Evaluating the Effect of Word Sense Disambiguation Applied to Ambiguous Terms with Biological and English Contexts.

Scholars Electives Research Project

Supervisor: Dr. Robert Mercer
Written By: Nicholas Elder
Computer Science
Western University
April 7, 2017

ABSTRACT:

This study examines the effectiveness of word sense disambiguation as applied to ambiguous terms that exist with biomedical and English senses. Adagram was used to automatically split words into a number of unique senses when processing biomedical journal data. The effectiveness of these senses was compared to the results generated by a tool without word sense disambiguation, Word2Vec. The success of word sense disambiguation was measured as an improvement in the distance between the expected result and the actual result of the word embedding process from Word2Vec to Adagram. Word sense disambiguation enabled better understanding of the sense of a word in context.

ACKNOWLEDGEMENTS:

Dr. Robert Mercer, Supervisor.

Dr. Jeff Hutter, Course Coordinator.

TABLE OF CONTENTS:

INTRODUCTION:

Word Senses:

Words in language often have multiple meanings known as senses which are each used in distinct contexts. A Bank is a building which houses money, however it also is the land near a river, a group or mass of items, tilting on an angle, and more. These various senses are understood by humans parsing language, however computers understand words as singular units. This single computer-understood unit causes ambiguities to exist when computers attempt to understand language.

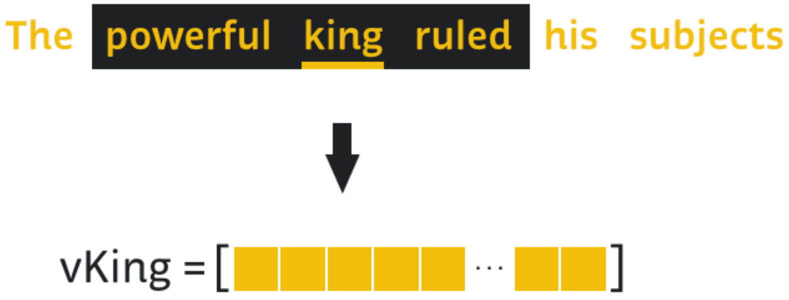
Natural Language Processing:

Natural language is complex but follows patterns which allow meaning to be communicated. Computer systems capable of understanding natural language derive their meaning from a large body of text, known as a corpus, which adequately expresses many examples of these patterns in language. Word embedding is one technique used to derive an understanding from a corpus. This process uses a vector space model which typically contains one vector per word. This vector space is defined with hundreds of dimensions used to store the complex relationships between words in language. Processing the corpus slowly translates[1] the vectors from their initial randomized positions towards a location with meaning in the model. Every example of the word found in the corpus contributes a small vector translation to this process. This vector space model stores word similarities in the proximity between the word vectors and relative meaning in the differences between vectors. Word embedding thus enables computers to automatically understand the meaning in language. This understanding allows for programs which can, for example, determine the capital city of any given country.

Each vector in the word embedding model represents an individual word and consists of a value of position from the origin in each dimension of space such as Vector = [0.123, -12.3, ... , 0.01]. This high-dimensionality allows the relationships in language to be represented. The word embedding process uses n-dimensional vector space and the words preceding and following each word to train the vector space model. This frame of context around each word is used to determine the direction of the vector translation. Typically 300 dimensional vectors, and a frame of 2 words on each side of the target word are used. The training process then process gigabytes of text to translate each word vector from their original randomized positions slowly towards their true meaning based locations in the vector space. As Figure 1 illustrates, king is surrounded by words like powerful and ruled which are likely to also surround words of similar meaning, this moving both words into similar positions in vector space. This training process uses a neural network to manage the translations made to the vector space model.

[1] Translate in the sense of adjusting the position of a vector by some amount in some direction.

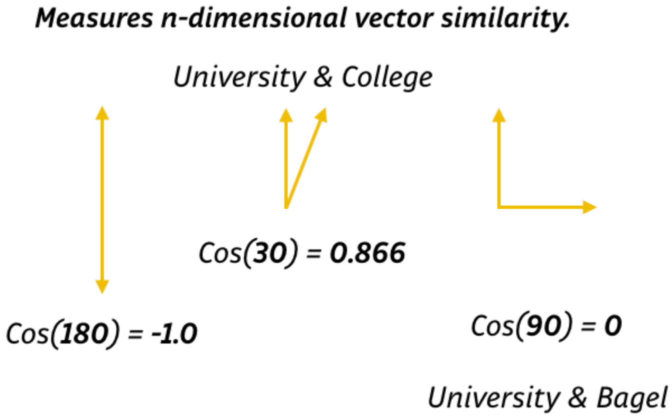
Figure 1: Word Embedding Target and Context Generates Vector Space Model Location.



The result is a trained vector space model in which the relative locations of words represent their relative meaning to each other. Similar words are also found in clusters, with synonyms likely extremely close in space to one another. King and Queen for example may be similarly located. Once this model is produced it can be queried to understand the language used in the corpus.

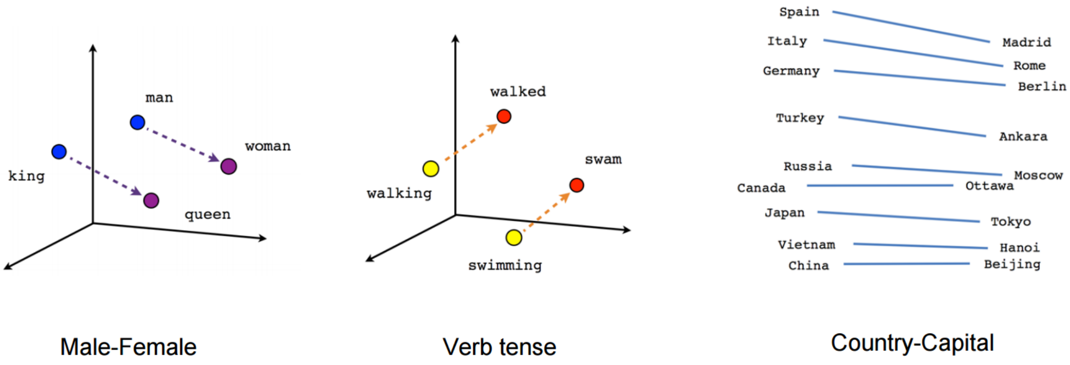
Within the highly dimensional vector space model, cosine distances are used to measure the similarity between points. Figure 2 illustrates some sample cosine distances and the angles they represent alongside examples from the word embedding vector space.

Figure 2: 2D representations of cosine distances with 300D example values from the model.



Due to the relative meaning stored in the model, as illustrated in Figure 3, taking the Vector(King) and subtracting the male quality Vector(Man) and adding instead the female quality Vector(Woman) results approximately in the Vector(Queen), the female version of a King. This relationship is illustrated in Figure 3. In a model trained on a small amount of data (100MB) the predicted location of Queen, was closest to the word’s actual location with a cosine distance of 0.62. With a larger 1GB corpus, this measure improved to 0.89. This improvement serves to demonstrate the value of a larger corpus. Hence it is possible to derive meaning from the relative locations of words in the word embedding model.

Figure 3: *Relational meaning of words in 3D and 2D vector space.*[2]



Male-Female

Verb tense

Country-Capital

The points indicated with colour in Figure 3 are representative of the heads of their respective vectors. These points in reality represent a vector extending from the origin to the indicated point. Due to the number of vectors involved, the heads are simply illustrated as points in space, with the origin as their starting point implied.

Problem:

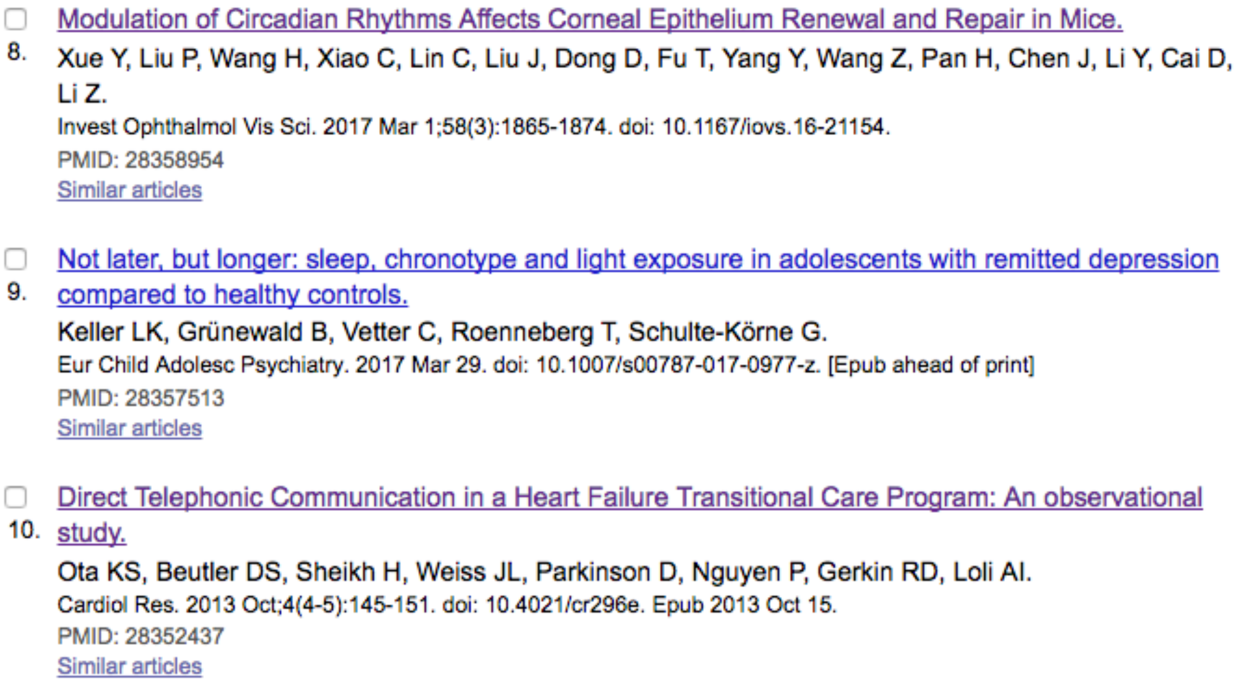
This study aims to address the problem posed by multiple word senses for the word embedding model which does not represent multiple word senses. In linguistics this process is referred to as word sense disambiguation. The occurrence of word sense ambiguity targeted here is within the biomedical domain. There are a number of words which have English and Gene-related meaning. Disambiguating these two senses will have a benefit for natural language processing applications in this field.

For example, should a researcher search for *clock* on the PubMed database, they may be expecting to find results related to the Clock Circadian Regulator.[3] However, using a traditional keyword-based search they might also be presented anything related to the English meaning of the word clock instead, confusing the results presented between multiple senses. Figure 4 outlines one example of this, with result 8 relating to the gene CLOCK and result 10 simply relating to individuals who had received "around the clock telephone access." This ambiguity is the weakness of typical keyword searches and would be solved by word sense disambiguation.

[2] "Vector Representations of Words," TensorFlow, accessed January 15, 2017, https://www.tensorflow.org/tutorials/word2vec/.

[3] "Clock Gene," Gene Cards, Accessed April 1, 2017, http://www.genecards.org/cgi-bin/carddisp.pl?gene=CLOCK&keywords=clock.

Figure 4: Sense Ambiguity as Seen on PubMed.[4]



□ 8. Modulation of Circadian Rhythms Affects Corneal Epithelium Renewal and Repair in Mice.
Xue Y, Liu P, Wang H, Xiao C, Lin C, Liu J, Dong D, Fu T, Yang Y, Wang Z, Pan H, Chen J, Li Y, Cai D, Li Z.
Invest Ophthalmol Vis Sci. 2017 Mar 1;58(3):1865-1874. doi: 10.1167/iovs.16-21154.
PMID: 28358954
Similar articles

□ 9. Not later, but longer: sleep, chronotype and light exposure in adolescents with remitted depression compared to healthy controls.
Keller LK, Grünewald B, Vetter C, Roenneberg T, Schulte-Körne G.
Eur Child Adolesc Psychiatry. 2017 Mar 29. doi: 10.1007/s00787-017-0977-z. [Epub ahead of print]
PMID: 28357513
Similar articles

□ 10. Direct Telephonic Communication in a Heart Failure Transitional Care Program: An observational study.
Ota KS, Beutler DS, Sheikh H, Weiss JL, Parkinson D, Nguyen P, Gerkin RD, Loli AI.
Cardiol Res. 2013 Oct;4(4-5):145-151. doi: 10.4021/cr296e. Epub 2013 Oct 15.
PMID: 28352437
Similar articles

The typical word embedding model currently treats each spelling of a word as an individual unit, and thus ambiguous terms may end up somewhere between the two or more locations. Each location representing the expected location of one sense in the vector space model.

Study Focus & Hypothesis:
This study will compare the ability of a tool which can disambiguate word senses over a baseline established by a tool which can not, and measure effectiveness on the basis of the similarity of a word to its correct sense.

This study hypothesizes that a tool that can disambiguate word senses will be more successful in placing each word in a similar location to its correct word sense over a baseline established by a tool which can not.

Vocabulary Base:
To generate a list of words to disambiguate that were likely to be both biomedical terms and English words, the intersect of these sets was generated. The English words were pulled from the local Unix dictionary to a text file.[5] The biomedical terms were gathered from the Human Gene Nomenclature Committee list of genes and synonyms.[6] At first a nested loop compared the dictionaries, however this code, outlined in Appendix A, proved quite slow and thus set

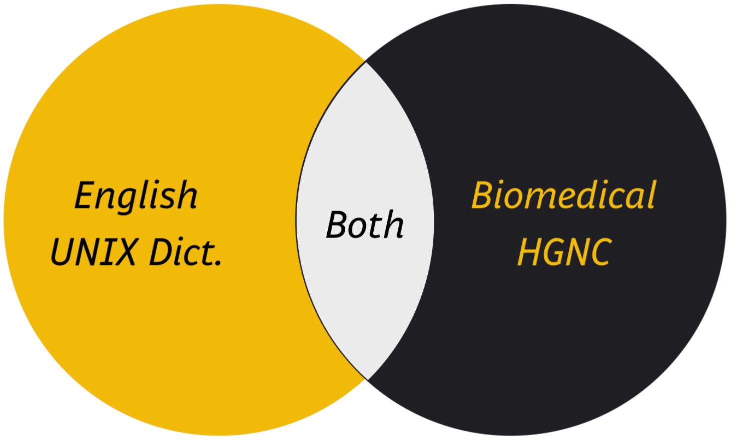
[4] “Clock,” PubMed, Accessed April 1, 2017, https://www.ncbi.nlm.nih.gov/pubmed/?term=clock.

[5] Available at /usr/share/dict/words, https://en.wikipedia.org/wiki/Words_%28Unix%29

[6] Sample in Appendix C, full dictionary available: https://github.com/nelder/word-sense-disambiguation-research/blob/master/data/HGNC_Dict.xml

operations were harnessed using the code in Appendix B which made the comparisons nearly instantly. The resulting set of 166 words is listed in Appendix D and contains words like *grasp*, *was*, and *clock*. With this list of words which are likely to be confused between English and biomedical senses available, a wide number of examples can be assessed to measure how close a particular word is to its expected location.

Figure 2: *Representation of the Dictionary created of Biomedical and English Terms.*



Breaking the Singular Unit:
The vector space model training process begins with randomized word locations which are slowly translated towards their correct locations given each occurrence found in the corpus. With two or more contexts competing to pull this word it is likely the individual instance of this word will be located between the correct locations. To address this issue word sense disambiguation will need to split the word into multiple units which can be independently understood and manipulated during the training process. Several methods could be harnessed to achieve this.

The first approach would involve using a known dictionary to split the singular word units into multiple senses, the challenge would then be to understand which of the senses to translate upon encountering an occurrence in the corpus during the training process.

A second possible approach is focusing on consistent forces in distinct directions adjusting the vector location during the training process. With these shearing forces identified, one could conceivably split the word into multiple senses based upon a certain threshold of opposing forces. This approach is that taken by Adagram, a tool for word sense disambiguation this study will assess.[7]

Measures of Success:
To measure the success of word sense disambiguation an expected and appropriate location in the vector space model must be generated to compare with the actual location of the word in

[7] Kondra, “Adagram.jl,” Github, accessed April 7, 2017, https://github.com/sbos/AdaGram.jl.

vector space. This difference was then measured using the cosine distance to assess if the word was where it was expected to be.

To generate the expected location of a given word, the context cluster approach was employed. The Human Gene Nomenclature Committee data source provided a set of similar words to each gene it supplied. Using the code outlined in Appendix E these synonyms were gathered. The English synonyms were generated using the Natural Language Toolkit and enclosed Wordnet through a Python interface.[8] The code used is included in Appendix F. These were combined into a single file using the code included in Appendix G. This final output is included in Appendix H. Initially the set included 166 words, however a manual pruning reduced this to 116 words. Words with biomedical senses that contained English words, or words with insufficient numbers of context words in their senses were removed. In some instances English word synonyms were added manually when the number in the cluster was not sufficient.

The context cluster method was initially selected because the reality understood by word embedding is entirely dependent on the content of the corpus it is trained on. A lack of examples of clocks which hang on walls will prevent the model from understanding that meaning of the word. The predicted context cluster method was selected because it is corpus independent. The same words are expected of each output, and the only variable modified is the training method between tests.

Initially the context cluster approach was conducted by averaging the cosine distances between the target word and each of the context words. Following this approach, the centroid vector of the context words was calculated and measured to the target word instead. While the difference was not initially understood, these approaches render the same final result: a cosine similarity between the location of a word and a context location it is expected to rest within. This allows a measurement of how close an ambiguous word is to its expected location and thus a measure of the success of word sense disambiguation.

Through the testing procedure, an additional measure of correct placement was developed to enhance the understanding of where the word was located. By querying the vector space model for the 10 nearest neighbours by cosine distance to a particular word the context can be examined as opposed to anticipated and measured. This nearest neighbours approach assesses the appropriateness of the 10 neighbours and the strength of their proximity. A correct location in vector space is indicated by a high similarity, such as 0.8, and neighbouring words that closely match the definition of the target word. A less appropriate location would alternatively be identified by a lower cosine distance, towards 0.5, along with neighbouring words from differing contexts.

The weakness of the nearest neighbours approach is its bias between corpora, to mitigate this only one corpus was focused on. If one corpus has a number of examples of clocks hanging on walls to confuse the clock gene name and another does not, this would affect the results.

[8] “NLTK,” Natural Language Toolkit, Accessed April 1, 2017, http://www.nltk.org/.

PREPARATION:

Motivation for Server Testing:

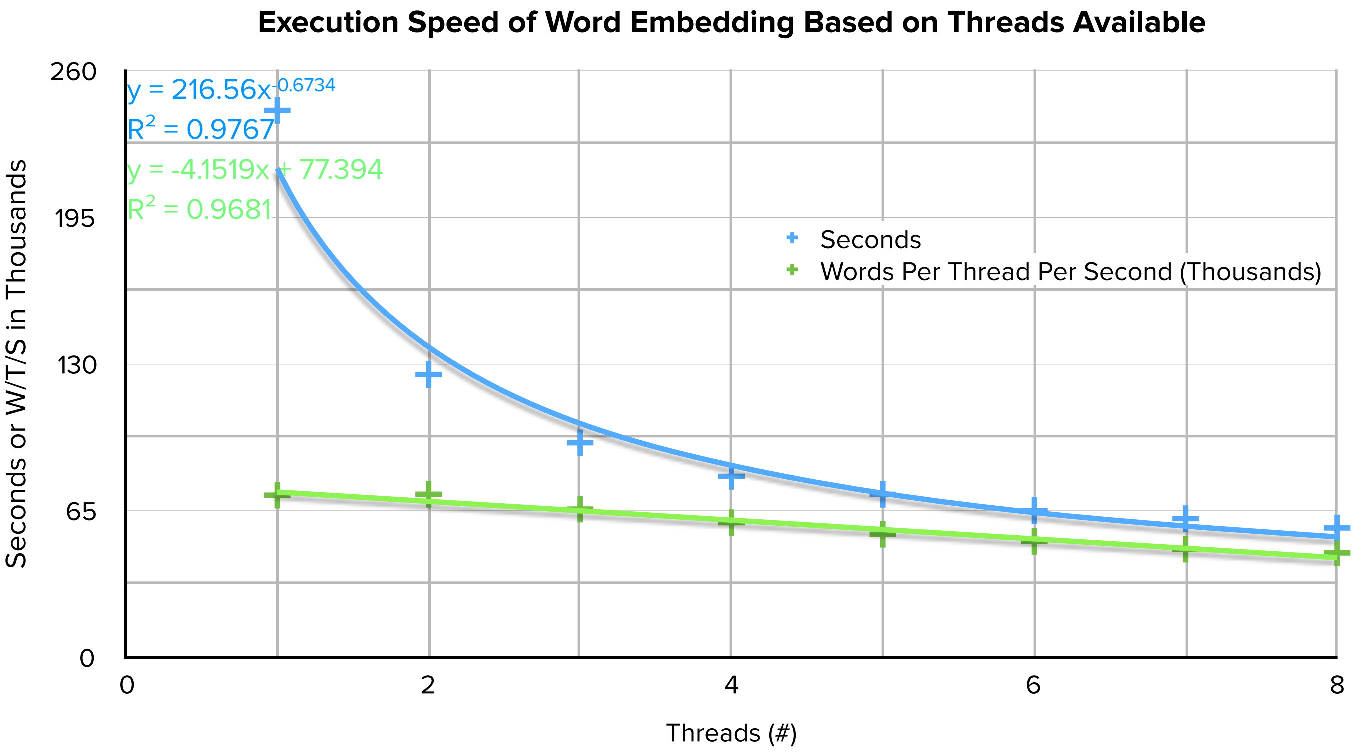
Testing was conducted on the efficiency of a variety of environments capable of hosting word embedding applications. Given the high number of computations required, relatively powerful computers are needed to process even 1GB of data in under an hour. Testing the execution times of Word2Vec[9] allowed for an understanding of the software to be developed. This testing also indicated the optimal platform for future testing.

A variety of CPU and GPU (Graphics Processors) based systems were tested in this study. It was originally theorized that highly parallelized processors like 28 core CPU units would out perform their 4 core cousins, and that GPU units would outperform both of these CPU alternates.

Multithreadding:

On a 2015 MacBook Pro, with an Intel i7 Processor (2.8GHz to 4.0GHz, 8 Threads, 6MB Cache)[10] a Skipgram[11] model test was run for a variety of thread counts which revealed diminishing returns of additional cores made available for the training procedure as illustrated in Chart 1.

Chart 1: Multithreading and Word2Vec Runtime



[9] Dav, "Word2vec," GitHub, July 02, 2016, accessed January 16, 2017, https://github.com/dav/word2vec.

[10] Intel, "Intel® Core™ i7-4980HQ Processor (6M Cache, up to 4.00 GHz) Specifications," Intel ARK (Product Specs), accessed January 15, 2017, https://ark.intel.com/products/83503/Intel-Core-i7-4980HQ-Processor-6M-Cache-up-to-4_00-GHz.

[11] Predicting context given word as opposed to predicting word given context, https://www.quora.com/What-are-the-continuous-bag-of-words-and-skip-gram-architectures-in-laymans-terms.

Despite the diminishing returns, multithreading of the training process achieved an improvement of **4.23x** over the base single threaded rate when using 8 threads. This certainly renders multithreading an important aspect of efficient word embedding applications. Perhaps software designed to take advantage of threads more efficiently could out perform the current benchmarks.

Central Processing Unit Based Platform:
A number of computers from dedicated servers to high end consumer desktops were observed. Each machine ran the same 1GB wikipedia data set.[12] A 300 dimensional space was used, with a window of 5, using the Skipgram model for this test. Below is a table of the platforms investigated.

Table 1: CPU Based Server Platforms Tested and Runtimes using 1GB corpus.

Name	CPU	Clock (GHz)	Threads	Type	Runtime (Seconds)
Softlayer Bare Server (Dual CPU)	Dual Intel Xeon E5-2690 v4	2.6	56	Dedicated Server	711.781
Macbook Pro	Intel i7-4980HQ CPU	2.80 - 4.0	8	Consumer Laptop	773.255
iMac	Intel i5	3.2	4	Consumer Desktop	855.813
Softlayer Bare Server	Intel Xeon Haswell E3-1270v3	3.5	8	Dedicated Server	958.673
Digital Ocean Virtual Server	Intel Xeon E5-2650L v3	1.80	32	Virtualized Server	1029.032

As illustrated in Table 1, the Softlayer-2 server was the fastest, however had to use 56 threads to overcome the benefit of the higher clock speed on the MacBook with 8 threads. Even then it had a runtime that was only **1.08x** faster than the MacBook which had **14%** the number of cores.

Generally speaking high clock speeds determined the fastest platform to run word embedding likely due to the highly interconnected nature of the neural network propagation of values.

In all cases except when two server CPUs were used on one machine (Softlayer-2) consumer CPUs outperformed server CPUs consistently. This is likely due to server CPU units focus on multithreading which is important to a server responding to multiple clients. Whereas a consumer CPU is responding to a single user who expects 0 latency.

Graphics Processing Unit Based Platform:
GPU units are optimized for linear algebra math and thus the calculations used in vector based machine learning applications like Word2Vec. The following results were generated using a GPU

[12] M. Mahoney, "Wikipedia 1GB Data Set," Florida Institute of Technology, accessed January 16, 2017, http://cs.fit.edu/~mmahoney/compression/enwik9.zip.

codebase and hence can not be directly compared to the CPU results.[13] Table 2 outlines the two models tested. With the 100MB test file,[14] the GPU was 2x faster than the CPU unit, however this effect was not consistent with the 1GB data set tested.

Table 2: GPU Server platforms tested with 1GB Corpus. [15] [16]

Name	Short	GPU	Clock (MHz)	Cores	Memory (GB)	Runtime (Seconds)
Softlayer GPU	SL-3	NVIDIA Grid K2	745	1536 x2	4 x2	1234
Research Network	RN	NVIDIA Tesla M2050	575	448	3	2681

The above results are slower than all CPU based tests, which is surprising. It is likely that the code used for this test is inferior to the CPU codebase. While the 100MB test saw 100% usage of one of the GPU units, the 1GB test saw usage hovering between 40% and 60%, indicating sub-optimal use of the resources available.

Due to the above results, further testing was conducted on the MacBook Pro hardware.

TESTING:

Preparing the Data:

The Biomedical corpora used in this study was prepared by a past researcher based on PubMed and Medline data.[17]

Without Word Sense Disambiguation Using Word2Vec :

The Gensim Word2Vec implementation was used because it enabled a simple Python interface to query the vector space models generated.[18] The model was generated using a vector space dimensionality of 300, context window of 5, and the continuous bag of words method as opposed to Skipgram.[19] The other settings were left as the Gensim defaults dictated. The corpus used was the PubMed corpus.

[13] Cudabigdata, "Cudabigdata/word2vec_cuda," GitHub, October 24, 2015, , accessed January 16, 2017, https://github.com/cudabigdata/word2vec_cuda.

[14] M. Mahoney, “Text8 100MB Data Set,” accessed February 3, 2017, http://mattmahoney.net/dc/text8.zip.

[15] "NVIDIA GRID K2," TechPowerUp, accessed January 16, 2017, https://www.techpowerup.com/gpudb/1700/grid-k2.

[16] "NVIDIA Tesla M2050," TechPowerUp, accessed January 16, 2017, https://www.techpowerup.com/gpudb/1534/tesla-m2050.

[17] *Open Access Subset*, accessed July 15, 2009, https://www.ncbi.nlm.nih.gov/pmc/tools/openftlist/.

[18] “Word2Vec,” Gensim, accessed April 1, 2017, https://radimrehurek.com/gensim/models/word2vec.html.

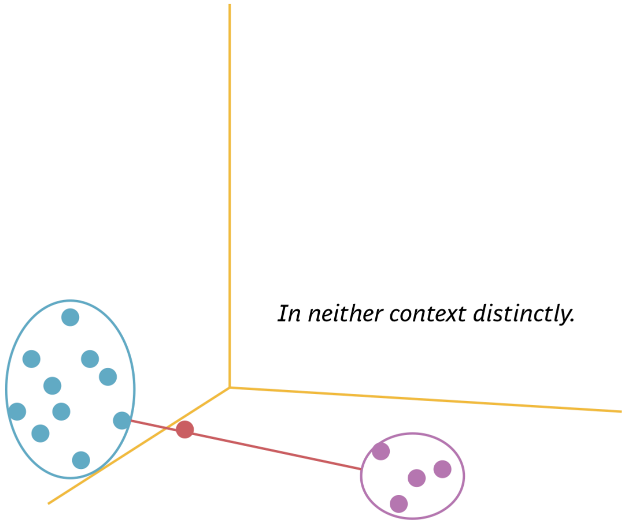
[19] Predicting word given context as opposed to predicting context given word, https://www.quora.com/What-are-the-continuous-bag-of-words-and-skip-gram-architectures-in-laymans-terms.

Word2Vec is an application originally published by Google which implements the word embedding system. When supplied a corpus, Word2Vec generates a vector space model by iterating through every word in the corpus and performing the learning translation described before.

To measure the distance from a word to its biomedical and English context clusters, the centroid measure code using Gensim was developed. A copy is included in Appendix I. The models were generated using the model builder code created by a past researcher and included in Appendix J. Pre-generated models from the past researcher were also used.

On average the ambiguous words had a cosine distance of **0.071** from the biomedical context cluster and a cosine distance of **0.132** from the English context cluster. Given the past success of 1GB Corpora and Word2Vec generating cosine similarities of synonyms in the order of 0.8, the low similarity suggests that these ambiguous words have been pulled into a space between the two correct context clusters and thus do not show high similarity to either cluster. This is illustrated in Figure 3, with the blue cluster roughly representing the English cluster, and the purple cluster representing the biomedical cluster. The red point indicates the ambiguous word. The higher similarity to the English cluster also suggests that there are a larger number of English uses of the ambiguous terms through the PubMed corpus than Biomedical uses, which is not surprising considering the frequency of common English words even in biomedical text.

Figure 3: Ambiguous Term.

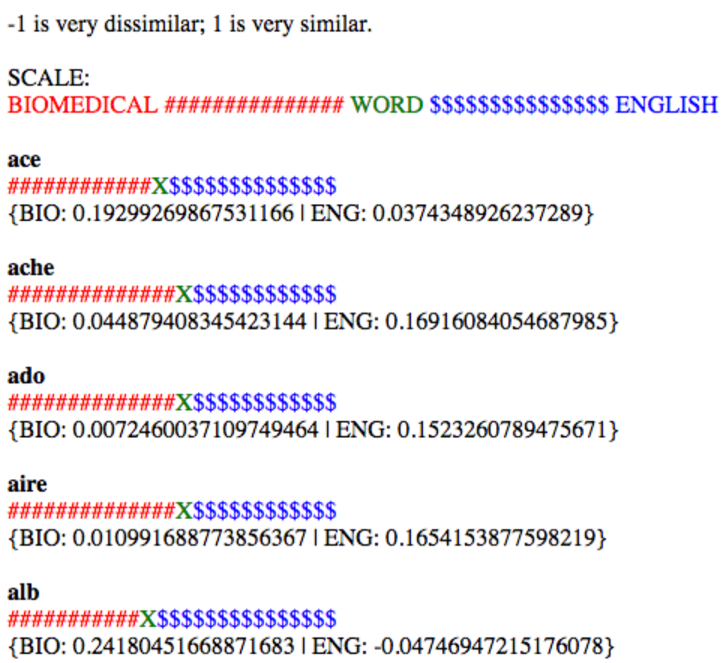


For example the word *ran* was measured in relation to *ras*, *oncogene*, *ara24*, *tc4*, and *gsp1* in the biomedical context and *run*, *scat*, *scarper*, *lam*, *escape*, and more in the English context. The cosine distance to the biomedical context was **-0.117** and the cosine distance to the English context was **0.120**.

The full dataset for the Word2Vec model without word sense disambiguation is included in Appendix L. This includes each word and their distances to the two context clusters as well as the constituent words within each context cluster and their individual distances to the target word.

The results were visualized using a simple web page generated using the cosine distances to each of the context clusters. The code used to generate this web page is included in Appendix K. Some of these results are visualized in Figure 4. The red and blue symbols represent a distance away from a particular context cluster. Overall the number of blue and red symbols represents the dissimilarity of these ambiguous words to their expected context cluster.

Figure 4: Ambiguous Words and Cosine Distances to English and biomedical contexts.[20]



-1 is very dissimilar; 1 is very similar.

SCALE:
BIOMEDICAL ############### WORD $$$$$$$$$$$$$$$ ENGLISH

ace
############X$$$$$$$$$$$$$$
{BIO: 0.19299269867531166 | ENG: 0.0374348926237289}

ache
##############X$$$$$$$$$$$$
{BIO: 0.044879408345423144 | ENG: 0.16916084054687985}

ado
##############X$$$$$$$$$$$$
{BIO: 0.0072460037109749464 | ENG: 0.1523260789475671}

aire
##############X$$$$$$$$$$$$
{BIO: 0.010991688773856367 | ENG: 0.1654153877598219}

alb
###########X$$$$$$$$$$$$$$$
{BIO: 0.24180451668871683 | ENG: -0.04746947215176078}

With Word Sense Disambiguation Using Adagram:
Word sense disambiguation aims to split the individual words into multiple senses based upon the number of distinct meanings they represent. Adagram achieves this by breaking words into a number of senses when the training process pulls them in distinct directions. These multiple units however complicate the context cluster method of measuring successful disambiguation. While the context cluster method was automated, a review of their nearest neighbours was added to enhance what is understood about the new contexts of the multiple senses.

Whereas Gensim was used to conduct the testing with Word2Vec, Julia and Adagram were used to test the code with word sense disambiguation available.[21] [22]

[20] Nicholas Elder, "Ambiguous pubMed Data Visualization," Github, accessed April 1, 2017, https://nelder.github.io/word-sense-disambiguation-research/pubMedWord2Vec.html

[21] Julia, accessed April 2, 2017, https://julialang.org/.

[22] Adagram, accessed April 2, 2017, https://github.com/sbos/AdaGram.jl.

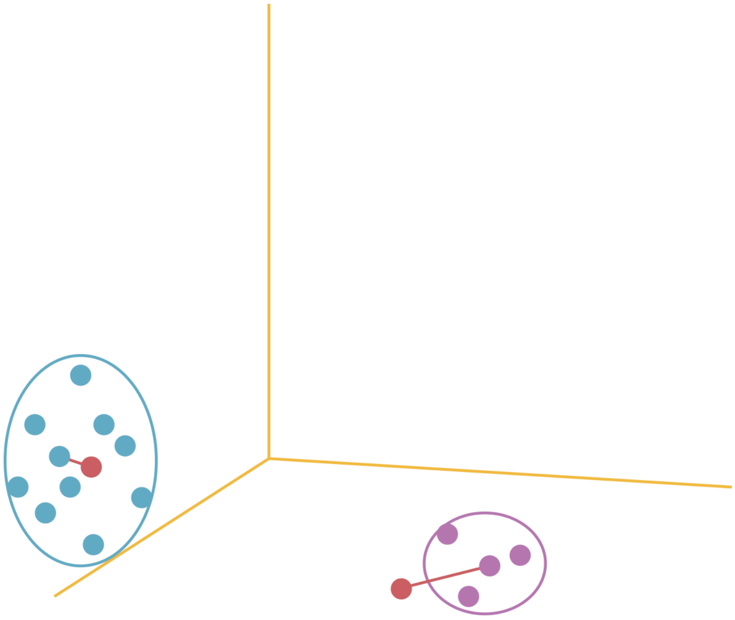
Evaluation of Results: Context Cluster for Adagram:
The word-sense-disambiguation-enabled Adagram was tested with the context clusters used to test Word2Vec. To conduct this testing a modified version of the context clusters code was used and is included in Appendix M. To test the correct automatically generated sense of the target word to measure to the context cluster, the number of integer characters within the 10 nearest neighbours was selected as a guiding characteristic. Given the biomedical sense typically is surrounded by genes which often have integers in their names, the biomedical sense was selected on this basis. This selection method was manually confirmed and performed accurately. The English sense selection was the remaining sense with the highest average cosine distance to its 10 nearest neighbours.

On average the disambiguated biomedical sense had a cosine distance of **0.206** from the biomedical context cluster and a cosine distance of **0.068** from the English context cluster. The cosine similarity between the biomedical sense and the biomedical context cluster was higher than the distance between the biomedical sense and the english cluster for **78.7%** of the words.

On average the disambiguated English sense had a cosine distance of **0.116** from the English context cluster and a cosine distance of **0.084** from the biomedical context cluster. The cosine similarity between the English sense and the English context cluster was higher than the distance between the English sense and the biomedical cluster for **60.0%** of the words.

As illustrated in Figure 5, the ambiguous word was split into multiple senses which were closer to their anticipated context cluster than their incorrect context cluster in the majority of cases.

Figure 5: Example of disambiguated word in vector space.



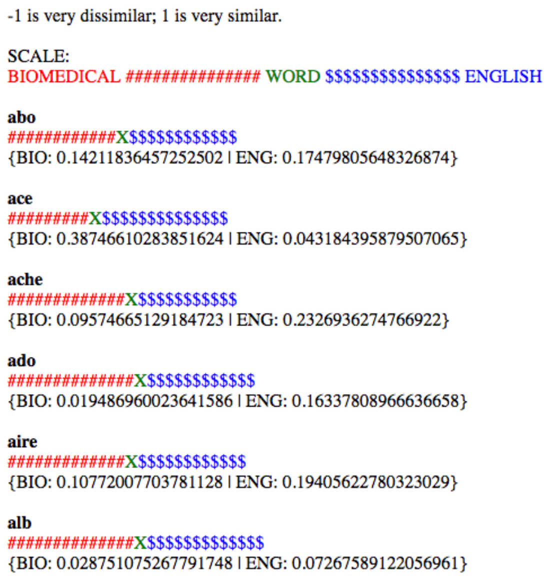
Typically the word sense disambiguation enabled tool was capable of differentiating between the biomedical and English words in the majority of cases. The correct sense on average had a higher cosine similarity to its context cluster whereas without word sense disambiguation the biomedical sense was not identified with a positive cosine similarity on average.

While direct comparisons between word embedding models is complicated by variations between their methodology and by random differences, the improvement from 0.071 with Word2Vec to 0.206 with Adagram with respect to proximity of the word to the biomedical sense suggests that the disambiguation process has improved the accuracy. The slight decline for the English cluster from 0.132 to 0.116 may very well be within the random uncertainty of the model, though this is not for certain.

While these results suggest the hypothesis was correct, they were challenged by the strength of the biomedical context cluster anticipated and the presence of the English synonyms in medical journal articles. These two factors resulted in sub-optimal context clusters and thus sub-optimal anticipated locations for the word sense. The biomedical context cluster obtained from the Human Gene Nomenclature Committee list often contained words that were not in the biomedical corpus and often included the components of the gene name which sometimes were English words. This may have resulted in the biomedical context cluster containing English words or not sufficiently covering the correct context of the biomedical sense of the word.

These results can be visualized in Figure 6 and Figure 7 in which the disambiguated English senses show a tendency towards a higher cosine similarity with the English context cluster and the biomedical senses show a tendency towards a higher cosine similarity with the biomedical context cluster.

Figure 6: Disambiguated English Senses and their Distance to Biomedical and English Context Clusters.[23] [24]

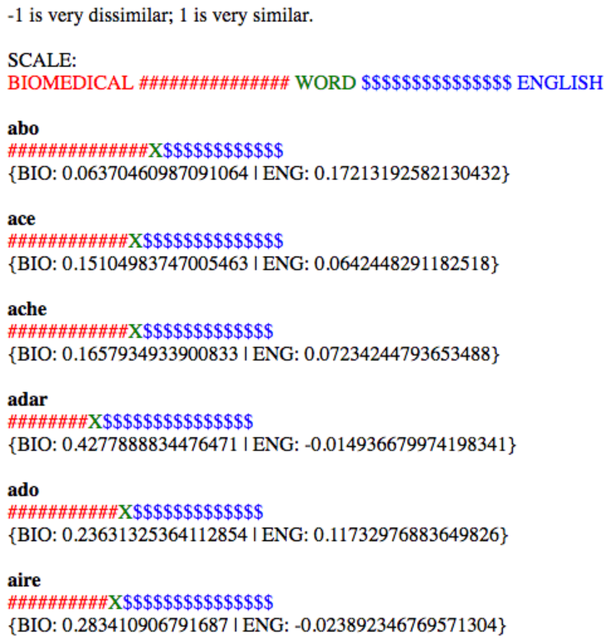


```
-1 is very dissimilar; 1 is very similar.

SCALE:
BIOMEDICAL ############### WORD $$$$$$$$$$$$$$$$ ENGLISH

abo
############X$$$$$$$$$$$$$
{BIO: 0.14211836457252502 | ENG: 0.17479805648326874}

ace
#########X$$$$$$$$$$$$$$$
{BIO: 0.38746610283851624 | ENG: 0.043184395879507065}

ache
#############X$$$$$$$$$$$$
{BIO: 0.09574665129184723 | ENG: 0.2326936274766922}

ado
##############X$$$$$$$$$$$$$
{BIO: 0.019486960023641586 | ENG: 0.16337808966636658}

aire
#############X$$$$$$$$$$$$$
{BIO: 0.10772007703781128 | ENG: 0.19405622780323029}

alb
##############X$$$$$$$$$$$$$$
{BIO: 0.028751075267791748 | ENG: 0.07267589122056961}
```

[23] Nicholas Elder, “Disambiguated pubMed English Sense,” Github, accessed April 2, 2017 https://github.com/nelder/word-sense-disambiguation-research/blob/master/data/pubMedDisambEng.json

[24] Nicholas Elder, “Disambiguated pubMed English Sense,” Github, accessed April 2, 2017 https://nelder.github.io/word-sense-disambiguation-research/ADApubMedENG.html

Figure 7: Disambiguated Biomedical Senses and their Distance to Biomedical and English Context Clusters.[25] [26]

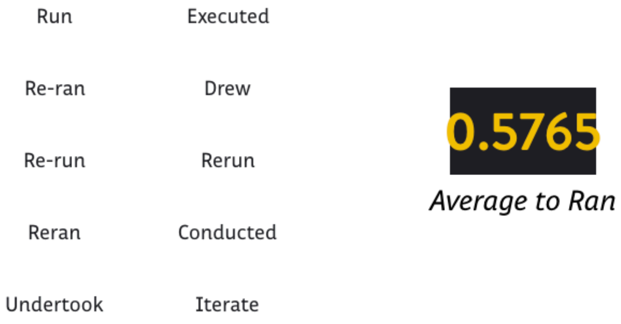


```
-1 is very dissimilar; 1 is very similar.

SCALE:
BIOMEDICAL ############### WORD $$$$$$$$$$$$$$$ ENGLISH

abo
##############X$$$$$$$$$$$$
{BIO: 0.06370460987091064 | ENG: 0.17213192582130432}

ace
############X$$$$$$$$$$$$$$
{BIO: 0.15104983747005463 | ENG: 0.0642448291182518}

ache
############X$$$$$$$$$$$$$
{BIO: 0.1657934933900833 | ENG: 0.07234244793653488}

adar
########X$$$$$$$$$$$$$$$
{BIO: 0.4277888834476471 | ENG: -0.014936679974198341}

ado
###########X$$$$$$$$$$$$$
{BIO: 0.23631325364112854 | ENG: 0.11732976883649826}

aire
##########X$$$$$$$$$$$$$$$
{BIO: 0.283410906791687 | ENG: -0.023892346769571304}
```

Evaluation of Results: Nearest Neighbours for Adagram:
Given the limitations of predicting the correct context cluster for each word, an analysis of the nearest neighbours of a word can, from the opposite direction, assess the appropriateness of the word's location in the vector space model.

Figures 8 through 10 outline the nature of the nearest neighbours for the word *ran*.

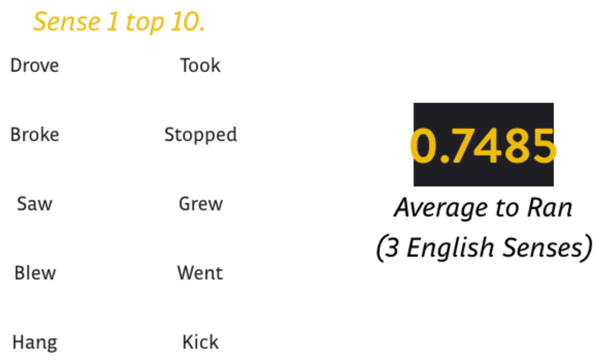
Figure 8: Nearest Neighbours of Ambiguous Word Ran.



[25] Nicholas Elder, "Disambiguated pubMed Biomedical Sense," Github, accessed April 2, 2017 https://github.com/nelder/word-sense-disambiguation-research/blob/master/data/pubMedDisambBio.json

[26] Nicholas Elder, "Disambiguated pubMed Biomedical Sense," Github, accessed April 2, 2017 https://nelder.github.io/word-sense-disambiguation-research/ADApubMedBIO.html

Figure 9: Nearest Neighbours of Disambiguated Word Ran in the English Sense



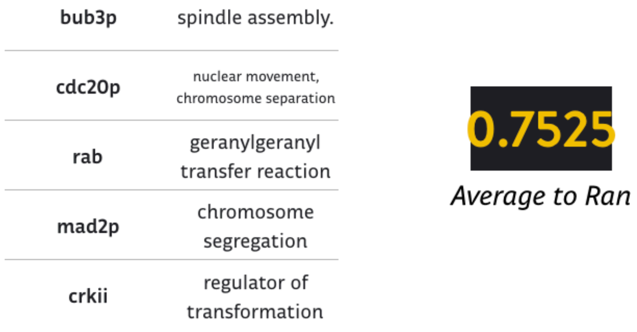
Sense 1 top 10.

Drove	Took
Broke	Stopped
Saw	Grew
Blew	Went
Hang	Kick

0.7485

Average to Ran
(3 English Senses)

Figure 10: Nearest Neighbours of Disambiguated Word Ran in the Biomedical Sense



bub3p	spindle assembly.
cdc20p	nuclear movement, chromosome separation
rab	geranylgeranyl transfer reaction
mad2p	chromosome segregation
crkii	regulator of transformation

0.7525

Average to Ran

Figure 8 outlines the nearest neighbours of *ran* in Word2Vec without a word sense disambiguation generated model. The average cosine distance between the nearest neighbours and ran was **0.5765** and despite the training using the PubMed corpus, there are none of the related genes present in the context cluster.

Figure 9 shows one of the three English sense automatically detected using Adagram. The average cosine distance between the nearest neighbours and this sense of ran was **0.7485**.

Figure 10 outlines the biomedical senses of the the word *ran* after word sense disambiguation. The average distance between the nearest neighbours and this sense of ran is **0.7525**. The biomedical definition of the word *ran* relates to a GTP binding protein which is involved in the control of DNA synthesis and cell cycle progression.[27] While the nearest neighbours of the biomedical sense of *ran* appear to be genes, it is significant that these genes are generally related specifically to the gene *ran* and not just to genes in general. *Cdc20p* for example is related to the process of chromosome separation which is an aspect of the cell cycle progression.

[27] “Ran Gene,” Gene Cards, Accessed April 2, 2017, http://www.genecards.org/cgi-bin/carddisp.pl?gene=RAN&keywords=ran.

As a number of additional disambiguated gene names demonstrated a similar progression, an analysis of the strength of the biomedical and English sense average cosine distance to their respective 10 nearest neighbours for the 116 known ambiguous words was conducted and revealed strong results with respect to cosine distances. The average cosine distance between the biomedical sense of each word and its 10 nearest neighbours was **0.719** and the average cosine distance between the English sense of each word and its 10 nearest neighbours was **0.664** which indicates relatively high similarity, as typical strong one to one synonym pairs experience cosine distances in the range of 0.8. This data is available in Appendix N. The code used to generate this data is included in Appendix O.

Conclusion:

Overall these findings indicate that while a tool without word sense disambiguation can functional relatively well within an English corpus, when there are significant differences between word senses it is not adequate to guarantee expected results. One such situation is medical journals in which gene names can also be English words.

By using Adagram, a tool with word sense disambiguation enabled, the vector space model from word embedding experiences what these results suggest to be improved placement relative to their expected location over Word2Vec, a tool without word sense disambiguation enabled.

With word sense disambiguation indicating the correct context cluster in the majority of cases, and what seems to be improved nearest neighbours, it seems likely that word sense disambiguation assists in enabling the word embedding model to represent ambiguous words accurately by breaking them into independent units.

Future Work:

While these results suggest the hypothesis was accurate, further testing can assist in improving that likelihood. An analysis of the meaning of the words in the nearest neighbours to each sense may clarify the word’s location in space. While a manual check of the gene name nearest neighbours was conducted for a number of genes, a manual confirmation of the 116 gene names relation to their biomedical nearest neighbours could strengthen these findings. A stronger context cluster data set may also yield stronger results and better predict the expected location of a given word in vector space. In addition, an examination of the positioning between the ambiguous words themselves would help better visualize how the vector space is affected. Finally a visualization of these vector space models may reveal additional information worthwhile examining.

BIBLIOGRAPHY:

Bartunov, Sergey. *Breaking Sticks and Ambiguities with Adaptive Skip-Gram.* National Research University Higher School of Economics, 2015.

Bezanson, Jeff. *Julia Language*. *https://julialang.org/*

Gene Cards. *www.genecards.org/*

Mikolov, Thomas. *Distributed Representations of Words and Phrases and their Compositionality.* Google, 2013.

Natural Language Toolkit, Accessed April 1, 2017, http://www.nltk.org/.

Rehurek, Radim. *Gensim Software*, 2017. http://radimrehurek.com/gensim/.

Appendix A: Python Bio-English Intersect Code, Nested Loop.

```
import xml.etree.ElementTree
e = xml.etree.ElementTree.parse('../../Var/HGNC_Dict.xml').getroot()

#Load the english words
with open('../../Var/words.txt', 'r') as file:
    for line in file:

        #print(line.rstrip("\n").lower())


        #For each of the names in the biomed corpus
        for atype in e.findall('token'):
            #print(atype.attrib['canonical'].lower())

            if(atype.attrib['canonical'].lower() ==
line.rstrip(“\n").lower()):

                print(line.rstrip(“\n").lower())
```

Appendix B: Python Bio-English Intersect Code, Set Operation.

```
import xml.etree.ElementTree
e = xml.etree.ElementTree.parse('../../Var/HGNC_Dict.xml').getroot()

a = []
b = []

#Load the english words
with open('../../Var/words.txt', 'r') as file:
    for line in file:

        a.append(line.rstrip("\n").lower())


#For each of the names in the biomed corpus
for atype in e.findall('token'):
    b.append(atype.attrib['canonical'].lower())

print(set(a) & set(b))
```

Appendix C: Biomedical Human Gene Nomenclature Committee Synonyms and Dictionary.

```
<?xml version="1.0" encoding="UTF-8" ?>
<synonym>
 <token canonical="A1BG">
   <variant base="A1BG"/>
   <variant base="alpha-1-B glycoprotein"/>
 </token>
 <token canonical="A2M">
   <variant base="A2M"/>
   <variant base="alpha-2-macroglobulin"/>
   <variant base="FWP007"/>
   <variant base="S863-7"/>
   <variant base="CPAMD5"/>
 </token>
 <token canonical="A2MP1">
   <variant base="A2MP1"/>
   <variant base="alpha-2-macroglobulin pseudogene 1"/>
   <variant base="A2MP"/>
 </token>
```

Appendix D: Intersect Dictionary of Biomedical and English Words.

```
{'styx', 'grasp', 'erg', 'amt', 'mall', 'arx', 'timeless', 'pam',
'hal', 'chat', 'mina', 'mars', 'ager', 'arc', 'marco', 'large',
'ache', 'ell', 'nama', 'bad', 'pir', 'oscar', 'penk', 'lif', 'prune',
'ada', 'skil', 'sell', 'aire', 'kit', 'napa', 'numb', 'aga', 'lat',
'wac', 'tub', 'ran', 'sri', 'plat', 'pip', 'lox', 'pry', 'dak',
'scap', 'ace', 'cit', 'jun', 'gan', 'ust', 'rel', 'gem', 'pomp',
'vip', 'clam', 'strap', 'chad', 'cepa', 'cad', 'tymp', 'ilk', 'kin',
'tars', 'gif', 'nog', 'th', 'saraf', 'mal', 'palm', 'boll', 'vim',
'alb', 'tec', 'atrip', 'reck', 'bora', 'dap', 'coil', 'tat', 'met',
'max', 'naga', 'copa', 'dao', 'gip', 'clock', 'camp', 'nodal', 'itch',
'mice', 'star', 'hexa', 'rax', 'oat', 'gype', 'vill', 'ide', 'lias',
'sync', 'was', 'corin', 'enam', 'son', 'set', 'prep', 'cast', 'tank',
'sea', 'hunk', 'pah', 'neb', 'abo', 'melas', 'impact', 'cop', 'si',
'musk', 'notum', 'wiz', 'galp', 'sele', 'tst', 'zan', 'ar', 'cask',
'lipa', 'mica', 'arse', 'themis', 'sag', 'alk', 'cat', 'ret', 'ado',
'cope', 'aes', 'gale', 'auh', 'pole', 'oaf', 'fuse', 'galt', 'she',
'mog', 'lum', 't', 'apod', 'kras', 'bid', 'tra', 'trio', 'sla',
'stam', 'rho', 'gast', 'mag', 'rest', 'poll', 'tox', 'lars', 'gal',
'ski', 'adar', 'fry', 'polk', 'chia', 'ankh'}
```

Appendix E: Expanded Biomedical Dictionary Generator with Synonyms.

```
import xml.etree.ElementTree
import json
e = xml.etree.ElementTree.parse('../../Var/HGNC_Dict.xml').getroot()
x = xml.etree.ElementTree.parse('../../Var/HGNC_Dict.xml').getroot()

a = []
b = []

#Load the english words
with open('../../Var/words.txt', 'r') as file:
    for line in file:

        a.append(line.rstrip("\n").lower())
```

```
#For each of the names in the biomed corpus
for atype in e.findall('token'):
    b.append(atype.attrib['canonical'].lower())

result = set(a) & set(b)
result = dict.fromkeys(result, [])



#For each of the names in the biomed corpus
output = {}
for atype in x.findall('token'):

    #For each variant
    for child in atype.findall('variant'):

        #Check if in dictionary and add each word in list
        if(atype.attrib['canonical'].lower() in result):

            #Split by word
            word_arr = ""
            word_arr = child.attrib['base'].lower()
            word_arr = word_arr.split()
            temp = []

            if atype.attrib['canonical'].lower() not in output:
                output[atype.attrib['canonical'].lower()]=[]

            for word in word_arr:
                if word != atype.attrib['canonical'].lower():

output[atype.attrib['canonical'].lower()].append(word)


out = open("../../Var/out.txt", 'w')
out.write(json.dumps(output, sort_keys=True, indent=2))
```

Appendix F: English Synonyms with Wordnet Generator.

```
#Libraries
from nltk.corpus import wordnet
import nltk
import json

#Generate an array with every synonym to a word
def engSyns(target):
     words = []
     sets = wordnet.synsets(target)
     for st in sets:
           for word in st.lemma_names():
                if word.lower() not in words and word.lower() !=
target:
                    words.append(word.lower())
    return words


#Fetch Biomedical Corpus
with open('../../Var/dump.txt') as data_file:
    bio = json.load(data_file)


#Output storage
output = {}

#For each ambigious term, get syns and store in output
for term in bio:
    output[term] = []
    english = engSyns(term)
    for word in english:
        if word not in bio[term]:
            output[term].append(word)

#Dump output to file
out = open("../../Var/ambigiousEnglishTermsWSyns.json", 'w')
out.write(json.dumps(output, sort_keys=True, indent=2))
```

Appendix G: English and Biomedical Context Cluster Combination Generator.

```
#Import Libraries
import json

#Load Data Files
with open('../../Var/ambigiousBiomedicalTermsWSyns.json') as
data_file:
    bio = json.load(data_file)

with open('../../Var/ambigiousEnglishTermsWSyns.json') as data_file:
    english = json.load(data_file)


#output
output = {}
for word in bio:
    output[word] = []
    output[word].append(bio[word])
    if len(english[word]) > 0:
        output[word].append(english[word])

#Dump output to file
out = open("../../Var/ambigiousTermsUnionSet.json", 'w')
out.write(json.dumps(output, sort_keys=True, indent=2))
```

Appendix H: English and Biomedical Terms and their Synonyms.

The following is a sample of the full data set. The entire set can be accessed here: https://github.com/nelder/word-sense-disambiguation-research/blob/master/data/ambigious_words.txt

The word is followed by its biomedical context. A comma then separates the English context from the biomedical context.

```
palm:paralemmin kiaa0270,thenar palm_tree decoration laurel_wreath
medal medallion ribbon handle

wiz:widely_interspaced_zinc znf803,ace adept champion sensation maven
mavin virtuoso genius hotshot star superstar whiz whizz wizard

napa:n-ethylmaleimide-sensitive factor attachment alpha alpha
snap,chinese_cabbage celery_cabbage pe-tsai brassica_rapa_pekinensis

pry:ptpn13-like y-linked ptpn13ly pry1,crowbar wrecking_bar pry_bar
prise prize lever jimmy intrude horn_in nose poke

numb:homolog drosophila c14orf41,benumb blunt dull asleep benumbed
dead

dak:dihydroxyacetone kinase 2 homolog s. cerevisiae dkfzp586b1621
net45,dhak palas butea_frondosa butea_monosperma

gal:galanin/gmap prepropeptide galn gmap gal-gmap glnn galanin-
message-associated peptide galanin/gmap prepropeptide,gallon girl
female woman

sync:syncoilin intermediate filament sync1 syncoilin,synchronize
synchronise

tat:tyrosine aminotransferase,cheapness tackiness sleaze
thematic_apperception_test intertwine

mag:myelin associated glycogma siglec4a siglec-4a s-mag sialic acid
binding ig-like lectin 4a,magazine

son:dna binding c21orf50 dbp-5 nrebp kiaa1019 bass1 flj21099 flj33914
nre-binding saccharomyces 1,boy word logos
```

Appendix I: Context Centroid to Word Measure Python Code

```
# import modules and set up logging
import json
import argparse
from gensim.models import word2vec
from gensim.models.word2vec import FAST_VERSION
assert FAST_VERSION > -1
import logging
import scipy.spatial.distance
import numpy as np
logging.basicConfig(format='%(asctime)s : %(levelname)s : %
(message)s', level=logging.INFO)

#import pprint
#pp = pprint.PrettyPrinter(indent=2)   #pp.pprint(inputData)

#take command line arguments
parser = argparse.ArgumentParser()
parser.add_argument('-i', help='input file formated: word:cluster
words,other cluster. One term to be tested on each line.',
required=True, metavar='Input')
parser.add_argument('-o', help='output file in JSON format.',
required=True, metavar='Output')
parser.add_argument('-m', help='input Gensim VSM file (.model)',
required=True, metavar='Model')
parser.add_argument('-r', help='raw output if 1 specified.
(Optional)', metavar='0 or 1')

args = parser.parse_args()

#load model to memory
model = word2vec.Word2Vec.load(args.m)

#disable training, trim unneeded model memory = use (much) less RAM.
model.init_sims(replace=True)

#prepare data in and out
output = {}
inputData = []

#data import from file (http://www.diveintopython.net/
scripts_and_streams/command_line_arguments.html)
with open(args.i) as file:

      #For each of the lines
      inCounter = -1
      for line in file:
            inCounter = inCounter + 1
            if len(line) > 1:

                  #process each line
                  new = line.rstrip('\n')
                  temp = new.split(":")
                  word = temp[0]
                  contexts_raw = temp[1]
                  contexts = contexts_raw.split(",")

                  inputData.append([word,[]])

                  for content in contexts:
                        clusterList = content.split(" ")
```

```
                inputData[inCounter][1].append(clusterList)


#print(json.dumps(inputData, sort_keys=True, indent=2))


#For each ambigious word and its respective clusters
for data in inputData:

    ##For each ambigious clusters
    output[data[0]] = []
    counter = -1
    if(data[0] == 'ar'):
        print((model[data[0]]).tolist())
    for cluster in data[1]:

        #Update current cluster counter
        counter = counter + 1

        ##Add data for this cluster in
        output[data[0]].append([0, cluster])

        ##For each word in the cluster

        wordCounter = -1
        vector = []


        for word in cluster:

            #Update current word counter
            wordCounter = wordCounter + 1

            #Query and store word distance.
            distance = 0


            try:
                distance =  model.similarity(data[0], word)
                vector.append(model[word])


            except Exception:
                pass

            output[data[0]][counter][1][wordCounter] =
[output[data[0]][counter][1][wordCounter], distance]


        #For each cluster, calculate the distance from the centroid
of the words to the target
        try:
            #Average vectors
            vector = np.matrix(np.array(vector))
```

```
                centroid = np.mean(vector, axis=0)

                #Centroid as list
                centroid_list = []
                for num in centroid.flat:
                     centroid_list.append(num)

                #Target List
                target_list = []
                for num in model[data[0]]:
                     target_list.append(num)

                if(data[0] == 'ar'):
                     print("nick:")
                     print(target_list)


                distance = 0
                for i in range(0,len(target_list)):
                     distance = distance + target_list[i] *
centroid_list[i]
                output[data[0]][counter][0] = distance
           except Exception:
                pass
```

Appendix J: Gensim Word2Vec Implementation

```
#Dump results to file
out = open(args.o, 'w')

if(args.r == '1'):
     out.write(json.dumps(output))
     print("\nOutput has been written to "+args.o + " in raw dump
mode.")
else:
     out.write(json.dumps(output, sort_keys=True, indent=2))
     print("\nOutput has been written to "+args.o + " in formatted
dump mode.")


# import modules and set up logging
from gensim.models import word2vec
import logging
#logging.basicConfig(format='%(asctime)s : %(levelname)s : %
(message)s', level=logging.INFO)

# load up preprocessed Corpus from
sentences = word2vec.LineSentence('../Corpus/text8.txt')

# train the skip-gram model; default window=5
model = word2vec.Word2Vec(sentences, size=300, sg=0, workers=32)
```

```
# pickle the entire model to disk, so we can load&resume training
later
model.save('../Var/english_model_out.model')
```

Appendix K: View Cluster Code to Visualize Word to Context Cluster Distance

```
import json
import argparse

#Fetch file arguments.
parser = argparse.ArgumentParser()
parser.add_argument('-i', help='input json cluster file.',
required=True, metavar='Input')
parser.add_argument('-o', help='HTML output file', required=True,
metavar='Output')
args = parser.parse_args()

#Globals
width = 30
sa = "#"
sb = "$"
sw = "X"

#Elder Score
e_count = 0
e_total = 0

#Output file
target = open(args.o, 'w')
target.write("<html>-1 is very dissimilar; 1 is very
similar.<br><br>SCALE:<br><span style='color:red'>BIOMEDICAL
###############</span><span style='color:green'> WORD </span><span
style='color:blue'>$$$$$$$$$$$$$$$ ENGLISH</span><br><br>")

#Open File
with open(args.i) as file:

      #Parse File
      data = json.load(file)

      #Grab relavant data
      for key, word in sorted(data.items()):

           if len(word) == 2:
                english = word[1][0] + 1
           else:
                english = 0

           biomed = word[0][0] + 1
           if biomed != 0 and english != 0 and biomed != 1 and english 
!= 1:
                target.write("<b>"+key +"</b><br><span 
style='color:red'>")

                num = -(width/2)*abs(biomed)+width
                for count in range(0,int(num)):
                     target.write(sa)

                target.write("</span><span style='color:green'>" + sw 
+ "</span><span style='color:blue'>")

                num_b = -(width/2)*abs(english)+width
                for count in range(0,int(num_b)):
                     target.write(sb)

                target.write("</span><br>{BIO: " + str(biomed-1) + " | 
ENG: " + str(english-1) +"}")
                target.write("<br><br>")

                #stregnth of polarization
                diff = abs(num - num_b)
                scaled_diff = diff / width
                e_count = e_count + 1
                e_total = e_total + scaled_diff

e_score = e_total / e_count
target.write("SCORE: "+ str(e_score))


target.write(“</html>")
```

Appendix L: Context Cluster Results Without Word Sense Disambiguation

The following is a sample of the full data set. The entire set can be accessed here: https://github.com/nelder/word-sense-disambiguation-research/blob/master/data/pubMedWord2Vec.json

Each word is followed by the biomedical then English context cluster information. Each cluster has an average and its constituent words and their distances listed. A distance of 0 indicates that the word was not present in the word embedding model for measurement.

```
  "rho": [
    [
      0.07497305333652093,
      [
        [
          "rhodopsin",
          0.054957124710694666
        ],
        [
          "rp4",
          0
        ],
        [
          "opn2",
          0
        ],
        [
          "csnbad1",
          0
        ],
        [
          "opsin",
          0.09498897788577168
        ]
      ]
    ],
    [
      -0.07055934640447958,
      [
        [
          "letter",
          -0.08792609516383294
        ],
        [
          "constellation",
          -0.05319260338332323
        ]
      ]
    ]
  ],
  "ski": [
    [
      -0.09658380216501428,
      [
        [
          "proto-oncogene",
          -0.09658379765845523
        ]
      ]
    ],
    [
      0.25471852070037926,
      [
        [
          "sports",
          0.40719411749680895
        ],
        [
          "skiing",
          0.460024077481316
        ],
        [
          "down_hill",
          0
        ],
        [
          "cross",
          0.04998577937219084
        ],
        [
          "country",
          0.25823737330185736
        ],
        [
          "pole",
          0.09815129060648624
        ]
      ]
    ]
  ]
```

Appendix M: Context Cluster Python Code Modified for Multiple Word Senses

```
#Import needed modules
import JSON

#Reset Globals
global inCount = 0

#Read File
file = "/Users/nick/Dropbox/School/Scholars\ Research/VSM\ Workspace/
julia/ambigious_words.txt"

#Globals
inputData = []

#Parse each line of a file
open(file,"r") do f
    for line in eachline(f)
        global inCount = inCount + 1

        #For every ambigious word:
        if length(line) > 1
            new = replace(line, "\n", "")
            temp = split(new, ":")
            word = temp[1]
            contexts_raw = temp[2]
            contexts = split(contexts_raw, ",")

            push!(inputData,[word,[]])

            for content in contexts
                clusterList = split(content, " ")
                push!(inputData[inCount][2],clusterList)
            end
        end
    end
```

```
end


output = Dict();

for data in inputData
    output[data[1]] = []
    counter = 0

    ambig_word = data[1]

    max_count_cluster = 0
    nu_max_count_cluster = 0

    try
        ambig_word_senses = expected_pi(vm, dict.word2id[ambig_word])
        println(ambig_word)

        #Try to guess which is the bio sense
        co = 0

        max_count = 0
        max_count_cluster = 0

        for i in ambig_word_senses
            co = co + 1
            if i > 0.001
                cluster = nearest_neighbors(vm, dict, ambig_word, co,
10)
                numz = 0

                for wrd in cluster
                    interest = wrd[1]
                    numz = numz + length(matchall(r"0", interest))
                    numz = numz + length(matchall(r"1", interest))
                    numz = numz + length(matchall(r"2", interest))
                    numz = numz + length(matchall(r"3", interest))
                    numz = numz + length(matchall(r"4", interest))
                    numz = numz + length(matchall(r"5", interest))
                    numz = numz + length(matchall(r"6", interest))
                    numz = numz + length(matchall(r"7", interest))
                    numz = numz + length(matchall(r"8", interest))
                    numz = numz + length(matchall(r"9", interest))

                end

                if numz > max_count
                    max_count = numz
                    max_count_cluster = co
                end
                #println(cluster[1][1])
            end
        end
        max_count = 0
        nu_max_count_cluster = 0
        co = 0

        for i in ambig_word_senses
            co = co + 1
            if i > 0.001
                cluster = nearest_neighbors(vm, dict, ambig_word, co,
10)
                numz = 0

                numz = cluster[1][3]

                if numz > max_count
                    if co != max_count_cluster
                        max_count = numz
                        nu_max_count_cluster = co
                    end
                end
                #println(cluster[1][1])
            end
        end



    end

    println(max_count_cluster)
    println(nu_max_count_cluster)




    for cluster in data[2]


        counter = counter + 1

        push!(output[data[1]], [0, cluster])

        wordCounter = 1
        vector = []

        tot_cosine = 0
        num_cosine = 0

        for word in cluster

            try
                #Pull number of senses
                senses = expected_pi(vm, dict.word2id[word])

                num_senses = 0
                cosine = 0
                for i in senses
                    if i>0.001

                        #Count number of senses appropriate
                        num_senses = num_senses + 1

                        #For each of these word senses that is 
appropriate
                        cosine = cosine + dot(vec(vm, 
dict.word2id[ambig_word], nu_max_count_cluster), vec(vm, 
dict.word2id[word], num_senses))


                    end
                end

                #Average cosine distance to context cluster
                cosine = cosine / num_senses
                tot_cosine = tot_cosine + cosine
                num_cosine = num_cosine + 1





            catch e
            end



        end

        #Average each cluster

        output[data[1]][counter][1] = tot_cosine/num_cosine

    end

end

file = “/Users/file.json"
open(file, "w") do f
    write(f, JSON.json(output))
end
```

Appendix N: Nearest Neighbours Cosine Similarity with Word Sense Disambiguation.

The following is a sample of the full data set. The entire set can be accessed here: https://github.com/nelder/word-sense-disambiguation-research/blob/master/data/nn.json

Each word is followed by the biomedical then English context cluster information. Each cluster has an average and its constituent words and their distances listed. A distance of 0 indicates that the word was not present in the word embedding model for measurement.

```
{
  "ran": [
    [
      0.7525337934494,
      [
        [
          "spg1p",
          1,
          0.76627397537231
        ],
        [
          "bub3p",
          1,
          0.76032739877701
        ],
        [
          "rho1",
          1,
          0.76021540164948
        ],
        [
          "cdc20p",
          1,
          0.75895142555237
        ],
        [
          "rhofamily",
          1,
          0.75202655792236
        ],
        [
          "rab",
          1,
          0.74850535392761
        ],
        [
          "mad2p",
          1,
          0.74652522802353
        ],
        [
          "sec4p",
          1,
          0.74634456634521
        ],
        [
          "crkii",
          1,
          0.74336504936218
    ],
    [
      "gtpaseactivating",
      1,
      0.7428030371666
    ]
  ]
],
[
  0.74853223562241,
  [
    [
      "work",
      3,
      0.75750041007996
    ],
    [
      "directed",
      1,
      0.75334674119949
    ],
    [
      "jp",
      1,
      0.75021094083786
    ],
    [
      "constructed",
      4,
      0.74959361553192
    ],
    [
      "wz",
      1,
      0.74841511249542
    ],
    [
      "cb",
      1,
      0.74797928333282
    ],
    [
      "lh",
      2,
      0.74624490737915
    ],
    [
      "undertook",
      1,
      0.74481874704361
    ],
    [
      "ab",
      4,
          0.74420654773712
        ],
        [
          "prepared",
          2,
          0.74300688505173
        ]
      ]
    ]
  ]
```

Appendix O: Nearest Neighbours Cosine Similarity Julia Code.

```
#Import needed modules
import JSON

#Reset Globals
global inCount = 0

#Read File
file = “/Users/ambig_words.txt"

#Globals
inputData = []

#DICTIONARY
#inputData = Dict([("A", 1), ("B", 2)])
#println(inputData["A"])
#inputData["LOL"] = 1212;

#Parse each line of a file
open(file,"r") do f
    for line in eachline(f)
        global inCount = inCount + 1

        #For every ambigious word:
        if length(line) > 1
            new = replace(line, "\n", "")
            temp = split(new, ":")
            word = temp[1]
            contexts_raw = temp[2]
            contexts = split(contexts_raw, ",")

            push!(inputData,[word,[]])

            for content in contexts
                clusterList = split(content, " ")
                push!(inputData[inCount][2],clusterList)
            end
        end
    end
end
```

```
output = Dict();

for data in inputData
    output[data[1]] = []
    counter = 0

    ambig_word = data[1]

    max_count_cluster = 0
    nu_max_count_cluster = 0

    try
        ambig_word_senses = expected_pi(vm, dict.word2id[ambig_word])
        println(ambig_word)

        #Try to guess which is the bio sense
        co = 0

        max_count = 0
        max_count_cluster = 0

        for i in ambig_word_senses
            co = co + 1
            if i > 0.001
                cluster = nearest_neighbors(vm, dict, ambig_word, co,
10)
                numz = 0

                for wrd in cluster
                    interest = wrd[1]
                    numz = numz + length(matchall(r"0", interest))
                    numz = numz + length(matchall(r"1", interest))
                    numz = numz + length(matchall(r"2", interest))
                    numz = numz + length(matchall(r"3", interest))
                    numz = numz + length(matchall(r"4", interest))
                    numz = numz + length(matchall(r"5", interest))
                    numz = numz + length(matchall(r"6", interest))
                    numz = numz + length(matchall(r"7", interest))
                    numz = numz + length(matchall(r"8", interest))
                    numz = numz + length(matchall(r"9", interest))

                end

                if numz > max_count
                    max_count = numz
                    max_count_cluster = co
                end
                #println(cluster[1][1])
            end
        end

        max_count = 0
        nu_max_count_cluster = 0
        co = 0

        for i in ambig_word_senses
            co = co + 1
            if i > 0.001
                cluster = nearest_neighbors(vm, dict, ambig_word, co,
10)
                numz = 0

                numz = cluster[1][3]

                if numz > max_count
                    if co != max_count_cluster
                        max_count = numz
                        nu_max_count_cluster = co
                    end
                end
                #println(cluster[1][1])
            end
        end
    end

    println(max_count_cluster)
    println(nu_max_count_cluster)


    #For each word calculat NN
    bio_cluster = max_count_cluster
    eng_strongest_cluster = nu_max_count_cluster

    try
        #Grab senses
        senses = expected_pi(vm, dict.word2id[ambig_word])


        counter = 0
        for sense in senses

            #Track which sense we are on
            counter = counter + 1

            if counter == bio_cluster

                total = 0
                total_num = 0
                cluster = nearest_neighbors(vm, dict, ambig_word,
counter, 10)
                for word in cluster
                    total_num = total_num + 1
                    total = total + word[3]
                end
                average = total/total_num
                push!(output[data[1]], [average, cluster])

            end


            if counter == eng_strongest_cluster

                total = 0
                total_num = 0
                cluster = nearest_neighbors(vm, dict, ambig_word,
counter, 10)
                for word in cluster
                    total_num = total_num + 1
                    total = total + word[3]
                end
                average = total/total_num

                push!(output[data[1]], [average, cluster])

            end

        end
    end





end

file = "/Users/out.json"
open(file, "w") do f
    write(f, JSON.json(output))
end
```