Latent Semantic Analysis Using Google N-Grams

Statistics 991 Computational Linguistics Final Project

Adam Kapelner

December 9, 2010

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, lingruistics, 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 $\boldsymbol{x}_{\cdot j} \in \mathbb{N}^{\,60{,}768}$. For ease of analysis, the counts in each entry were transformed via:

$$x'_{ij} = \frac{\ln\left(1 + x_{ij}\right)}{\text{entropy}(\boldsymbol{x}_{i\cdot})} = \frac{\ln\left(1 + x_{ij}\right)}{\sum_{l=1}^{p} \mathbb{P}\left(x_{il}\right) \log_2\left(\mathbb{P}\left(x_{il}\right)\right)} = \frac{\ln\left(1 + x_{ij}\right)}{\sum_{l=1}^{p} \frac{x_{il}}{\sum_{k=1}^{p} x_{ik}} \log_2\left(\frac{x_{il}}{\sum_{k=1}^{p} x_{ik}}\right)} \tag{1}$$

To represent all the data, we can concatenate each column together and we define the data matrix as $X = \left[\boldsymbol{x}'_{\cdot 1} \cdots \boldsymbol{x}'_{\cdot p}\right]$ 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 $\hat{\boldsymbol{x}}_{\text{T.}}^{(d)}$ for "target", and the row vector for "completion", denoted $\hat{\boldsymbol{x}}_{\text{a.}}$ 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:

$$g^* = \underset{g\in\{a,b,c,d\}}{\arg\min} \left\{ \theta_{\hat{\boldsymbol{x}}_{\text{T.}}^{(d)},\ \hat{\boldsymbol{x}}_{\text{g.}}^{(d)}} \right\} = \underset{g\in\{a,b,c,d\}}{\arg\max} \left\{ \cos\left(\theta_{\hat{\boldsymbol{x}}_{\text{T.}}^{(d)},\ \hat{\boldsymbol{x}}_{\text{g.}}^{(d)}} \right) \right\} = \underset{g\in\{a,b,c,d\}}{\arg\max} \left\{ \frac{\left\langle \hat{\boldsymbol{x}}_{\text{T.}}^{(d)},\ \hat{\boldsymbol{x}}_{\text{g.}}^{(d)} \right\rangle}{\left|\left| \hat{\boldsymbol{x}}_{\text{T.}}^{(d)} \right|\right| \left|\left| \hat{\boldsymbol{x}}_{\text{g.}}^{(d)} \right|\right|} \right\} \quad (2)$$

How well did their algorithm do? See figure 1.

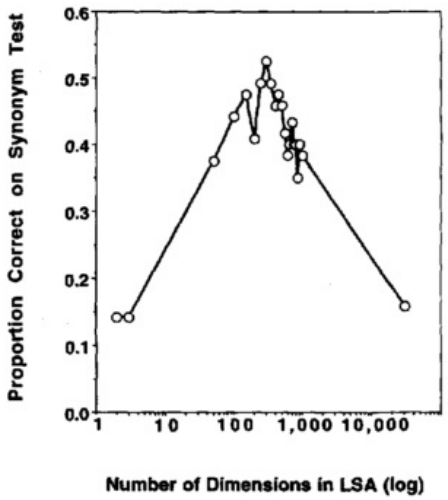

Figure 1: Performance on 80-synonym test as a function of d, the number of dimensions used to approximate X, log scale (excerpted from page 220 in Landauer and Dumais (1997))

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.[1]

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.

[1]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.

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.[2]

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.[3] 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.

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

[3]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.

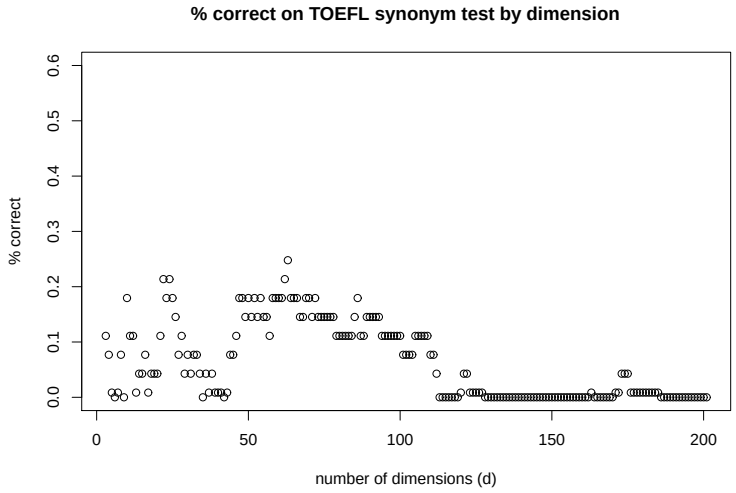

Figure 2: Our results

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 encylclopedia 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.

References

N. Chomsky. Linguistics and Cognitive Science: Problems and Mysteries." In A. Kasher (ed.), The Chomskyan Turn. Cambridge, MA: Basil Blackwell. 1991.

Web 1T 5-gram Corpus Version 1.1, 2006. Google, Inc.

Grolier. *Grolier's Academic American Encyclopedia*. Arte Publishing, 1980.

TK Landauer and ST Dumais. A solution to Platos problem: The latent semantic analysis theory of acquisition, induction and representation of knowledge. *Psychological review*, 1 (2):211–240, 1997.

A Technical Appendix

A.1 Singular Value Decomposition

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

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

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 exmaining 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 mechanims 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.

$$\begin{aligned} X &= F_1 \begin{bmatrix} W_{11} & 0 & 0 \\ 0 & W_{22} & 0 \\ & & \ddots \end{bmatrix} F_2^T \\ \hat{X}^{(1)} &= F_1 \begin{bmatrix} W_{11} & 0 & 0 \\ 0 & 0 & 0 \\ & & \ddots \end{bmatrix} F_2^T \end{aligned}$$

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:

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

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/).

lsa_duplication_experiment.rb

```ruby
require 'linalg'
include Linalg
require 't_o_e_f_l'
include TOEFL

MAX_FIVEGRAMS = 2000
MAX_TOEFL_INDEX_COUNT = 5
MAX_DIMENSIONS = 200

######################
### Unigrams
#
#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(/"(.*)",/).captures.first
  #  print unigram
  #  print "\r\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}"


######################
### TOEFL-synonym-test-related
#

```

```
47 #let's get all the words from the synonym test
48 toefl_words = []
49 SynonymQuestions.each_with_index do |q, i|
50   next if QuestionsToBeKilled.include?(i + 1)
51   toefl_words << q[:target] << q[:alternatives] << q[:correct_answer]
52 end
53 toefl_words = toefl_words.flatten.uniq #kill duplicates
54 p "num toefl questions: #{NumSynonymQuestions}"
55 p "num toefl words #{toefl_words.length}"
56
57 toefl_words_not_found_in_unigrams = []
58 toefl_words.each do |toefl_word|
59   toefl_words_not_found_in_unigrams << toefl_word if unigrams_indices[
       toefl_word].nil?
60 end
61 p "num toefl words not in unigrams: #{
     toefl_words_not_found_in_unigrams.length}: #{
     toefl_words_not_found_in_unigrams.join(', ')}"
62
63 #now get the indices of the toefl words inside the unigrams
64 toefl_unigram_indices = toefl_words.map{|word| unigrams_indices[word]}
65 #p "toefl indices:"
66 #toefl_unigram_indices.each_with_index do |index, i|
67 #  print "#{toefl_words[i]} -- #{index}   "
68 #end
69 toefl_unigram_indices_count = {}
70
71 ######################
72 ### N-Grams
73 #
74
75 five_grams = []
76 unique_words_in_five_grams = {} #global tabulature of unique words found in
       the five gram
77
78 #now we should load some 5-grams and keep track of the unique words
79 f = File.open("C:\\Users\\kapelner\\Desktop\\google_data\\
     google_10000_4grms", "r")
80 k = 0
81 f.each_line do |line|
82   k += 1
83 #  p line
84   #keep the entire five gram and the format is:
85   #[word 1 index, word 2 index, word 3 index, word 4 index, word 5 index,
       frequency]
86   five_gram_raw = line.split(/\s/).map{|unigram_ind| unigram_ind.to_i}
       #ensure integers
```

```
87    #the first five positions are the indices of the unigrams, so that's the "
          real" 5-gram
88    five_gram = five_gram_raw[0...5]
89    #now if the gram does not contain any of the words in the toefl test, ditch
          it
90  #  p "array intersection: #{(toefl_unigram_indices & five_gram)}"
91    intersection = toefl_unigram_indices & five_gram
92    skip_this_context = intersection.empty?
93    intersection.each do |toefl_index|
94      toefl_unigram_indices_count[toefl_index] = (
            toefl_unigram_indices_count[toefl_index].nil? ? 1 :
            toefl_unigram_indices_count[toefl_index] + 1)
95      skip_this_context = true if toefl_unigram_indices_count[
            toefl_index] > MAX_TOEFL_INDEX_COUNT
96    end
97    skip_this_context = false if rand < 0.001
98    next if skip_this_context
99
100   #now we need to keep track of the words (and their counts) used in all five-
          grams
101   #by tabulating the words in this particular five gram
102   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)}
103 #  n = five_grams[l].last.to_i #the last is the frequency
104 #  p n + " times: " + five_gram.map{|ind| unigrams_ordered[ind.to_i]}.join('
        ')
105   #finally add it to the array
106   five_grams << five_gram_raw
107 #  p "five grams: #{five_grams.length} k = #{k}" if five_grams.length % 50 ==
        0
108   break if five_grams.length >= MAX_FIVEGRAMS
109 end
110 f.close
111
112 #kill the words that only appear once
113 unique_words_in_five_grams.each{|unigram_ind, freq|
      unique_words_in_five_grams.remove(unigram_ind) if freq.zero?}
114
115 #now we need to go back to see how many toefl words were left out of the five
      grams
116 toefl_words_not_found_in_ngrams = {}
117 toefl_unigram_indices.each do |toefl_unigram_index|
118   toefl_words_not_found_in_ngrams[toefl_unigram_index] = true if
          unique_words_in_five_grams[toefl_unigram_index].nil?
119 end
```

```
120 p "num toefl words not found in n grams: #{
      toefl_words_not_found_in_ngrams.length}"
121
122 #now we have to go through an nuke the questions that we cannot attempt
123 questions_we_can_answer = []
124 SynonymQuestions.each do |q|
125   toefl_words = ([] << q[:target] << q[:alternatives] << q[:
        correct_answer]).flatten
126   keep_question = true
127   toefl_words.each do |word|
128     index = unigrams_indices[word]
129     if index.nil? #or toefl_words_not_found_in_ngrams[index]
130       keep_question = false
131       break
132     end
133   end
134   questions_we_can_answer << q if keep_question
135 end
136 p "num toefl synonym questions we can answer: #{
      questions_we_can_answer.length}"
137
138 exit if questions_we_can_answer.length.zero?
139
140 #unique_words.each do |unigram_index, freq|
141 #  p "#{unigrams_ordered[unigram_index]} --- #{freq}"
142 #end
143
144 #now we need the unique words in an ordering
145 unique_words_in_five_grams_ordering = {}
146 unique_words_in_five_grams.keys.each_with_index{|unigram_ind, i|
      unique_words_in_five_grams_ordering[unigram_ind] = i}
147
148 p "num unique words in five grams: #{unique_words_in_five_grams.
      length}"
149
150 ########################
151 ### Matrix creation and SVD
152 #
153
154 #now create X, the data matrix
155 #the number of rows should be n = # of unique words
156 n = unique_words_in_five_grams.length
157 #the number of cols should be the number of five grams
158 p = five_grams.length
159
160
161 #now create X using the linalg package
```

```
xmat = DMatrix.new(n, p)
#now we have to set up the X matrix, and we have to do it manually
#use convention that i iterates over rows, and j iterates over columns
five_grams.each_with_index do |five_gram_raw, j|
#  p "five gram to be inserted: #{five_gram_raw.join(', ')} (#{five_gram_raw.map{|ind| unigrams_ordered[ind.to_i]}.join(' ')})"
  freq = five_gram_raw.last #pull out frequency
  five_gram_raw[0...5].each do |unigram_ind|
    i = unique_words_in_five_grams_ordering[unigram_ind]
#    p "insert i=#{i} j=#{j} of dims #{n} x #{p}   #{five_gram_raw.join(', ')} (#{five_gram_raw.map{|ind| unigrams_ordered[ind.to_i]}.join(' ')})"
    xmat[i, j] = freq
  end
end
p "matrix created with dims #{n} x #{p}, now preprocessing"
#p xmat

#now we have to preprocess the X
0.upto(xmat.num_rows - 1) do |i|
  #first find sum of the row (number of words throughout all n-grams)
  sum = 0.to_f
  0.upto(xmat.num_columns - 1) do |j|
    sum += xmat[i, j]
  end
  #now find the probabilities for each n-gram
  probs = Array.new(xmat.num_columns)
  0.upto(xmat.num_columns - 1) do |j|
    probs[j] = xmat[i, j] / sum
  end
  #now calculate entropy (remember if prob = 0, entropy defined to be 0, consistent with limit)
  entropy = 0.to_f
  0.upto(xmat.num_columns - 1) do |j|
    entropy += probs[j] * Math.log(probs[j]) unless probs[j].zero?
  end
  #now assign a new value (only if its non-zero to begin with)
  0.upto(xmat.num_columns - 1) do |j|
    unless xmat[i, j].zero?
      xmat[i, j] = entropy.zero? ? 0 : (1 + xmat[i, j]) / entropy
    end
  end
end

#p xmat

#keep first d singular values
def reduce_dimensions(d, x)
```

```
206   0.upto([x.num_rows - 1, x.num_columns - 1].min) do |i|
207     x[i, i] = 0 if i >= d
208   end
209   x
210 end
211
212 #svd analysis
213 p "done preprocessing, now running matrix SVD"
214 f_1_mat, wmat, f_2_mat = xmat.singular_value_decomposition
215 p "done matrix SVD now beginning dimensional tests"
216 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}"
217
218 #p w_mat
219 #now check questions
220
221 def calculate_cosine(x, i1, i2)
222   row1 = x.row(i1)
223   row2 = x.row(i2)
224   cosine = ((row1 * row2.t) / (row1.norm * row2.norm))[0, 0] #a scalar,
       but we still need to return the float
225   (cosine.infinite? or cosine.nan?) ? 10000 : cosine ####HACK
226 end
227
228 results = Array.new([wmat.rank, MAX_DIMENSIONS].min + 1, 0)
229 ([wmat.rank, MAX_DIMENSIONS].min).downto(2) do |d| #we go down so we don'
    t have to waste time copying the matrix w_mat
230
231   wmat_d = reduce_dimensions(d, wmat)
232   xmathat_d = f_1_mat * wmat_d * f_2_mat
233
234   #now calculate some answers
235   questions_answered = 0
236   questions_correct = 0
237   questions_we_can_answer.each do |q|
238     i_target = unique_words_in_five_grams_ordering[unigrams_indices[q[:
        target]]]
239     next if i_target.nil?
240     cosines = {}
241     q[:alternatives].each do |alt|
242       i_alt = unique_words_in_five_grams_ordering[unigrams_indices[alt
          ]]
243       next if i_alt.nil?
244       cosines[alt] = calculate_cosine(xmathat_d, i_target, i_alt)
245     end
246     next if cosines.empty?
```

```
247 #    p cosines
248     guess = cosines.invert[cosines.values.max]
249     questions_answered += 1
250     questions_correct +=1 if guess == q[:correct_answer]
251 #    p "guess '#{guess}' for question with target word '#{q[:target]}'"
252
253   end
254   results[d] = questions_correct / questions_answered.to_f
255   p "results for dimension #{d} -- #{questions_correct} / #{
        questions_answered} = #{'%.2f' % (results[d] * 100)}%"
256 end
257
258 #now dump results... note that they are not yet adjusted for guessing
259 File.open('results.txt', 'w') do |f|
260   results.each{|res| f.puts(res)}
261 end
```

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

```
1   SynonymQuestions = [
2 #    1. enormously
3 #    a. appropriately
4 #    b. uniquely
5 #    c. tremendously*
6 #    d. decidedly
7     {
8       :target => 'enormously',
9       :alternatives => %w(appropriately uniquely tremendously decidedly
            ),
10      :correct_answer => 'tremendously'
11    },
12 #    2. provisions
13 #    a. stipulations*
14 #    b. interrelations
15 #    c. jurisdictions
16 #    d. interpretations
17    {
18      :target => 'provisions',
19      :alternatives => %w(stipulations interrelations jurisdictions
          interpretations),
20      :correct_answer => 'stipulations'
21    },
22 .
23 .
24 .
```