Abstract

We replicate the analysis found in Landauer and Dumais (1997) where they use Grolier encyclopedia articles to train a program to answer closest synonym questions using their Latent Semantic Analysis (LSA) algorithm. Our analysis instead uses four-gram data generated from an Internet corpus. In short, we validate LSA’s dimension reduction optimality property but a full comparison was not possible due to computational constraints.

1 Introduction

Plato noted 2400 years ago that people seemingly have more knowledge than appears to be present in the information they are exposed to. His solution was that we are born with the knowledge innately and we just need a hint or intimation to “unlock” it from within.

A canonical problem in learning theory is how we acquire vocabulary: how exactly do we learn the thousands of words we know? According to recent research, the estimated rate of acquisition is astonishing — nearly 7-15 words per day for children. More shockingly, Chomsky (1991) and others showed that adult language is insufficient to learn both grammar and core competency vocabulary.

There must be some mechanism inside the human mind that can make efficient use the incoming information, which is paltry, that explains our observed language learning.

2 Latent Semantic Analysis

2.1 Background and Theory

To answer Plato’s problem, Landauer and Dumais (1997) posit a less numinous theory of learning called “Latent Semantic Analysis” (LSA). The learning is “latent” because it lies unobserved and mostly unconscious; it is “semantic” because it is based on learning words, and there is an “analysis” because an internal computation is made.

The idea is simple. People encounter knowledge in the form of written contexts, conversations, movies, etc. Each of these experiences we’ll denote as “contexts”. After experiencing many contexts, over time, the brain begins to build knowledge intuitively. The idea of
LSA is that contexts are interrelated and knowledge is built by the brain exploiting these interrelations.

Furthermore, there exists a mechanism that shrinks the dimensionality of the space of all knowledge. Obviously two articles about computer hardware may be shrunk into one “computer hardware” space. However, you can imagine a section of a psychology textbook concerning artificial intelligence also getting sucked into the “computer hardware” space due to some degree of interrelation. The brain’s ability to shrink the space of experiences (dimension optimization) can greatly amplify learning ability. If “artificial intelligence” and “computer hardware” were treated as independent facts, they would never be associated.

It is worth noting that this idea is quite iconoclastic. There exists a great body of literature in psychology, linguistics, etc. that assume that human knowledge rests upon special foundational knowledge (knowing grammar structures for instance) rather than a simple, general principle.

How do we simulate knowledge-learning from experiencing contexts? To adduce evidence for the validity of their theory, the following experiment was run.

### 2.2 The LSA Experiment and Results

Landauer and Dumais (1997) used 30,473 articles from Grolier (1980), a popular contemporary academic encyclopedia in the United States. To simulate “contexts”, they took the first 2,000 characters (or the whole text, whichever was shorter) which averaged to be approximately 150 words, about the size of a large paragraph. They then removed all punctuation and converted the words to lowercase.

To simulate “knowledge” induction, they had to define what “knowledge” meant exactly in this example. They chose the canonical problem of learning vocabulary. The number of words that appeared more than once in the encyclopedia of which there were 60,768, became these granules of knowledge. The test of their method would be an assessment of LSA’s ability to learn words.

For each context, the word totals were tabulated (words that appeared in only one context were discarded). Therefore for each context, we can generate a vector \( \mathbf{x}_j \in \mathbb{N}^{60,768} \). For ease of analysis, the counts in each entry were transformed via:

\[
\begin{align*}
x'_{ij} &= \frac{\ln (1 + x_{ij})}{\text{entropy}(x_i)} = \frac{\ln (1 + x_{ij})}{\sum_{l=1}^{p} \mathbb{P}(x_{il}) \log_2 (\mathbb{P}(x_{il}))} = \frac{\ln (1 + x_{ij})}{\sum_{l=1}^{p} \frac{x_{il}}{\sum_{k=1}^{n} x_{ik}} \log_2 \left( \frac{x_{il}}{\sum_{k=1}^{n} x_{ik}} \right)} \tag{1}
\end{align*}
\]

To represent all the data, we can concatenate each column together and we define the data matrix as \( X = [\mathbf{x}_1 \cdots \mathbf{x}_p] \) where the \( n = 60,768 \) rows represent unique words and the \( p = 30,473 \) columns are the contexts.

To find the salient dimensions of the data, the authors used singular value decomposition (SVD). See appendix A.1 for a review of the mathematics. Now, they “shrink” the space of experience following the method of least squares approximation (see appendix A.2) which entails picking the first \( d \ll p \) most salient dimensions. Denote the approximated matrix \( \hat{X}^{(d)} \).
Landauer and Dumais (1997) then gauged the theory’s ability to actually learn words. They used 80 synonym questions from a retired TOEFL examination. The format is “problem word” and four “alternative choices”. In the example below “zenith” is the problem word and the correct answer is (b) “pinnacle”.

5. zenith
   a. completion
   b. pinnacle
   c. outset
   d. decline

How would the \( d \)-dimensional model evaluate the first alternative? It would somehow have to compute the similarity between the row vector for “zenith”, denoted \( \hat{x}_{\text{T}}^{(d)} \) for “target”, and the row vector for “completion”, denoted \( \hat{x}_a \) because it is the first choice.

There are many ways to compare vectors but they want to ensure that there is no effects due to the raw frequencies of the words (i.e. their magnitude). This is akin to representing the topic rather than how much is said about the topic. In information retrieval applications (as we have seen in class), the cosine measure works well, which was their choice.

The best guess \( (g^*) \) to a synonym question would be to take the smallest angle between the vectors i.e. the largest cosine which can be denoted formally:

\[
g^* = \arg \min_{g \in \{a,b,c,d\}} \left\{ \theta_{\hat{x}_{\text{T}}^{(d)}, \hat{x}_g^{(d)}} \right\} = \arg \max_{g \in \{a,b,c,d\}} \left\{ \cos \left( \theta_{\hat{x}_{\text{T}}^{(d)}, \hat{x}_g^{(d)}} \right) \right\} = \arg \max_{g \in \{a,b,c,d\}} \left\{ \frac{\langle \hat{x}_{\text{T}}^{(d)}, \hat{x}_g^{(d)} \rangle}{\| \hat{x}_{\text{T}}^{(d)} \| \| \hat{x}_g^{(d)} \|} \right\} \tag{2}
\]

How well did their algorithm do? See figure 1.

![Figure 1: Performance on 80-synonym test as a function of \( d \), the number of dimensions used to approximate \( X \), log scale (excerpted from page 220 in Landauer and Dumais (1997))](image)

There are two extraordinary results (a) after controlling for chance, their model achieved a stellar 51.5% correct after correcting for chance guessing (see appendix A.3) which would
garner the computer running the LSA algorithm admission to a United States university (b) the model features a local max at approximately $d = 300$ dimensions.

The second result has implications in learning theory. The authors have demonstrated that as the space of contexts is squished, the performance on the synonym test increases until a certain threshold and then begins to decrease. This suggests that too many dimensions can “confuse” the algorithm, disallowing fruitful interrelationships from gelling. However, too few dimensions are too “squished” and knowledge begins to blend together. What is amazing is the global max at approximately 300 dimensions which allowed for triple the performance compared with the full model with $p$ dimensions.

2.3 Other LSA Analyses and Thoughts

To further validate that LSA models human knowledge induction, the authors measured how quickly the algorithm could “learn words”. In short, they found that the model’s improvement, per paragraph of new encyclopedia contexts, is comparable to estimates to the rate that children acquire vocabulary.

Almost as astounding as the model’s overall success is how parsimonious its structure is. The model treats the contextual examples as mere unordered bags-of-words without structure or punctuation. The model treats each glyph as a separate word (obviously related words such as “color” and “colors” would be considered different). The model cannot take advantage of morphology, syntax, similarity in spelling, or any of the myriad other properties humans are privy to.

The authors conclude that LSA partially answers Plato’s ancient question of knowledge induction. Beyond vocabulary acquisition, it may be extendable to all forms of knowledge representation.\footnote{This study was cited about 2,500 times since its publication in 1997. A further literature search was not done to see if the model was improved upon. However, the paper does appear to be very popular in the machine learning literature.}

3 LSA Duplication Using Google 4-Grams

In short, my study attempts to repeat the LSA experiment found in section 2.2 replacing contexts from the truncated Grolier encyclopedia entries with 4-grams from the Google web corpus (Goo, 2006).

3.1 Experimental Setup

We proceed with minor alterations from the procedures found in Landauer and Dumais (1997). The code can be found in appendix B.

We first load the first 100,000 unigrams and the 80 TOEFL questions. We refer to “TOEFL” words for both the target and the alternatives. Immediately, 12 questions were disqualified since they were not even present in the unigram listing.
Also, due to computational restrictions, we could not load all of the 4-grams (not even close). Therefore, we picked a certain maximum number \( p = 2000 \) which was also chosen to allow expedient computation.

However, since the TOEFL words are rare, we were unable to grab the first \( p \) 4-grams naively, we had to choose the \( p \) wisely. A happy medium was to ensure that 10% of the 4-grams contained at least one TOEFL word. This still posed problems. Some TOEFL words are much more common than others so the 10% would fill up with words like “large” and “make”. Therefore, the number of 4-grams allowed in per TOEFL word was capped at 5.

Each 4-gram formed a column. The rows were then the taken as the union of words in all \( p \) 4-grams (\( n = 2173 \)). The column entries were either zero if the word did not appear in 4-gram, or the count of how often the 4-gram appeared in the Google web corpus. We then employ the same preprocessing as found in equation 1 to arrive at the data matrix \( X \).

We then calculate the SVD and do a dimension reduction to obtain \( \hat{X}(d) \), the least squares estimate. Then, to assess the effectiveness of the model, we assess using the same TOEFL questions from the original LSA study.

Once again, not all the TOEFL words were present. Therefore, we skipped questions that were missing the target word (the question prompt), and questions that were missing all the alternative choices. However, for expediency, we left in questions where a subset of the alternative choices and the target word was represented in \( X \). Unfortunately, this left us only with 39 questions of the original 80 and not all of them complete with all four answer choices.

We compute our best guesses for the TOEFL synonym test by using the cosine measure found in equation 2. We repeat this procedure for all the dimensions \( d \in \{2, 3, \ldots, \text{MAX}\} \). The MAX dimension was chosen to be 200 for computational concerns. For each chosen dimension, we calculate the adjusted score using the correction for chance guessing found in appendix A.3. The entire computation took about 45min on a 3GHz machine. The adjusted score versus the number of dimensions is shown in figure 2.

### 3.2 Conclusions and Comparison to Previous Study

Since our implementation suffered from computational difficulties, we were unable to make use all the synonym questions during assessment, so a direct comparison with the results from Landauer and Dumais (1997) was not possible.

The crowning achievement of this crude duplication, is that we see a very clear effect of optimum dimensionality. In figure 2, we observe a local maximum at around 50-70 dimensions. This agrees with the result of the original LSA study where a dimension-reduction by a large factor yields optimal results. Note the precipitous drop in performance after \( d = 110 \). Most likely, if the knowledge space isn’t sufficiently condensed and the interrelations are not exploited, four word contexts offer very, very little information about the meaning of words.

\[ \text{Note that rank}[X] = 983 < \min\{n, p\} \text{ since most of the 2173 words were singletons and therefore ignored in the computation by zeroing out the row completely.} \]

\[ \text{Note that it is possible the correct answer did not appear, but this would not bias any given dimension’s performance, which was the goal of this duplication effort.} \]
I still do not have any intuition on if many 4-grams could compete with encyclopedia articles. The more that are included, the better representated the knowledge space becomes. Since we were unable to increase $p$ without computational issues, this was not able to be investigated.

### 3.3 Future Directions

This study was limited due to computational constraints. It would be nice to use five-grams and increase the number of vectors $p$ and be able to assess using all the synonym questions. This would require a more efficient SVD algorithm and upgraded hardware.

Using the 4-grams from the Internet is a convenient way to test a variety of very short contexts. A problem I find to be of interest is optimizing over both length of context and dimensionality of context. I propose a study where we we duplicate Landauer and Dumais (1997) and vary their 2000 character word limit on the encyclopaedia articles. Instead of using Grolier (1980) encyclopedia, we can use the Internet. This would require a more updated Google corpus with n-grams beyond $n = 5$ or, even better, a raw text dump.

### Acknowledgments

Thanks to Professor Dean Foster for mentorship and giving me this interesting project. Thanks to Bill Oliver from Professor Thomas Landauer’s lab in the University of Colorado at Boulder for providing the TOEFL questions from the original study.
A Technical Appendix

A.1 Singular Value Decomposition

Any real matrix $X \in \mathbb{R}^{n \times m}$ with rank $[X] = p \leq m$ can be decomposed into three matrices in a process known as singular value decomposition (SVD):

$$X = F_1 W F_2^T$$

where $F_1 \in \mathbb{R}^{n \times p}$, $W \in \mathbb{R}^{p \times p}$, and $F_2^T \in \mathbb{R}^{p \times m}$

The matrices $F_1, F_2$ are traditionally orthonormal can be thought of as “dimensions” or “factors” of the matrix $X$, which can be thought of as principal components, capturing the internal structure of the data. The matrix $W$ is a positive diagonal matrix where the non-zero entries in a particular column $W_{jj}$ are called “singular values” and represent the “weight” of the corresponding column vector in factor matrices $F_1, F_2$.

Why are there two factor matrices and why are they not related? In short, because we have a rectangular matrix. If $X$ was square, we would have a standard eigendecomposition of $X = FWF^{-1}$ where all the dimensionality is really captured in one matrix (since the second factor matrix is a computable function of the first).

As a side note, the two factor matrices in SVD actually correspond to eigenvectors for two special spaces of $X$. $F_1$ is actually the eigenvectors for the matrix $XX^T$ also called the “output” basis vectors for $X$ and $F_2$ is actually the eigenvectors for the matrix $X^TX$ which is also called the “input” basis vectors for $X$. The $W$ matrix’s entries are actually the square root of the eigenvalues of both $XX^T$ and $X^TX$ (which explains why all diagonal entries are positive).

It is also traditional to “order” the singular values in $W$, putting the largest in the $W_{11}$ position, the second-largest in the $W_{22}$ position, and so forth. The corresponding eigenvectors in $F_1, F_2$ are then analogously ordered making the corresponding switches. This is really convenient because upon inspection it allows the investigator to see the vectors in order of importance and how important they are relative to each other (by examining the weights) without the need to dart all over the matrix.
A.2 Matrix Approximation via SVD

We can approximate the original matrix $X$ by using a subset of the factors found in the previous section. This would be akin to using a subspace of a lower dimension to represent a higher dimensional object. Since this approximation is really a projection of a higher space onto a lower space, i.e. the same mechanism used in “least squares” regression, we call this approximation $\hat{X}$, the least squares estimate of $X$.

How is this done in practice? A subset of the factors of the original space are selected. A convenient computational mechanism is to vanish the weights on the factors you no longer desire. For instance, let’s say we want to estimate $X$ by just the most important vector, we can set $W_{22} = W_{33} = \ldots = 0$ and compute.

$$X = F_1 \begin{bmatrix} W_{11} & 0 & 0 \\ 0 & W_{22} & 0 \\ \vdots & & \ddots \end{bmatrix} F_2^T$$

$$\hat{X}^{(1)} = F_1 \begin{bmatrix} W_{11} & 0 & 0 \\ 0 & 0 & 0 \\ \vdots & & \ddots \end{bmatrix} F_2^T$$

The least squares estimate is denoted $\hat{X}^{(1)}$ where the “1” denotes that we are approximating it using only one dimension. This estimate can be accurate if most of the variation in $X$ is captured in that one hyperline; if the variation is more evenly distributed among other dimensions, the approximation will be very poor.

A.3 Chance Guessing on Tests

Consider a test with 100 questions and 5 answer choices. The taker knew 50 answers but on the other 50, he guessed. In expectation, he will get 10 of the 50 guesses correct for a total score of 60. In order to adjust for guessing we could subtract off $\frac{1}{4}$ of the incorrect (algebra not shown).

Generally speaking, a test with $T$ questions and $Q$ answer choices and a raw score of $p$ percent will have an adjusted score $p'$ given by:

$$p' = \left( \frac{\# \text{correct} - \# \text{incorrect} \times \text{adj}}{\# \text{total}} \right)_+ = \left( \frac{pT - (1-p)TQ^{-1}}{T} \right)_+ = \left( \frac{pQ - 1}{Q - 1} \right)_+$$

We need the positive component in the above formula since if the taker guessed all the questions at random, he deserves a zero (not a negative score, which is impossible).

B Ruby Code

We now provide the code used to run the experiment found in section 3. Ruby 1.8.6 (http://www.ruby-lang.org/en/) was used as well as the ruby linear algebra library which is a wrapper for a native fortran implementation (http://linalg.rubyforge.org/).
require 'linalg'
include Linalg
require 't_o_e_f_l'
include TOEFL

MAX_FIVEGRAMS = 2000
MAX_TOEFL_INDEX_COUNT = 5
MAX_DIMENSIONS = 200

MAX_UNIGRAMS = 10

# We first should load the unigrams
# we want to have the unigrams in order that they appear since
# this is how they are referenced in the n-grams file
unigrams_ordered = []

# we also would like to have a hash of the words so we can have a
dictionary to check if a mystery word actually exists, we might
# as well store the index as a value since we get it for free
unigrams_indices = {}

f = File.open("google_top_100000_unigrams", "r")
# read each line, split along whitespace in middle and take the left part
l = 0
f.each_line do |line|
  unless l.zero?  # we don’t care about the header
    unigram = line.match(/".*(\n)\"/).captures.first
    # print unigram
    # print \"\n"
    unigrams_ordered << unigram
    unigrams_indices[unigram] = l
    # break if l >= MAX_UNIGRAMS
  end
  l += 1
end
f.close

p "total unigrams: #{unigrams_ordered.length}"
# let's get all the words from the synonym test

toefl_words = []
SynonymQuestions.each_with_index do |q, i|
  next if QuestionsToBeKilled.include?(i + 1)
toefl_words << q[:target] << q[:alternatives] << q[:correct_answer]
end
toefl_words = toefl_words.flatten uniq # kill duplicates
p "num toefl questions: #{NumSynonymQuestions}"
p "num toefl words #{toefl_words.length}"
toefl_words_not_found_in_unigrams = []
toefl_words.each do |toefl_word|
toefl_words_not_found_in_unigrams << toefl_word if unigrams_indices[toefl_word].nil?
end
toefl_words_not_found_in_unigrams = toefl_words_not_found_in_unigrams.flatten uniq # kill duplicates
p "num toefl words not in unigrams: #{
  toefl_words_not_found_in_unigrams.length}: #{
  toefl_words_not_found_in_unigrams.join(', ')}"

# now get the indices of the toefl words inside the unigrams
toefl_unigram_indices = toefl_words.map{|word| unigrams_indices[word]}
p "toefl indices:"
toefl_unigram_indices.each_with_index do |index, i|
  print "#{toefl_words[i]} --- #{index} "
end
toefl_unigram_indices_count = {}

### N-Grams

five_grams = []
unique_words_in_five_grams = {} # global tabulation of unique words found in the five gram

# now we should load some 5-grams and keep track of the unique words
f = File.open("C:\Users\kapelner\Desktop\google_data\google_10000_4grms", "r")
k = 0
f.each_line do |line|
k += 1
  five_gram_raw = line.split(/\s/).map{|unigram_ind| unigram_ind.to_i}
  # ensure integers
#the first five positions are the indices of the unigrams, so that's the "real" 5-gram
five_gram = five_gram_raw [0 .. 5]

# now if the gram does not contain any of the words in the toefl test, ditch it

# p "array intersection: #{(toefl_unigram_indices & five_gram)}"
intersection = toefl_unigram_indices & five_gram
skip_this_context = intersection.empty?

intersection.each do |toefl_index|
  toefl_unigram_indices_count[toefl_index] = (  
    toefl_unigram_indices_count[toefl_index].nil? ? 1 :  
    toefl_unigram_indices_count[toefl_index] + 1)
  skip_this_context = true if toefl_unigram_indices_count[
    toefl_index] > MAX_TOEFL_INDEX_COUNT

end
skip_this_context = false if rand < 0.001
next if skip_this_context

# now we need to keep track of the words (and their counts) used in all five-grams
# by tabulating the words in this particular five gram
five_gram.each{|unigram_ind| unique_words_in_five_grams[unigram_ind]
  = unique_words_in_five_grams[unigram_ind].nil? ? 1 :  
  unique_words_in_five_grams[unigram_ind] + 1}

# n = five_grams[1].last.to_i # the last is the frequency

# p n + " times: " + five_gram.map{|ind| unigrams_ordered[ind.to_i]}.join(' , ')

# finally add it to the array
five_grams <<= five_gram_raw

# p "five grams: #{five_grams.length} k = #{k}" if five_grams.length % 50 == 0
break if five_grams.length >= MAX_FIVEGRAMS
end
f.close

# kill the words that only appear once
unique_words_in_five_grams.each{|unigram_ind, freq|
  unique_words_in_five_grams.remove(unigram_ind) if freq.zero?}

# now we need to go back to see how many toefl words were left out of the five grams
toefl_words_not_found_in_ngrams = {}
toefl_unigram_indices.each do |toefl_unigram_index|
  toefl_words_not_found_in_ngrams[toefl_unigram_index] = true if
    unique_words_in_five_grams[toefl_unigram_index].nil?
end
Now we have to go through an nuke the questions that we cannot attempt
SynonymQuestions.each do |q|
  toefl_words = ([] << q[:target] << q[:alternatives] << q[:correct_answer]).flatten
  keep_question = true
  toefl_words.each do |word|
    index = unigrams_indices[word]
    if index.nil? # or toefl_words_not_found_in_ngrams[index]
      keep_question = false
      break
    end
  end
  questions_we_can_answer << q if keep_question
end
p "num toefl synonym questions we can answer: #{questions_we_can_answer.length}"

exit if questions_we_can_answer.length.zero?

#now we need the unique words in an ordering
unique_words_in_five_grams_ordering = {}
unique_words_in_five_grams.keys.each_with_index{|unigram_ind, i| unique_words_in_five_grams_ordering[unigram_ind] = i}
p "num unique words in five grams: #{unique_words_in_five_grams.length}"

# Matrix creation and SVD
#now create X, the data matrix
#the number of rows should be n = # of unique words
n = unique_words_in_five_grams.length
#the number of cols should be the number of five grams
p = five_grams.length

#now create X using the linalg package
162 xmat = DMMatrix.new(n, p)
163 # now we have to set up the X matrix, and we have to do it manually
164 # use convention that i iterates over rows, and j iterates over columns
165 five_grams.each_with_index do |fivegram, j|
166   # p "five gram to be inserted: #{fivegram.join(', ')} (#{fivegram.join(',')})"
167   freq = fivegram.last # pull out frequency
168   fivegram[0...5].each do |unigram|
169     i = unique_words_in_five_grams_ordering[unigram]
170     # p "insert i=#{i} j=#{j} of dims #{n} x #{p} #{fivegram.join(', ')}" #
171     xmat[i, j] = freq
172   end
173 end
174 p "matrix created with dims #{n} x #{p}, now preprocessing"
175 # p xmat
176
177 # now we have to preprocess the X
178 0.upto(xmat.num_rows - 1) do |i|
179   # first find sum of the row (number of words throughout all n-grams)
180   sum = 0.0
181   0.upto(xmat.num_columns - 1) do |j|
182     sum += xmat[i, j]
183   end
184   # now find the probabilities for each n-gram
185   probs = Array.new(xmat.num_columns)
186   0.upto(xmat.num_columns - 1) do |j|
187     probs[j] = xmat[i, j] / sum
188   end
189   # now calculate entropy (remember if prob = 0, entropy defined to be 0, consistent with limit)
190   entropy = 0.0
191   0.upto(xmat.num_columns - 1) do |j|
192     entropy += probs[j] * Math.log(probs[j]) unless probs[j].zero?
193   end
194   # now assign a new value (only if its non-zero to begin with)
195   0.upto(xmat.num_columns - 1) do |j|
196     unless xmat[i, j].zero?
197       xmat[i, j] = entropy.zero? ? 0 : (1 + xmat[i, j]) / entropy
198     end
199   end
200 end
201
202 # p xmat
203
204 # keep first d singular values
205 def reduce_dimensions(d, x)
0.upto([x.num_rows - 1, x.num_columns - 1].min) do |i|
  x[i, i] = 0 if i >= d
end
x
end

# SVD analysis
p "done preprocessing, now running matrix SVD"
f_1_mat, wmat, f_2_mat = xmat.singular_value_decomposition
p "done matrix SVD now beginning dimensional tests"
p "dims: f_1_mat #{f_1_mat.dimensions.join('x ')} w_mat #{wmat.dimensions.join('x ')} f_2_mat #{f_2_mat.dimensions.join('x ')}
  rank: #{wmat.rank}"

def calculate_cosine(x, i1, i2)
  row1 = x.row(i1)
  row2 = x.row(i2)
  cosine = ((row1 * row2.t) / (row1.norm * row2.norm))[0, 0] # a scalar, but we still need to return the float
  (cosine.infinite? or cosine.nan?) ? 10000 : cosine
end

results = Array.new([wmat.rank, MAX_DIMENSIONS].min + 1, 0)
([wmat.rank, MAX_DIMENSIONS].min).downto(2) do |d|

  wmat_d = reduce_dimensions(d, wmat)
xmathat_d = f_1_mat * wmat_d * f_2_mat

  # now calculate some answers
  questions_answered = 0
  questions_correct = 0
  questions_we_can_answer.each do |q|
    i_target = unique_words_in_five_grams_ordering[unigrams_indices[q[:target]]]
    next if i_target.nil?
    cosines = {}
    q[:alternatives].each do |alt|
      i_alt = unique_words_in_five_grams_ordering[unigrams_indices[alt]]
      next if i_alt.nil?
      cosines[alt] = calculate_cosine(xmathat_d, i_target, i_alt)
    end
    next if cosines.empty?
end
```ruby
# p cosines

guess = cosines.invert[cosines.values.max]
questions_answered += 1
questions_correct += 1 if guess == q[:correct_answer]

# p "guess #{guess}" for question with target word "#{q[:target]}"

end

results[d] = questions_correct / questions_answered.to_f
p "results for dimension #{d} -- #{{questions_correct}} / #{questions_answered} = #{'%.2f' % (results[d] * 100)}%"

end

# now dump results... note that they are not yet adjusted for guessing
File.open('results.txt', 'w') do |f|
  results.each{|res| f.puts(res)}
end

In addition, below is an example of what the module which contains the TOEFL synonym listings looks like:

```

SynonymQuestions = [
  # 1. enormously
  # a. appropriately
  # b. uniquely
  # c. tremendously*
  # d. decidedly
  {
    :target => 'enormously',
    :alternatives => %w(appropriately uniquely tremendously decidedly ),
    :correct_answer => 'tremendously'
  },
  # 2. provisions
  # a. stipulations*
  # b. interrelations
  # c. jurisdictions
  # d. interpretations
  {
    :target => 'provisions',
    :alternatives => %w(stipulations interrelations jurisdictions interpretations),
    :correct_answer => 'stipulations'
  }
]
```