Vector Properties of Good Summaries

Adly Templeton
Williams College
at7@williams.edu

Jugal Kalita
University of Colorado, Colorado Springs
jkalita@uccs.edu

***Abstract*—Vector semantics is used in many areas of Natural Language Processing. We explore the application of vector semantics to the problem of automatic summarization. We demonstrate several properties of vector semantics useful for this purpose. In particular, we show that cosine similarity between sentence vectors and document vectors is strongly correlated with sentence importance and that vector semantics can identify and correct gaps between the sentences chosen so far and the document. In addition, we identify specific dimensions which are linked to effective summaries. To our knowledge, this is the first time specific dimensions of sentence embeddings have been connected to sentence properties. We also compare the features of different methods of sentence embeddings. Many of these insights have applications in uses of sentence embeddings far beyond summarization.**

***Index Terms*—Vector Semantics, Automatic Summarization, Extractive Summarization, Sentence Embeddings**

I. Introduction

The large volume of news articles published every day motivates effective forms of automatic summarization. The most basic and most well-studied form of automatic summarization is sentence extraction, in which entire unmodified sentences are selected from the original document. These selected sentences are concatenated to form a short summary, which ideally contains the most important information from the original document while avoiding redundancy.

While a variety of successful models have been proposed for sentence extraction, these models often rely on various metrics, such as term frequency, which are based on the occurrence of individual words. For instance, a state-of-the-art graph-based summarization model by Parveen et al. uses the words shared between sentences for all three of their major metrics (importance, redundancy, and coherence) [1]. However metrics based on the occurrence of specific words, and not the meaning of these words, may perform sub-optimally in certain situations (such as dealing with synonyms).

Meanwhile, *vector semantics* have been growing in popularity for many other natural language processing applications. Vector semantics attempt to represent words as vectors in a high-dimensional space, where vectors which are close to each other have similar meanings. Various models of vector semantics have been proposed, such as LSA [2], word2vec [3], and GLOVE [4], and these models have proved to be successful in other natural language processing applications. While these models work well for individual words, producing equivalent vectors for sentences or documents has proven to be more difficult.

This material is based upon work supported by the National Science Foundation under Grant No. 1659788 and 1359275

II. Problem Description

For our purposes, extractive summarization is essentially reducible to *sentence selection*. That is, we want to select a subset of sentences from the set of all sentences in the original document, which maximizes the quality of the summary while remaining under some word limit. Note that this problem is more complex than text classification: The quality of each individual sentence depends of the other sentences in the summary. In particular, a good summary should contain minimal redundancy. This trade off between sentence salience and redundancy is the basis for many summarization algorithms, including some in our work.

Any practical summarization system would likely include other steps after sentence selection, such as *sentence reordering* or *text-simplification*. However, effective algorithms for these tasks already exist, and we are primarily concerned with sentence selection in this paper.

We are primarily concerned with multi-document summarization algorithms, which summarize a *cluster* of related documents. However, all our algorithms disregard the document-by-document information and treat the document cluster.

III. Related Work

Particular attention should be given to the method of combining word embeddings into sentence embeddings, a very difficult problem. However, news summarization may not require some of the nuances in meaning to be represented, as we are primarily concerned with the topic, not the meaning, of a sentence. In particular, news articles will rarely contain sentences expressing contradictory views on the same subject. For instance, our algorithm will rarely need to differentiate between the sentence embeddings for sentences such as “John loves Mary” and “Mary loves John”, which have divergent meanings but the same words. This is in sharp contrast to typical testing cases for sentence embeddings, such as the detection of paraphrased sentences. Then, summarization gives us an opportunity to compare the effectiveness of different sentence embeddings in practice.

In recent years, a number of techniques for sentence embeddings have emerged. One promising method is *paragraph vectors* (Also known as Doc2Vec), described by Le and Mikolov [5]. The model behind paragraph vectors resembles that behind word2vec, except that a classifier uses an additional ‘paragraph vector’ to predict words in a *Skip-Gram* model. When trained, the paragraph vectors capture the meanings of their paragraphs.

Another model, *skip-thoughts*, attempts to extend the word2vec model in a different way [6]. The center of the skip-thought model is an encoder-decoder neural network which, given a sentence, attempts to predict the surrounding sentences. The result, *skip-thought vectors*, achieve good performance on a wide variety of natural language tasks.

Li et al. used a LSTM-based auto encoder, where the encoding in the middle of the autoencoder is taken as a vector representation of a sentence. In addition, a hierarchical version of the autoencoder can also be used to form vectors for whole paragraphs or documents. [7]

Despite the complexity of these neural network approaches, simpler approaches based on linear combinations of the word vectors have managed to achieve state-of-the-art results for non-domain-specific tasks [8]. Arora et al. [9] offer one particularly promising such approach. First, this model finds the weighted average of the word vectors (less frequent words weighted higher). This weighting is achieved through the smooth inverse function $w = \frac{a}{a+f}$, where w is the weight of a word's vector, f is the estimated frequency of that word, and a is a hyperparameter ($a \approx 0.0001$). Note that this weighting is roughly analogous to weighting by tfidf. The second step is to subtract the projection along a specific vector, c_0. c_0 is the "common discourse vector", correlated with words such as "the" or "and" which appear consistently in all English contexts. c_0 is found by taking the first principal component of a representative sample of text vectors. This simple method was found to achieve equal or greater performance in some tasks than more complicated supervised learning methods.

A. Extractive Summarization

State-of-the-art extractive summarization techniques have been achieved with a wide variety of methods. Here, we provide a short overview of some recent successful techniques techniques.

Cao et al. [10] used a recursive neural network, which operates on a parsing tree of a sentence, to rank sentences for summarization.

Cheng and Lapata [11] successfully used a neural-network-based sentence extractor. This extractor was a recurrent neural network with a neural attention mechanism. The network considered the document encodings and the previously selected sentences, as well as the current sentence in making its decisions.

Parveen et al. [1] used a graph-based approach, modeling a document as "a bipartite graph consisting of sentence and entity nodes", and used various rankings on the graph to select sentences according to certain criteria (importance, redundancy, and coherence).

Ren et al. [12] achieved state-of-the-art results through a regression-based approach. A variety of engineered features are used as inputs into a regression model, which estimates the redundancy-aware importance of a sentence given the sentences already selected. The single highest rated sentence is selected, and the process is repeated.

B. Previous Embedding-based Approaches

To our knowledge, no one has explored the use of modern sentence embedding methods, such as Paragraph Vectors or Skip-Thought vectors, in summarization. However, some work has been done on summarization using word2vec representations.

Gong and Liu [13] presented a version of text summarization based on vector semantics. However, instead of using the embeddings of a word as determined by a larger corpus, they attempted to calculate the embedding of a word based off of analysis only on the document in question. In addition, due to the age of their work, they used LSA instead of techniques such as word2vec.

Kageback et al. [14] used cosine similarity between the sum of word2vec embeddings, as well as a recursive auto-encoder, to modify another standard summarization algorithm (sub-modular optimization)

Ren et al. [12] used "Average Word Embedding" as one of many independent variables in a regression model, but it is unclear how much that particular variable contributed to the success of the model.

Cheng and Lapata [11] used word embeddings as the input to a neural network as part of summarization, but they did not directly compare embeddings. They used a single-layer convolutional neural network to encode sentences, and a LSTM neural network to encode documents.

Nayeem and Chali [15] recently achieved state-of-the-art results using a modified version of LexRank, using a combination of cosine similarity of weighted averages of vectors and named entity overlap.

IV. Methods

To explore potential sentence embeddings, we implement the four sentence embeddings above as *vector functions*, which convert sentences or documents to vectors.

A. Vector Functions

- *SIF Average*: The most basic sentence embedding is simply the weighted average of word vectors from Arora et al. [9], without the common component removal. We use the Brown corpus [16] for word frequency information.
- *Arora*: This method is simply the method described in Arora et al. It is equivalent to the one above, except with common component removal added. We use the Brown corpus both to compute the common component vector, and for word frequency information.
- *Paragraph Vectors*: The paragraph vector approach described above. We used the 300-dimensional DBOW model pretrained by Lau et al. [17] on the wikipedia corpus.
- *Skip-Thought Vectors* The skip-thought vector approach described above. We used the 4800-dimensional combined-skip model [6], [18].

All sentence embeddings are normalized to produce unit vectors

B. Potential Selector Functions

There are many potential ways to use vector semantics. To explore the design space, we consider combinations of *vector functions* and *selector functions*, functions which, given vector representations for sentences, extracts a summary. We present a large variety of example selector functions in order to allow us to explore interaction effects between selector functions and vector functions.

- *Near*: The most basic selector function, *Near*, selects the sentences whose sentence vectors have the highest cosine similarity with the document vector
- *Near Nonredundant*: An attempt at balancing redundancy with salience, Near Nonredundant down-weights the cosine similarity scores by their average cosine similarity the sentences selected so far. Because this redundancy measure is strongly (quadratically) correlated with cosine similarity to the document, we fit a regression for redundancy for each vector function, and use the residual on this regression for the final algorithm.
- *LexRank*: Our next selector is based off of the classical LexRank algorithm [19]. The center of this algorithm is a weighted graph where nodes represent sentences, and weights are the similarities between sentences, as determined by cosine similarity between tfidf vectors. The PageRank algorithm is then used on this graph to identify the most salient sentences for extraction. We use a modified version of this algorithm, where the weights of edges are determined by the cosine similarity between sentence embeddings.
- *Cluster*: The basis of this selector is a simple clustering algorithm in the vector space of sentence embeddings. We use an Agglomerative Clustering algorithm (using cosine similarity as its distance metric) to find clusters in the set of sentence embeddings. We then find the sentence closest to the average of each cluster and add it to the summary. To ensure we find summaries which meet the word-length requirement, we increase the number of clusters we search for until we have selected sentences totaling 100 words.
- *Greedy*: The greedy selector, at each step, selects the sentence such that the cosine similarity of the new summary (including previously selected sentences) is maximized. This is subtly different than the Near selector for average-based vector functions, but significantly different for Paragraph Vectors.
- *Brute Force*: Another attempt at optimizing the cosine similarity between the summary and the document, this selector creates a pool of the 20 sentences with the highest cosine similarity. From this pool, every combination of sentences (with an appropriate word count) is tried, and the combination with the highest cosine similarity is selected as the summary.
- *Max Similarity*: A proof-of-concept selector which computes results for both the Greedy and Brute Force selectors and then selects the result with the highest cosine similarity to the document vector.
- *Near-then-Redundancy*: Similar to the Brute Force selector, this selector creates the same pool of sentences described above, except with a size of 15. From this pool, this algorithm optimizes via brute force to minimize redundancy (defined as the average cosine similarity between pairs of sentences). Note that the size of the sentence pool, which is essentially a computational shortcut in the Brute Force selector, is now a performance-critical hyper-parameter.
- *PCA*: This selector performs Principal Component Analysis (PCA) on the set of sentence vectors in a document. Then, the algorithm selects the one sentence closest to the first component, one sentence closest to the second component, and so on, until the length capacity is met.
- Random: This selector simply selects sentences at random, until the word limit is reached. This provides a lower-bound on the performance of an effective algorithm, and is used for baseline comparisons

V. Performance of Selector Functions

A. Experimental Evaluation

Because evaluation of summarization is fundamentally a subjective task, human evaluations are ideal. However, human evaluations are often expensive and time-consuming to obtain. Luckily, some metrics of automatically evaluating summaries, by comparison to a human-written summary, have been developed. Traditionally, various forms of the ROUGE metric, which compare shared n-grams, have been used. [20]. ROUGE has been shown to correlate strongly with human judgments [21], and is our primary metric for evaluating summaries[1] We report ROUGE-1 and ROUGE-2 statistics, which correspond to unigrams and bigrams, respectively.

We split the document clusters in the DUC 2004 dataset into a testing set and a validation set of approximately equal sizes. The pre-defined training set of the DUC 2001 dataset was used as a training set for some of the graphs and data analysis presented here.

B. Results

We present results for Multi-Document Summarization on the DUC 2004 dataset (Table I). A few notes on the results:

- Our results fall far below the state of the art, likely due to the simplicity of our selector functions as compared to the state-of-the-art methods. In addition, almost all of our methods are unsupervised, unlike most of the state-of-the-art methods.
- The best performing selector, Greedy, is both very simple and based on fundamental principles of vector semantics.
- Paragraph Vectors work much worse with the Clustering and Greedy algorithms, and work much better with Near and SVMs.
- Many combinations of selector function and vector function do not work above the level of random chance.
- In general, despite their sophistication, Paragraph Vectors and Skip-Thought vectors perform worse than much more basic approaches.

[1]For direct comparison with Hong et al., we trucate summaries to 100 words and use the following parameters, for direct comparison with Hong et al. [22]: -n 4 -m -a -l 100 -x -c 95 -r 1000 -f A -p 0.5 -t 0.

	SIF Average	Arora	Paragraph Vectors	Skipthought
LexRank	32.6 (6.8)	32.6 (6.8)	32.6 (6.8)	**32.6 (6.8)**
Near Nonredundant	33.6 (6.1)	**34.5** (6.3)	32.6 (5.5)	32.1 (4.9)
Brute Force	32.0 (5.7)	32.2 (6.3)	**33.0 (6.6)**	31.4 (4.5)
Near-then-Redundancy	33.2 (6.2)	34.2 **(6.9)**	31.5 (5.4)	33.1(5.3)
PCA	32.9 (5.6)	33.5 (5.6)	32.0 (5.5)	NA
Max Similarity	32.0 (5.7)	32.2 (6.3)	**33.0 (6.6)**	NA
Greedy	**35.1 (7.0)**	33.1 (6.0)	NA	NA
Near	32.5 (5.4)	32.2 (5.5)	33.1 (6.1)	NA
Cluster	NA	NA	NA	32.1 (4.6)
Random	30.1 (4.1)			
State-of-the-art (Ren et al.) [12]	40.4 (11.7)			

TABLE I: ROUGE-1 Results on the DUC 2004 dataset. ROUGE-2 results in parentheses. All combinations which do not perform significantly better than random chance ($p < .05$, using a paired t-test) are replaced with 'NA' for clarity. SIF Average with either Max Similarity or Brute Force were included, despite having p=.051. In addition, one combination (Max Similarity with Skipthought Vectors) are not computed, but are not expected to perform better than chance. Selector Functions are roughly organized according to the vector functions with which they are effective. For Skipthought vectors, *docvec-avg* is used (Section VI-E)

	SIF Average	Arora	Paragraph Vectors	Skipthought
Near Nonredundant	-1.98	-1.72	-0.734	+3.81
Brute Force	-0.323	-0.256	-1.24	+3.42
Near-then-Redundancy	+0.739	-0.584	-0.205	+3.46
Max Similarity	-0.323	-0.256	-1.24	NA
Greedy	+0.254	-0.813	-3.11	NA
Near	-0.868	-0.0652	-5.53	+1.76
Total Average	-.417	-.614	-2.01	+2.74

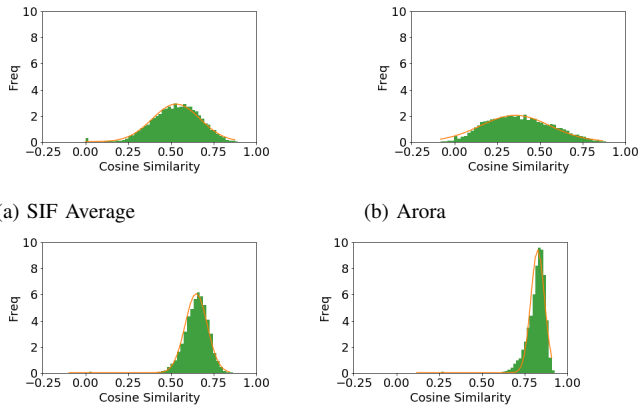
TABLE II: A comparison of document vector methods. Numbers represent the difference in ROUGE-1 scores between document vector methods. Positive numbers represent a gain when using *docvec-avg*. Selectors which do not use the document vector have been omitted.

VI. Discussion

Despite the poor performance of our models compared to the baselines, analyses of the underlying data provide many useful insights into the behavior of vector semantics in real-world tasks.

A. Distributions of Cosine Scores

The cosine scores between all sentence vectors and the corresponding document vectors follow a normal distribution for all vector functions (Fig. 1), but this effect is most pronounced for paragraph vectors ($r^2 = .996$). In addition, the sentence embeddings for paragraph vectors and skip-thought vectors are far closer to the document embedding than would be expected from a random distribution, with mean cosine similarities of .65 and .84, respectively (Unsurprisingly, this also holds for Average and Arora, though the similarity is notably lower (.52 for Average, .35 for Arora).



(a) SIF Average (b) Arora

(c) Paragraph Vectors (d) Skipthought

Fig. 1: Distribution of cosine similarity scores between each sentence vector and their corresponding document vector, for all four vector functions.

B. Correlation of Cosine Scores with Good Summaries

By identifying the sentences present in an optimal summarization, we show that optimal sentences have higher cosine scores, and that this effect is increased after adjusting cosine scores for word length (Fig. 2). However, there is a lot of overlap, implying that, although this method has some power to discern good summaries from bad summaries, the power of this method alone is not high enough to product good summaries.

C. Regression on Vector Dimensions

We calculated the isolated ROUGE score of each individual sentence in the training set, and the sentence embeddings for these sentences on all four vector functions. To partially eliminate the effects of the sentence's context, we subtract the corresponding document vector from all sentence vectors before regression.

Due to the large number of predictor variables, we use a Bonferroni correction, considering values significant only if they have p-values of $\frac{\alpha}{n}$, which, for $\alpha = .05$, corresponds

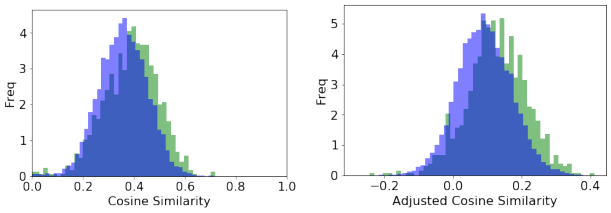
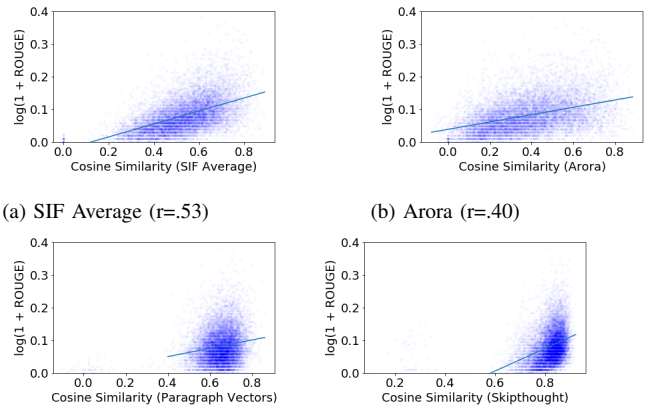


Fig. 2: Cosine Similarity to the Document Vector for non-optimal (blue) and optimal (green) sentences. Figure on the right shows Cosine Similarity adjusted for sentence word count.



(a) SIF Average (r=.53)

(b) Arora (r=.40)

(c) Paragraph Vectors (r=.17)

(d) Skipthought (r=.33)

Fig. 3: Correlation of ROUGE scores and Cosine Similarity scores per sentence. ROUGE scores transformed with $r' = log(1 + r)$, to account for zero values. Some high-leverage points have been excluded for Paragraph Vectors and Skipthought.

approximately to $p < .00001$ for the skip-thought vectors, and $p < .00016$ for all other vectors.

Three dimensions are significant at this level for *SIF Average* vectors. No dimensions are significant at this level for *Arora* vectors, though the three values significant for *SIF Average* achieve p values of .0021, .0485 and .0006. 29 dimensions are significant for *Paragraph Vectors*. 5 dimensions are significant, at the much higher threshold, for *Skip-Thought Vectors*. It appears that these specific dimensions correspond to aspects of a sentence that make it somehow more suited for a summary. Despite the theoretical implications of this result, the regression models do not have enough predictive power to create good summaries by themselves.

D. The Performance of the Greedy Algorithm

The Greedy algorithm is the most successful algorithm we present here. As its name implies, the Greedy algorithm appears to be simply an attempt at maximizing the following objective function:

$$f_{cos}(summary) = vector(summary) \cdot vector(document) \tag{1}$$

Of course, this objective function can only be an approximation to the informally-defined criteria for good summaries.

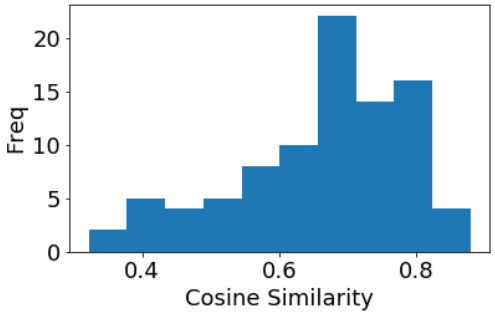


Fig. 4: A histogram of the cosine similarities of sentences selected by the Greedy algorithm.

Even so, Table I suggests that the performance of the greedy algorithm is not based on the accuracy of the corresponding objective function. In particular, consider the two other strategies which try to maximize the same objective function: Brute force, and Maximum Similarity (which simply selects Greedy or Brute Force based on which one creates a summary with a higher cosine similarity). Brute Force consistently and significantly creates summaries with higher cosine similarity to the document, outperforming the Greedy selector on its objective function. By construction, the Max Similarity algorithm outperforms in cosine similarity to an even greater degree. But both of these algorithms perform much worse than the Greedy algorithm.

Deeper analysis into the decisions of the Greedy algorithm reveals some reasons for this discrepancy. It appears that the good performance of the Greedy algorithm results not from the associated objective function, but by the way in which it maximizes this objective function. In particular, the Greedy algorithm selects sentences with low cosine similarity scores in a vacuum, but which increase the cosine similarity of the overall sentence (Fig. 4).

To understand why this is true, it we consider the step-by-step behavior of the Greedy algorithm. The first choice of the greedy algorithm is simple: it chooses the sentence with maximum cosine similarity to the document vector:

$$\bar{s}_1 = \underset{\bar{s} \in S}{\operatorname{argmax}}\, \bar{s} \cdot \bar{d}$$

(Recall that all vectors have unit-length, so cosine similarity is equivalent to the dot product).

To select the second vector, the greedy algorithm is maximizing the following equation:

$$\bar{s}_2 = \underset{\bar{s} \in S'}{\operatorname{argmax}} \left(\frac{\bar{s} + \bar{s}_1}{\|\bar{s} + \bar{s}_1\|}\right) \cdot \bar{d} \tag{2}$$

$$= \underset{\bar{s} \in S'}{\operatorname{argmax}} \frac{\bar{d} \cdot \bar{s}_1 + \bar{d} \cdot \bar{s}}{\sqrt{1 + \bar{s}_1 \cdot \bar{s}}} \tag{3}$$

Where S' is the set of remaining sentences [2]

[2]Note that the results reported above do not represent the Greedy algorithm averaging together the vectors, though the difference is minimal for SIF Average and Arora vectors(see Section VI-E for more information)

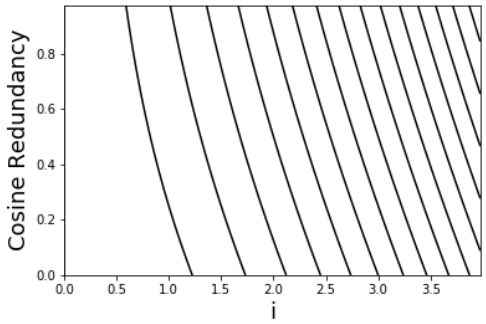


Fig. 5: A contour plot of the denominator of Equation 5

Note that this equation consists of three parts: $\bar{d} \cdot \bar{s}_1$ (a constant wrt. $\bar{s}$), $\bar{d} \cdot \bar{s}$, which is simply the salience of a sentence measured by cosine similarity, and the denominator, which is essentially a measure of redundancy. Not only does this simple metric lead to a 'natural' penalty for redundancy, it performs better than our handcrafted redundancy penalties. The way this algorithm scales when picking the i^{th} sentence is particularity noteworthy:

$$\bar{s}_{i+1} = \underset{\bar{s} \in S'}{\operatorname{argmax}} \left(\frac{\frac{1}{i+1}\bar{s} + \frac{i}{i+1}\bar{s}_p}{\left\| \frac{1}{i+1}\bar{s} + \frac{i}{i+1}\bar{s}_p \right\|} \right) \cdot \bar{d} \tag{4}$$

$$= \underset{\bar{s} \in S'}{\operatorname{argmax}} \frac{i(\bar{d} \cdot \bar{s}_p) + \bar{d} \cdot \bar{s}}{\sqrt{i^2 + 1 + 2i\bar{s}_p \cdot \bar{s}}} \tag{5}$$

$$\text{Where } \bar{s}_p = \frac{\sum_{j=0}^{i} \bar{s}_j}{\left\| \sum_{j=0}^{i} \bar{s}_j \right\|}$$

As shown in Figure 5, the behavior of this function changes as i increases. In particular, the function becomes more sensitive to redundancy, and less sensitive to salience, as the algorithm selects more sentences. In other words, the algorithm will first try to select important sentences, and then select sentences to fill in the gaps. This result, and the success of the resulting algorithm, has implications for balancing salience and redundancy in future summarization algorithms.

E. Document Vector Computation

In general, there are two ways to compute a document vector. The most obvious is to pass the entire text of the document into the vector function. This has two theoretical problems. The first is that the 'documents' in our algorithms are really clusters of documents, and are therefore non-coherent. The second is that Skip-thought vectors are not designed to handle text longer than a sentence. However, an alternative document vector, *docvec-avg*, is formed by taking the mean of the (normalized) sentence vectors. This corresponds to treating the document as a collection of sentences, instead of a collection of words. We present a comparison of the two methods in Table II

As expected, Skipthought vectors, which are not designed for text larger than a sentence, perform significantly better with the *docvec-avg* strategy. More notable is the poor performance of the *docvec-avg* strategy with Paragraph Vectors. The size of the performance gap here implies that Paragraph Vectors can combine information from multiple sentences in a manner more sophisticated than simple averaging.

More interesting is the performance for SIF Average and Arora vectors. For these vector functions, which are based on taking the average of words, *docvec-avg* very closely resembles the simple strategy. And yet there is a small but significant performance gap. The difference between the two document vectors is the weighting. *Docvec-avg*, which normalizes vectors before adding them together, removes some weighting information present in the simple strategy. In particular, the simple strategy assigns more weight to sentences with a lot of highly-weighted words. Presumably, *docvec-avg*, by ignoring this weighting, leaves out useful information. This hypothesis is supported by the greater performance gap for Arora vectors, which effectively downweights certain common words and therefore could be expected to carry more information in word weightings. Similar, but much smaller, gaps exist when computing the vectors for summaries at each step in the greedy algorithm.

F. Properties of Different Sentence Embeddings

Based on the interactions between selector functions and vector functions, as well as a variety of other pieces of data in the proceeding sections, we present a broad comparison of the properties of different sentence embedding schemes.

1) SIF Average/Arora Vectors: Three selector functions perform better with both SIF Average and Arora vectors: Near Nonredundant, Greedy, and PCA. These functions seem to be united in their comparisons between the vectors of sentence embeddings (implicitly, in the case of the greedy algorithm). These selector functions correspond to the most basic test for sentence embeddings: Judging the similarity of two sentences.

The exact difference the common component removal makes is less clear. Arora vectors hold a slight performance edge for all selectors except for Near and Greedy (the Greedy algorithm loses a full 2 points).

2) Paragraph Vectors: Two selector functions perform better with Paragraph Vectors: Near and Brute Force. Both of these are very similar: They require comparing sentence vectors to the document vector. The poor performance on algorithms such as Near Nonredundant suggests that Paragraph Vectors are especially poor at comparing sentence vectors to each other. These results suggest that Paragraph Vectors are especially good at computing document vectors, a hypothesis also implied by the results of Section VI-E.

The other distinguishing property of Paragraph Vectors is their very high correlation when regressing on the individual features.

3) Skipthought Vectors: It is hard to disentangle the properties of Skipthought vectors from the high dimensionality of the pretrained vectors we used. In general, Skipthought vectors performed poorly. They only performed better than other vector functions with one selector, Clustering, although their performance with this selector was significant.

In general, these results suggest that different sentence embedding methods are suited for different tasks, often drastically so, and the choice should be made carefully.

REFERENCES

[1] D. Parveen and M. Strube, "Integrating importance, non-redundancy and coherence in graph-based extractive summarization." in *IJCAI*, 2015, pp. 1298–1304.

[2] T. K. Landauer and S. T. Dumais, "A solution to plato's problem: The latent semantic analysis theory of acquisition, induction, and representation of knowledge." *Psychological Review*, vol. 104, no. 2, p. 211, 1997.

[3] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[4] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation." in *EMNLP*, vol. 14, 2014, pp. 1532–1543.

[5] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 1188–1196.

[6] R. Kiros, Y. Zhu, R. R. Salakhutdinov, R. Zemel, R. Urtasun, A. Torralba, and S. Fidler, "Skip-thought vectors," 2015, pp. 3294–3302.

[7] J. Li, M.-T. Luong, and D. Jurafsky, "A hierarchical neural autoencoder for paragraphs and documents," *arXiv preprint arXiv:1506.01057*, 2015.

[8] J. Wieting, M. Bansal, K. Gimpel, and K. Livescu, "Towards universal paraphrastic sentence embeddings," *arXiv preprint arXiv:1511.08198*, 2015.

[9] S. Arora, Y. Liang, and T. Ma, "A simple but tough-to-beat baseline for sentence embeddings," *ICLR*, 2016.

[10] Z. Cao, F. Wei, L. Dong, S. Li, and M. Zhou, "Ranking with recursive neural networks and its application to multi-document summarization." in *AAAI*, 2015, pp. 2153–2159.

[11] J. Cheng and M. Lapata, "Neural summarization by extracting sentences and words," *arXiv preprint arXiv:1603.07252*, 2016.

[12] P. Ren, F. Wei, Z. Chen, J. Ma, and M. Zhou, "A redundancy-aware sentence regression framework for extractive summarization."

[13] Y. Gong and X. Liu, "Generic text summarization using relevance measure and latent semantic analysis," in *Proceedings of the 24th annual international ACM SIGIR conference on Research and development in information retrieval*. ACM, 2001, pp. 19–25.

[14] M. Kågebäck, O. Mogren, N. Tahmasebi, and D. Dubhashi, "Extractive summarization using continuous vector space models," in *Proceedings of the 2nd Workshop on Continuous Vector Space Models and their Compositionality (CVSC)@ EACL*, 2014, pp. 31–39.

[15] M. T. Nayeem and Y. Chali, "Extract with order for coherent multi-document summarization," *arXiv preprint arXiv:1706.06542*, 2017.

[16] W. N. Francis and H. Kucera, "The Brown Corpus: A Standard Corpus of Present-Day Edited American English," 1979, brown University Liguistics Department.

[17] J. H. Lau and T. Baldwin, "An empirical evaluation of doc2vec with practical insights into document embedding generation," *arXiv preprint arXiv:1607.05368*, 2016.

[18] R. Al-Rfou, G. Alain, A. Almahairi, C. Angermueller, D. Bahdanau, N. Ballas, F. Bastien, J. Bayer, A. Belikov, A. Belopolsky *et al.*, "Theano: A python framework for fast computation of mathematical expressions," *arXiv preprint*, 2016.

[19] G. Erkan and D. R. Radev, "Lexrank: Graph-based lexical centrality as salience in text summarization," *Journal of Artificial Intelligence Research*, vol. 22, pp. 457–479, 2004.

[20] C.-Y. Lin, "Rouge: A package for automatic evaluation of summaries," in *Text summarization branches out: Proceedings of the ACL-04 workshop*, vol. 8. Barcelona, Spain, 2004.

[21] P. A. Rankel, J. M. Conroy, H. T. Dang, and A. Nenkova, "A decade of automatic content evaluation of news summaries: Reassessing the state of the art." in *ACL (2)*, 2013, pp. 131–136.

[22] K. Hong, J. M. Conroy, B. Favre, A. Kulesza, H. Lin, and A. Nenkova, "A repository of state of the art and competitive baseline summaries for generic news summarization." 2014, pp. 1608–1616.