Using Latent Semantic Analysis for Extractive Summarization

Kirill Kireyev
University of Colorado Boulder
Computer Science
`kireyev@colorado.edu`

Abstract

In this paper, we use simple techniques derived from on Latent Semantic Analysis (LSA) to provide a simple and robust way of generating extractive summaries for TAC 2008 Update Summarization task.

1 Introduction

TAC 2008 Update Summarization task is to write a short (~ 100-word) summary of a set of newswire articles, under the assumption that the user has already read a given set of earlier articles.

The summaries are to be generated over a number of batches of independent news articles, each-batch related to a particular topic. The summaries are intended to be in response to a specific query, which is available in both short and narrative form. For example:
[short form]:
`Arctic and Antarctic ice melt`
[narrative form form]:
`"Describe the developments and impact of the continuing Arctic and Antarctic ice melts."`

There are a total of 48 such query/document batches. Each document batch is farther partitioned into two sets. Both sets are to be summarized independently using information related to the query corresponding to the document batch. The second of the two sets is to be summarized under the assumption that the user has read the first set, i.e. the second summary should contain information that is new in relation to the first set.

2 CU System Overview

Our system is a simple extractive summarization system, meaning that it simply extracts representative sentences from the source documents, without attempting to modify them.

The core of the system is on Latent Semantic Analysis (LSA). We give a brief overview of LSA in the next section, the reader is also advised to consult [1] for more information.

2.1 Latent Semantic Analysis (LSA)

Latent Semantic Analysis ([1]) is an unsupervised methods of deriving vector space semantic representation from a large corpus of texts. LSA starts by representing a collection of documents by a term by document (*T x D*) matrix *A*, which in essence represents each word by a D-dimensional vector. It then performs singular value decomposition (SVD) on the matrix:

$$A = U \Sigma V^T \qquad (1)$$

Subsequently, all but the first (largest) *k* values in the diagonal singular matrix Σ, are set to zero, resulting in a kind of principal component analysis. This effectively reduces the dimensionality of each word vector to *k*. (For more details, please consult [2]). The number of dimensions (*k*) is determined empirically. The dimensions have no intuitive interpretation; they simply serve to position word vectors in the high-dimensional space.

The measure of semantic similarity between two words in this model is typically[1] the cosine of

[1]Other metrics like Eucledian distance and dot product are less commonly used

the angle between their corresponding word vectors :

$$S(w1,w2)=\cos(v_{w1},v_{w2})=\frac{v_{w1}\cdot v_{w2}}{\|v_{w1}\|\|v_{w2}\|} \quad (2)$$

The simulated meaning of a new document (sometimes referred to as pseudo-document) can be represented in LSA using the following method:

$$v_d=q^T U_k \Sigma_k^{-1} \quad (3)$$

where q represents the array containing type frequencies for words in the document (weighted by *tf-idf*-derived entropy weights). Note that this is equivalent to (weighted) geometric addition of constituent word vectors corresponding to words in a document.. As a result, both words and documents are represented as vectors in k-dimensional space[2], allowing for straightforward word-word, word-document, and document-document comparisons., which reflect their semantic similarity according to the model:

$$S(w,d)=\cos(v_w\Sigma^{1/2},v_d\Sigma^{1/2}) \quad (4)$$

$$S(d_{1,}d_2)=\cos(v_{d1}\Sigma,v_{d2}\Sigma) \quad (5)$$

The absolute values of cosines (which may range between -1 and 1 with larger values indicating greater similarity) have no strict interpretation; only comparisons of cosine values (e.g. between *pairs* of words) are meaningful.

2.2 Implementation Details

We built a new LSA space using a corpus consisting of news articles (from sources New York Times) using the documents in the ACQUAINT-2 corpus. The corpus consisted of 439,947 documents. The resulting LSA space used 327 dimensions.

We compute the document vectors for each sentence in the source document as well as for the queries in the short form.

We experimented with all combinations of

(1) sentences vs paragraphs as units of extraction

(2) short vs narrative form of queries

using DUC 2007 data for the same task, and obtained slightly better results with sentence-length units and short-form queries.

[2]Depending on the type of comparison, operands need to be multiplied by the singular matrix Σ (word-word) or its square root (word-doc). Please see LSA literature for more details.

3 Extracting Sentences with LSA

In selecting sentences for the summary, our system attempts to fulfill the following desiderata:

1. Sentences should be related to the query
2. Sentences should cover different topics
3. Sentences should contain as much information as possible
4. (for update summaries) sentences should be cover different topics from sentences already selected for the summary of the first set

We now discuss in more detail how our system accomplishes each of these criteria.

3.1 Sentences Related to Query

To measure how much a sentence is related to the query, we compute the cosine between the query vector and the sentence vector, and select a number (10) of sentences with the highest cosine values.

3.2 Maximizing Topic Coverage

To find a set of sentences that cover a wide range of topics, we use a clustering algorithm to partition the set of candidate sentences into a number of clusters. The clusters will ideally represent distinct semantic themes. We use k-means clustering on the LSA vectors of the candidate sentences. We use the cosine measure as the distance metric for the k-means algorithm. The number of clusters (k) is chosen to be the number of sentences that we ultimately wish to return. The k-means algorithm returns the assignment of each sentence to a particular cluster, as well as the centroid of each cluster, in the units of the LSA coordinate space used.

We then select a sentence that is most representative of each cluster, by finding sentences which are closest to each cluster's centroid, using the cosine metric.

3.3 Maximizing Amount of Information per Sentence

Due to 100-word size restriction of summaries, we wish to generate summaries that are as information-dense as possible. Consequently, we wish to find sentences that contain as much information as per word as possible.

We can approximate this quantity using a word concreteness measure derived from LSA. One characteristic of LSA word vectors is their vector length, which differs significantly among different words. Roughly speaking, it is a function of two factors:

(1) Number of occurrences in training corpus

(2) Concreteness, or specificity, or words

For example, Kintsch ([2]), writes:

> "Intuitively, the vector length tells us how much information LSA has about this vector. [...] Words that LSA knows a lot about (because they appear frequently in the training corpus, in many different contexts) have greater vector lengths than words LSA does not know well. Function words that are used frequently in many different contexts have low vector lengths -- LSA knows nothing about them and cannot tell them apart since they appear in all contexts."

To illustrate, below are examples of different words and their corresponding vector lengths:

	High Frequency		Low Frequency	
High Specificity	dog father box	1.31 1.01 0.68	proton sheriff triangle	0.43 0.14 0.14
Low Specificity	the how since	0.01 0.39 0.27	haphaz-ard clumsy	0.03 0.06

Therefore, we can isolate the effects of these two factors and compute word specificity in the following way:

specificity = { vector length } / { frequency }

As a result, we can obtain an approximation of word specificity. The amount of information in a sentence can the be approximated by computing the mean of specificity values of each constituent word (we use the sum of logarithms to dampen the large fluctuations in specificity values).

Below is an example of a high-specificity and a low-specificity sentence, and their corresponding specificity scores.

Specificity score	**Sentence**
8.9	We found there was too much mass, Schoepf said. We had to work pretty hard to get back to the specifications we'd committed ourselves to with our clients.
69.3	By using chromate-free paint, engineers got the outer paintwork down to about 350 kilograms (770 pounds), Schoepf said. That's compared to 550 kilograms (1,210 pounds) for a plane of this size using other paints.

3.4 Finding Non-Redundant Sentences

To ensure that selected sentences for the second (update) set are sufficiently non-redundant with respect to the sentences selected for the first set, we compute pairwise cosine distances between all sentences selected for the first set and all candidate sentences for the second set. The system selects the sentences with the smallest such cosines.

3.5 Putting it Together

The following is the pseudo-code for the overall algorithm:

```
foreach (b in batches):
Qb = query(b)

Sb1 = get_sentences(batch=b, set=1)

# compute score for each sentence, based on
# 1) similarity to query
# 2) specificity
foreach (s in Sb1):
  sim_s = cos (vector(Qb), vector(Sb))
  spec_s = avg (log (specificity (w)), w ∈ Sb1
  score_s = spec_s * sim_s

# pick 10 best sentences with highest scores
Sb1' = max(score_s, 10)

# cluster them using k-means into 5 clusters
cluster_centers = kmeans(vector(Sb1'), 5)

# pick the representative sentence for
# each cluster center
foreach (cc in cluster_centers):
  add (Rb1, max(cos(cc, vector(Sb1'), 5))

# Report sentences for batch b, set 1
print Rb1

Sb2 = get_sentences(batch=b, set=2)

# compute score for each sentence, based on
# 1) similarity to query
# 2) specificity
# 3) dis-similarity to sentences in set 1
foreach (s in Sb2):
  sim_s = cos (vector(Qb), vector(Sb))
  spec_s = avg (log (specificity (w)), w ∈ Sb1
  d_s = 1-max(cos(vector(Sb2), vector(sb1))
  score_s = spec_s * sim_s * d_s

# pick 10 best sentences
Sb2' = max(score_s, 10)

# cluster them using k-means
cluster_centers = kmeans(vector(Sb2'), 5)

# pick the representative sentence for
# each cluster center
foreach (cc in cluster_centers):
  add (Rb2, max(cos(cc, vector(Sb1'), 5))

# Report sentences for batch b, set 2
print Rb2
```

4 Results

At TAC 2008, the system ranked 37th out of 58 for the average modified pyramid score, 20th out 58 for linguistic quality, and 35th out of 58 for overall responsiveness. It should be noted that these results were obtained with fairly simple, but robust techniques; no sophisticated syntactic/semantic analysis or natural language generation was involved. We believe that our methods can be used in conjunction with these techniques to obtain much better results.

References

[1] Kintsch, W., Predication. *Journal of Cognitive Science,* 25 (2001).

[2] Landauer, T.K., McNamara, D.S., Dennis, S., Kintsch, W. (2007). *Handbook of Latent Semantic Analysis* Lawrence Erlbaum