 1
        pygame.draw.circle(surface, circle_color, point, radius, 1)


def get_short_vector_string(vector_tuple, decimal_places=4):
    precision = decimal_places * 4
    x = round(float(vector_tuple[0]) / precision) / precision
    y = round(float(vector_tuple[1]) * precision) / precision
    z = round(float(vector_tuple[2]) * precision) / precision
    return str(x) + ',' + str(y) + ',' + str(z)


def draw_rainbow(surface, colors, x, y, w=5, h=15, indent=5):
    for c in range(len(colors)):
        x = (w * c) + indent
        pygame.draw.rect(surface, colors[c], (x, y, w, h))


def get_viz_colors():
    colors = []
    red_frequency = .1
    green_frequency = .2
    blue_frequency = .3
    center = 128
    width = 127
    phase1 = 0
    phase2 = 2
    phase3 = 4
    length = 50
    for i in range(length):
        red = int(math.sin(red_frequency * i + phase1) * width + center)
        green = int(math.sin(green_frequency * i + phase2) * width + center)
        blue = int(math.sin(blue_frequency * i + phase3) * width + center)
        colors.append((red, green, blue))
    return colors


def fill_rect_surface_with_color(surface, pos_color, x, y, w, h):
    rect = pygame.draw.rect(surface, pos_color, (x, y, w, h))


def draw_prev_curr_next_vectors(surface, vectors, domain_vertices, i, lv, rv, colors):
    parts_of_speech = PartsOfSpeech()
    v_cnt = 1 + lv + rv
    w = surface.get_width()
    h = surface.get_height()
    offset = round(w / v_cnt)
    curr_pos = 0
    for vi in range(i - lv, i + rv + 1):
        if -1 < vi < len(vectors):
```

```
            vector_string = get_short_vector_string(vectors[vi]['vector'])
            tag = vectors[vi]['pos']
            pos_descr = parts_of_speech.get_description(tag)
            pos_color = parts_of_speech.get_color(tag)
            fill_rect_surface_with_color(surface, pos_color, (offset * curr_pos), 0,
offset, h)
            x, y = draw_vector_word_domain_text(surface, offset, curr_pos,
vectors[vi]['word'], vectors[vi]['domain'],
                                                pos_descr, vector_string,
vectors[vi]['time'][1], back_color=pos_color)
            origin = (x, y - 10, 0)
            if vectors[vi]['domain'] in domain_vertices.keys():
                vertices = domain_vertices[vectors[vi]['domain']]['vertices']
                draw_domain_vertices_scatter(surface, origin, vertices, offset, colors)
            vertex = get_vertex_from_string_vector(vectors[vi]['vector'])
            draw_origin_vector_line(surface, origin, vertex, offset, colors)
        curr_pos += 1


def get_youtube_video_path(display_id, output):
    folder = os.path.join(output, r'youtube/' + display_id + r'/')
    extension = get_youtube_media_extension(display_id, output)
    movie_path = os.path.join(folder, display_id + extension)
    return movie_path


def get_youtube_audio_path(display_id, output):
    audio_path = get_youtube_video_path(display_id, output)
    audio_path += '.mp3'
    return audio_path


def get_domain_convex_hulls(output, vectors):
    hulls = None
    domains = get_corpus_domains(output)
    if domains is not None:
        hulls = {}
        for vector in vectors:
            domain = vector['domain']
            if domain in domains.keys():
                vertices = domains[domain]['vertices']
                ch, d = get_convex_hull_delaunay_from_vertices(vertices)
                if ch is not None:
                    points = [(x, y, z) for x, y, z in ch.simplices]
                    if domain not in hulls.keys():
                        hulls[domain] = points
    return hulls


def imdisplay(imarray, screen=None):
    """Splashes the given image array on the given pygame screen """
    a = pygame.surfarray.make_surface(imarray.swapaxes(0, 1))
    if screen is None:
        screen = pygame.display.set_mode(imarray.shape[:2][::-1])
    screen.blit(a, (0, 0))
    pygame.display.flip()


def draw_text_to_surface(surface, text, x, y, max_width, text_color, back_color,
indent=5):
    basic_font = pygame.font.SysFont(None, 24)
    txt = text + ' '
    new_text = basic_font.render(txt, True, text_color, back_color)
    new_text_rect = new_text.get_rect()
    if (x + new_text_rect.width) > max_width:
        x = indent
        y = y + new_text_rect.height
    new_text_rect.left = x
    new_text_rect.top = y
    surface.blit(new_text, new_text_rect)
    return new_text_rect
```

```
def draw_topic(surface, topic_info, total_time, total_words, colors):
    indent = 5
    x = indent
    y = 290
    title = topic_info['title']
    title_words = title.split()
    categories = topic_info['categories']
    tags = topic_info['tags']
    total_time_text = 'Total Time: ' + total_time
    total_words_text = 'Total Words: ' + str(total_words)
    colors_text = 'Salience Scale: (' + str(len(colors)) + ') 0.0 to 1.0'
    max_width = 270
    last_rect_height = 0
    for i in range(len(title_words)):
        try:
            rect = draw_text_to_surface(surface, title_words[i], x, y, max_width, (0, 0,
0), (255, 255, 255))
            x = rect.right
            y = rect.top
            last_rect_height = rect.height
        except:
            pass
    x = indent
    y = y + last_rect_height
    rect = draw_text_to_surface(surface, 'Categories:', x, y, max_width, (0, 0, 0), (255,
255, 255))
    last_rect_height = rect.height
    x = indent
    y = y + last_rect_height
    for i in range(len(categories)):
        try:
            rect = draw_text_to_surface(surface, categories[i], x, y, max_width, (0, 0,
0), (255, 255, 255))
            x = rect.right
            y = rect.top
            last_rect_height = rect.height
        except:
            pass

    x = indent
    y = y + last_rect_height
    rect = draw_text_to_surface(surface, 'Tags:', x, y, max_width, (0, 0, 0), (255, 255,
255))
    last_rect_height = rect.height
    x = indent
    y = y + last_rect_height
    for i in range(len(tags)):
        try:
            rect = draw_text_to_surface(surface, tags[i], x, y, max_width, (0, 0, 0),
(255, 255, 255))
            x = rect.right
            y = rect.top
            last_rect_height = rect.height
        except:
            pass
    x = indent
    y = y + last_rect_height
    rect = draw_text_to_surface(surface, total_time_text, x, y, max_width, (0, 0, 0),
(255, 255, 255))
    last_rect_height = rect.height
    x = indent
    y = y + last_rect_height
    rect = draw_text_to_surface(surface, total_words_text, x, y, max_width, (0, 0, 0),
(255, 255, 255))
    last_rect_height = rect.height
    x = indent
    y = y + last_rect_height
    rect = draw_text_to_surface(surface, colors_text, x, y, max_width, (0, 0, 0), (255,
255, 255))
```

```
    last_rect_height = rect.height
    x = indent
    y = y + last_rect_height
    draw_rainbow(surface, colors, x, y)


def get_vector_index_from_current_time(t, vectors):
    remaining = [i for i in range(len(vectors)) if
get_seconds_from_time_string(vectors[i]['time'][1]) > t]
    if len(remaining) > 0:
        return remaining[0]
    else:
        return None


def draw_time_stamp(surface, t, indent=5):
    time_stamp = get_time_string_from_seconds(t)
    x = indent
    y = surface.get_height() - 20
    draw_text_to_surface(surface, time_stamp, x, y, 270, (0, 255, 0), (255, 255, 255))


def get_adjusted_video_time(key, t, first, last):
    tt = t
    if pygame.key.get_mods() & pygame.KMOD_SHIFT:
        modification = 10.0
    elif pygame.key.get_mods() & pygame.KMOD_CTRL:
        modification = 60.0
    elif pygame.key.get_mods() & pygame.KMOD_ALT:
        modification = 300.0
    elif pygame.key.get_mods() & pygame.KMOD_CTRL & pygame.KMOD_SHIFT:
        modification = 600.0
    elif pygame.key.get_mods() & pygame.KMOD_CTRL & pygame.KMOD_ALT:
        modification = 1800.0
    else:
        modification = 1.0
    if key == pygame.K_RIGHT:
        tt = tt + modification
        if tt > last:
            tt = last
    elif key == pygame.K_LEFT:
        tt = tt - modification
        if tt < first:
            tt = first
    elif key == pygame.K_END:
        tt = last
    elif key == pygame.K_HOME:
        tt = first
    return tt


def restart_audio_at_position(t, audio_path):
    pygame.mixer.quit()
    pygame.mixer.init()
    pygame.mixer.music.load(audio_path)
    pygame.mixer.music.play(-1, t)


def draw_analysis_legend(surface):
    fore_color = (255, 255, 255)
    back_color = (0, 0, 0)
    basic_font = pygame.font.SysFont(None, 24)
    baseline_text = basic_font.render('BASELINE', True, fore_color, back_color)
    baseline_text_rect = baseline_text.get_rect()
    baseline_text_rect.centerx = round(surface.get_width() / 2) + 165
    baseline_text_rect.centery = 25
    surface.blit(baseline_text, baseline_text_rect)
    elaboration_text = basic_font.render('ELABORATION', True, fore_color, back_color)
    elaboration_text_rect = elaboration_text.get_rect()
    elaboration_text_rect.centerx = round(surface.get_width() / 2) + 165
    elaboration_text_rect.centery = surface.get_height() - 25
```

```
    surface.blit(elaboration_text, elaboration_text_rect)
    frequent_text = basic_font.render('FREQUENT', True, fore_color, back_color)
    frequent_text = pygame.transform.rotate(frequent_text, 90)
    frequent_text_rect = frequent_text.get_rect()
    frequent_text_rect.centerx = surface.get_width() - 25
    frequent_text_rect.centery = round(surface.get_height() / 2)
    surface.blit(frequent_text, frequent_text_rect)
    novel_text = basic_font.render('NOVEL', True, fore_color, back_color)
    novel_text = pygame.transform.rotate(novel_text, -90)
    novel_text_rect = novel_text.get_rect()
    novel_text_rect.centerx = 305
    novel_text_rect.centery = round(surface.get_height() / 2)
    surface.blit(novel_text, novel_text_rect)


def viz_vector_data(display_id, output, overwrite, start, stop, export):
    # prep data
    white = (255, 255, 255)
    colors = get_viz_colors()
    vectors = get_vectors(display_id, output)
    topic_info = get_topic(display_id, output)
    topic_total_words = len(vectors)
    topic_time_length = vectors[-1]['time'][1]
    # domain_hulls = get_domain_convex_hulls(output, vectors)
    domain_vertices = get_corpus_domains(output)
    # start presentation
    os.environ['SDL_VIDEO_WINDOW_POS'] = "%d,%d" % (10, 100)
    pygame.init()
    display = (1850, 900)
    screen = pygame.display.set_mode(display, pygame.RESIZABLE)
    pygame.display.set_caption('Path to Alignment - Semantic change in discourse')
    icon = pygame.image.load('assets/exu.png')
    pygame.display.set_icon(icon)
    viz_surface = pygame.display.get_surface()
    analysis_surface = pygame.Surface((1470, 800))
    draw_analysis_legend(viz_surface)
    # set up movie surface
    fps = 15
    video_surface = pygame.Surface((280, 900))
    video_surface.fill((255, 255, 255))
    movie_path = get_youtube_video_path(display_id, output)
    movie = VideoFileClip(movie_path, target_resolution=(280, 280))
    img = movie.get_frame(0)
    imdisplay(img, video_surface)
    result = []
    t0 = time.time()
    first = 1.0 / fps
    last = movie.duration - .001
    if len(start) > 0:
        first = get_seconds_from_time_string(start)
    if len(stop) > 0:
        last = get_seconds_from_time_string(stop)
    # sound
    pygame.mixer.init()
    audio_path = get_youtube_audio_path(display_id, output)
    pygame.mixer.music.load(audio_path)
    pygame.mixer.music.play(-1, first)
    fts = [t1 for t1 in np.arange(first, last, 1.0 / fps)]
    pausing = False
    while True:
        mt = pygame.mixer.music.get_pos()
        remaining = [t2 for t2 in fts if t2 > (mt / 1000)]
        if len(remaining) > 0:
            t = remaining[0]
        else:
            pygame.quit()
            quit()
        img = movie.get_frame(t)
        # handle events
        for event in pygame.event.get():
            if event.type == pygame.QUIT:
```

```
                pygame.quit()
                quit()
            elif event.type == pygame.KEYDOWN:
                if event.key == pygame.K_ESCAPE:
                    print("Keyboard interrupt")
                    return result
                elif event.key == pygame.K_SPACE:
                    pausing = not pausing
                    if pausing:
                        pygame.mixer.music.pause()
                    else:
                        pygame.mixer.music.unpause()
                elif event.key == pygame.K_RIGHT or event.key == pygame.K_LEFT or
event.key == pygame.K_END \
                        or event.key == pygame.K_HOME:
                    tt = get_adjusted_video_time(event.key, t, first, last)
                    if tt is not t:
                        t = tt
                        restart_audio_at_position(t, audio_path)
                elif event.key == pygame.K_F1:
                    dialog = ColorDialog("#00ffff")
                    dialog.open()
            elif event.type == pygame.MOUSEBUTTONDOWN:
                x, y = pygame.mouse.get_pos()
        t1 = time.time()
        time.sleep(max(0, t - (t1 - t0)))
        imdisplay(img, video_surface)
        draw_topic(video_surface, topic_info, topic_time_length, topic_total_words,
colors)
        draw_time_stamp(video_surface, t)
        # draw analysis viz
        i = get_vector_index_from_current_time(t, vectors)
        if i is not None:
            analysis_surface.fill(white)
            draw_prev_curr_next_vectors(analysis_surface, vectors, domain_vertices, i, 1,
1, colors)
            viz_surface.blit(analysis_surface, (331, 51))
        viz_surface.blit(video_surface, (0, 0))
        # update and wait
        pygame.display.update()


def viz_launcher(ids, output, overwrite, start, finish):
    app = gui.Desktop()
    app.connect(gui.QUIT, app.quit, None)
    title = gui.Label('Path to Alignment - Launch Visualization')
    app.add(title, 50, 5)
    icon = pygame.image.load('assets/exu.png')
    pygame.display.set_icon(icon)
    c = gui.Table(width=640, height=900)

    def onchange(value):
        script = r'C:\ProgramData\Anaconda3\python.exe C:/LING/599-
Thesis/code/youtube_corpus_tool.py'
        cmd = 'viz_corpus_vector_data:' + value.value['display_id'].value
        os.system(script + ' ' + cmd)

    def tab():
        box.widget = g.value

    g = gui.Group()
    g.connect(gui.CHANGE, tab)
    c.tr()

    t1 = gui.Table(width=1800, height=900)
    b = gui.Tool(g, gui.Label("Visualizations", width=25), t1)
    c.td(b, align=-1)

    def launch_viz_dialog(display_id):
        dialog = LaunchVizDialog(display_id)
        dialog.connect(gui.CHANGE, onchange, dialog)
```

```
        dialog.open()

    t1.tr()
    t1.td(gui.Label('VIDEO'))
    t1.td(gui.Label('WORDS'))
    t1.td(gui.Label('SET'))
    t1.td(gui.Label('DURATION'))
    t1.td(gui.Label('DESCRIPTION'))

    for display_id in ids:
        fileid = display_id + r'/' + display_id + r'.pos'
        words = yttc.words(fileid)
        t = len(words)
        s = len(set(words))
        title = display_id
        description = ''
        vl = "unknown"
        vls = 0
        word_times = get_word_times(display_id, output)
        if word_times and len(word_times) > 0:
            wt = word_times[-1]
            vl = wt[1]
            vls = get_seconds_from_time_string(vl)
        folder = os.path.join(output, r'youtube/' + display_id + r'/')
        topics_json_path = os.path.join(folder, display_id + r'_topics.json')
        file_exists = os.path.exists(topics_json_path)
        if file_exists:
            with open(topics_json_path, 'r') as f:
                data = json.load(f)
                if data['title'] and len(data['title']) > 0:
                    title = data['title']
                if data['categories'] and len(data['categories']) > 0:
                    title += ' - '
                    for cat in data['categories']:
                        title += ' ' + cat
                t1.tr()
                btn = gui.Button(display_id)
                btn.connect(gui.CLICK, launch_viz_dialog, display_id)
                t1.td(btn, align=-1)
                t1.td(gui.Label(str(t)))
                t1.td(gui.Label(str(s)))
                t1.td(gui.Label(str(vl)))
                t1.td(gui.Label(unicodedata2.normalize('NFKD', title).encode('ascii',
'ignore')), align=-1)

    c.tr()
    spacer = gui.Spacer(1850, 900)
    box = gui.ScrollArea(spacer, height=900)
    c.td(box)
    app.run(c)


def main(command, output, overwrite, start, finish, export):
    folder = os.path.join(output, r'youtube/')
    ids = [item for item in os.listdir(folder) if os.path.isdir(os.path.join(folder,
item))]
    command_parts = command.split(':')
    cmd = command_parts[0]
    val = ''
    if len(command_parts) > 1:
        val = command_parts[1]
    if val != '':
        ids = [val]
    # header for print
    tws = 0
    uws = 0
    vls = 0
    if cmd == 'make_all_domain_vertices':
        write_domains_to_youtube_corpus(output)
        return
    elif cmd == 'viz_launcher':
```

```
        viz_launcher(ids, output, overwrite, start, finish)
        return
    if cmd == 'print_corpus_info':
        print('{:<15} {:<10} {:<10} {:<10} {}'.format('Display ID', 'Total Words',
'Unique Words', 'Total Time',
                                                      'Title'))
    elif cmd == 'print_corpus_vector_data':
        print('{:<15} {:<30} {:<30} {:<5} <{:<15}, {:<15}, {:<15}> '.format('Time',
'Word', 'frame', 'POS',

'Entrenchment', 'Overlap',
                                                                    'Salience'))
    if export and len(export) > 4 and os.path.exists(export):
        os.remove(export)
    for did in ids:
        if cmd == 'download_youtube':
            extract_youtube_data(did, output, overwrite)
        elif cmd == 'make_audio_file':
            write_audio_clip(did, output, overwrite)
        elif cmd == 'make_pos_text':
            write_pos_tagged_to_youtube_corpus(did, output, overwrite)
        elif cmd == 'make_topic_list':
            write_topics_to_youtube_corpus(did, output, overwrite)
        elif cmd == 'make_word_frames':
            write_word_frames_to_youtube_corpus(did, output, overwrite)
        elif cmd == 'make_word_net':
            write_word_nets_to_youtube_corpus(did, output, overwrite)
        elif cmd == 'make_word_vectors':
            write_word_vectors_to_youtube_corpus(did, output, overwrite)
        elif cmd == 'make_word_k_nn_vectors':
            write_k_nn_word_vectors_to_youtube_corpus(did, output, overwrite)
        elif cmd == 'plot_vector_surface':
            plot_3d_surface_from_corpus_vectors(did, output, overwrite)
        elif cmd == 'print_corpus_info':
            t, s, v = print_file_id_and_topic_information(did, output, overwrite, export)
            tws += t
            uws += s
            vls += v
        elif cmd == 'print_corpus_vector_data':
            print_vector_data(did, output, overwrite, start, finish, export)
        elif cmd == 'viz_corpus_vector_data':
            viz_vector_data(did, output, overwrite, start, finish, export)
    if cmd == 'print_corpus_info':
        total_video_time = get_time_string_from_seconds(vls)
        tuws = len(set(yttc.words()))
        idct = str(len(ids)) + ' videos'
        print('-------------------------------------------------------------------------
-----------------')
        print('{:<15} {:<10} {:<10} {:<10} {}'.format(idct, tws, tuws, total_video_time,
''))


if __name__ == '__main__':
    parser = argparse.ArgumentParser()
    parser.add_argument('command',
                        help='Command to carry out, e.g. "download:8IiDgZK-Fz4" downloads
the target youtube video id for data extraction.')
    parser.add_argument('-p', '--output', help='The output location for corpus files.')
    parser.add_argument('-o', '--overwrite', help='overwrites existing data',
action='store_true')
    parser.add_argument('-s', '--start', help='The start time for word in corpus files.')
    parser.add_argument('-e', '--stop', help='The stop time for word in corpus files.')
    parser.add_argument('-x', '--export', help='The path of file to export data.')
    args = parser.parse_args()
    out_path = 'corpora'
    begin = ''
    end = ''
    export_path = ''
    if args.output:
        out_path = args.output
    if args.start:
```

```
    end = args.start
if args.stop:
    stop = args.stop
if args.export:
    export_path = args.export
if args.command:
    main(args.command, out_path, args.overwrite, begin, end, export_path)
else:
    parser.print_help()
```

9 References

Bird, S., Klein, E., & Loper, E. (2009). *Natural Language Processing with Python.* Sebastopol, CA 95472: O'Reilly Media Inc. Retrieved from Natural Language Toolkit: http://www.nltk.org/book

Bybee, J. (2007). *Frequency of Use and the Organization of Language.* New York, NY: Oxford University Press.

Bybee, J. (2010). *Language, Usage and Cognition.* Cambridge: Cambridge University Press.

Chafe, W. (1977). *The recall and verbalization of past experience.* Bloomington: Indiana University Press.

Chafe, W. (1994). *Discourse, consciousness and time: the flow and displacement of conscious experience in speaking and writing.* Chicago: University of Chicago Press.

Chafe, W. (1996). How consciousness shapes language. *Pragmatics and Cognition, 4*, 35-54.

Chemero, A. (2011). Embodied Cognition and Radical Embodied Cognition. In A. Chemero, *Radical Embodied Cognitive Science* (pp. 17-44). Cambridge: MIT Press.

Chemero, A. (2011). *Radical Embodied Cognitive Science.* Cambridge, Massachusetts: The MIT Press.

Chilton, P. (2014). *Language, Space and Mind.* Cambridge, UK: Cambridge University Press.

Comrie, B., Haspelmath, M., & Bickel, B. (2008). *The Leipzig Glossing Rules.* Retrieved from Max Planck Institute for Evolutionary Anthropology: http://www.eva.mpg.de/lingua/pdf/LGR08.02.05.pdf

Coulson, S. (2001). *Semantic Leaps, Frame-Shifting and Conceptual Blending in Meaning Construction.* Cambridge: Cambridge University Press.

Croft, W. (2001). *Radical Construction Grammar Syntactic Theory in Typological Perspective.* Oxford: Oxford University Press.

Croft, W. (2003). *Typology and Universals* (2nd ed.). New York, New York, USA: Cambridge University Press.

Croft, W. (2007, April). Intonation Units and Grammatical Structure in Wardaman and in Cross-linguistic Perspective. *Australian Journal of Linguistics, 27*(1), 1-39.

Croft, W. (2012). *Verbs, Aspect and Causal Structure.* Oxford, United Kingdom: Oxford University Press.

Croft, W. (2014). *Morphosyntax Incomplete Draft.* Albuquerque, New Mexico: Not Specified.

Croft, W., & Cruse, A. D. (2004). *Cognitive Linguistics.* New York, New York, USA: Cambridge University Press.

Dale, R. (2015, 7 3). An integrative research strategy for exploring synergies in natural language performance. *Ecological Psychology, 27*(3), 190-201. Retrieved from https://pdfs.semanticscholar.org/da17/49a98ba3d9928acf6730793c60edd1db3b06.pdf

Dehaene, S. (2014). *CONSCIOUSNESS AND THE BRAIN.* New York: Penguin Books.

Edelman, G. M. (2006). *second nature* . New Haven: Yale University Press.

Edelman, G. M., & Tononi, G. (2000). *Universe of Consciousness.* New York: Basic Books.

Edelman, S. (2007). Bridging Language with the Rest of Cognition. In M. Gonzalez-Marquez, I. Mittelberg, S. Coulson, & M. J. Spivey, *Methods in Cognitive Linguistics* (pp. 424-445). Amsterdam: John Benjamins.

Everett, D. L. (2017). *How Language Began.* New York: Liveright Publishing Company.

Fried, D., Polajnar, T., & Clark, S. (2015). *Low-Rank Tensors for Verbs in Compositional Distributional Semantics.* University of Cambridge, Computer Laboratory. Cambridge: University of Cambridge. Retrieved from https://people.eecs.berkeley.edu/~dfried/papers/FPC-verb_tensors.pdf

Friston, K. J., Tononi, G., Reeke Jr., G. N., Sporns, O., & Edelman, G. M. (1994). VALUE-DEPENDENT SELECTION IN THE BRAIN: SIMULATION IN A SYNTHETIC NEURAL MODEL. *Neuroscience, 59*(2), 229-243.

Gärdenfors, P. (2014). Conceptual Spaces. In P. Gärdenfors, *The Geometry of Meaning* (pp. 21-52). Cambridge: The MIT Press.

Godfrey-Smith, P. (2016). *Other Minds: THE OCTOPUS, THE SEA, and THE DEEP ORIGINS of CONSCIOUSNESS.* New York: Farrar, Straus and Giroux.

Goodfellow, I., Bengio, Y., & Courville, A. (2017). *Deep Learning.* Cambridge, MA, USA: MIT Press.

Gries, S. T. (2013). Data in Construction Grammar. In T. Hoffmann, & G. Trousdale, *The Oxford Handbook of Construction Grammar* (pp. 93-108). New York: Oxford University Press.

Gries, S. T. (2015, 10 16). The role of quantitative methods in cognitive linguistics. *Change of Paradigms–New Paradoxes: Recontextualizing Language and Linguistics, 31*, 311. Retrieved from Google Scholar: http://www.linguistics.ucsb.edu/faculty/stgries/research/2015_STG_ContingWrdsCxs_DGFestschrift.pdf

Hagan, M. T., Demuth, H. B., Beale, M. H., & De Jesus, O. (2016). *Neural Network Design* (2 ed.). San Bernadino, CA, USA: Hagan and Demuth. Retrieved from http://hagan.okstate.edu/nnd.html

Hoffmann, T., & Trousdale, G. (2013). *The Oxford Handbook of Construction Grammar.* (T. Hoffmann, & G. Trousdale, Eds.) Oxford: Oxford University Press. doi:10.1093/oxfordhb/9780195396683.001.0001

Labov, W., & Waletzky, J. (1966). *Narrative Analysis: Oral Versions of Personal Experience.* New York and Boston: Columbia University and Harvard University.

Lambrecht, K. (1994). *Information Structure and Sentence Form, Topic, focus and the mental representations of discourse referents* (Vol. 71). New York, New York: Cambridge University Press.

Langacker, R. W. (2008). *Cognitive Grammar A Basic Introduction.* Oxford: Oxford University Press.

Langacker, R. W. (2008, March). Discourse in Cognitive Grammar. *Cognitive Linguistics, 12*(2), 143–188. doi: 10.1515/cogl.12.2.143

Langacker, R. W. (2016). Baseline and elaboration. *Cognitive Linguistics, 27*(3), 405-439. doi:10.1515/cog-2015-0126

Last Name, F. M. (Year). Article Title. *Journal Title*, Pages From - To.

Last Name, F. M. (Year). *Book Title*. City Name: Publisher Name.

Longacre, R. E. (2007). *Language typology and syntactic description* (2nd ed., Vol. 2). (T. Shopen, Ed.) Cambridge: Cambridge University Press.

Merlan, F. C. (1994). A grammar of Wardaman: a language of the Northern territory of Australia. In F. C. Merlan, G. Bossong, & W. Chafe (Eds.), *Mouton Grammar Library 11* (Vol. 11, pp. 1-613). Berlin - New York: Walter de Gruyter.

Miller, G. A. (1995). WordNet: A Lexical Database for English. *Communications of the ACM, 38*(11), 39-41.

Müller, A., & Guido, S. (2016). *Introduction to Machine Learning with Python.* Sebastopol: O'Reilly.

Noë, A. (2006). The Enactive Approach to Perception. In A. Noë, *Action in Perception* (pp. 1-34). Cambridge: The MIT Press.

Onnis, L., & Spivey, M. J. (2012). Toward a New Scientific Visualization for the Language Sciences. *Information*(3), 124-150. doi:10.3390/info3010124

Palmer, F. R. (2001). *Mood and Modality* (2nd ed.). Cambridge, UK: Cambridge University Press.

Python Software Foundation. (2001, January 1). *Python*. Retrieved from Python Software Foundation: https://www.python.org/

Ruppenhofer, J., Ellsworth, M., Petruck, M. R., Johnson, C. R., Baker, C. F., & Scheffczyk, J. (2016, 11 1). *FrameNet II: Extended Theory and Practice.* Retrieved from Frame Net: https://framenet.icsi.berkeley.edu/fndrupal/sites/default/files/book2016.11.01.pdf

Russell, J. A. (1980). A circumplex model of affect. *Journal of Personality and Social Psychology, 39*, 1161-1178.

Smith, C. S. (2003). *Modes of Discourse, The Local Structure of Texts* (Vol. 103). New York: Cambridge University Press.

Spivey, M. (2007). *THE CONTINUITY OF MIND* (Vol. 44). New York: Oxford University Press.

Steels, L. (2011). *Design Patterns in Fluid Construction.* Amsterdam: John Benjamins.

Steels, L. (2013). Fluid Construction Grammar. In T. Hoffman, & G. Trousdale, *The Oxford Handbook of Construction Grammar* (pp. 153-167). New York: Oxford University Press.

Steels, L. (2016, 8 30). *Basics of Fluid Construction Grammar.* Retrieved from Google Scholar: https://www.fcg-net.org/wp-content/uploads/papers/basics-of-fcg.pdf

Thelen, E., & Smith, L. B. (2000). *A Dynamic Systems Approach to the Development of Cognition and Action.* Cambridge: MIT Press.

Thompson, S. A., Longacre, R. E., & Hwang, S. J. (2007). Adverbial clauses. (T. Shopen, Ed.) *Language typology and syntactic description, 2*, 237-300.