# Shakespeare in One Hundred Words

CS 221/224N Project Report

Samir Bajaj
samirb@stanford.edu

## ABSTRACT

This paper describes the results of running four unsupervised generic text summarization algorithms on perhaps the most widely known and timeless works in English Literature: a collection of thirty-seven plays by William Shakespeare. While text summarization has been applied to newswire and other curated content, it is not known to the author whether any such attempt has been made to create summaries of literature that is rich in semantic and idiomatic content.

As if the task wasn't challenging enough to begin with, an aggressive target of one hundred words was set for the size of the summary extracts.

The four algorithms used in this investigation are *tf-idf*–based relevance ranking, clustering of sentences using $K$-Means, Singular Value Decomposition (SVD) of a term-sentence matrix, and identification of salient sentences by computing the PageRank of each sentence. A central idea is to identify non-redundant sentences that capture the theme of the document being summarized.

## Categories and Subject Descriptors

Artificial Intelligence [**Natural Language Processing**]: Text Summarization

## General Terms

Computational Linguistics, Information Retrieval

## Keywords

tf-idf, $K$-Means, Singular Value Decomposition, PageRank, Sentence Extraction

## 1. INTRODUCTION

According to one estimate [1], we create 2.5 quintillion bytes of data every day—so much that 90% of the data in the world today has been created in the last two years alone. This data comes from everywhere: sensors used to gather climate information, posts to social media sites, digital pictures and videos, purchase transaction records, and cell phone GPS signals to name a few. It is no surprise that efficient and accurate summarization of data is increasingly becoming an important area of research.

Text summarization has seen a number of successes in the last two decades [2], especially following the explosive growth of the Internet and easy availability of raw data. Summaries of news reports and facts-based articles can now be produced with reasonably good quality.

However, processing text that is filled with nuances and rich semantic content presents a different set of challenges, and is still an active area of research. Nevertheless, it is worth employing standard techniques from Linear Algebra and the field of Information Retrieval to explore what is possible and set a baseline for further work.

The author's hypothesis is that Relevance Ranking based on *tf-idf* weights and the PageRank-based random walk model that are both used by many successful Web Search engines, as well as Singular Value Decomposition, which has proved successful in Latent Semantic Analysis of textual data, in addition to the standard $K$-Means clustering algorithm, can all be used effectively in the task at hand.

## 2. DATA

The complete text of thirty-seven plays by Shakespeare is available at [3], annotated in XML. The longest of these is *Hamlet*, comprising of 32,152 words, and the shortest is *A Comedy of Errors*, at 16,185 words. The playwright has used 25,984 unique words across the entire set.

The text of each play was processed and transformed via a pipeline of components described below.

**Preprocessing**: Credits for the XML markup and the licensing information were removed, and the raw text was saved in UTF-8 format. One encoding error was fixed by hand. Additionally, all-uppercase names of actors preceding the lines they deliver in the play were removed. This allows the algorithms to focus on the attributes of the protagonists, without getting distracted by their names.

**Normalization**: The text was converted to lowercase, and all punctuation was removed. In order to prevent longer sentences from dominating the output of the sentence extraction process and expending the 100-word budget, the semi-colon was also treated as a sentence terminator in addition to conventional symbols like the period and the exclamation mark. As a result, summaries may contain fragments of (large) sentences from the original text.

**Stop Words**: A dictionary of 129 stop words in the English language was used, and all occurrences of these words were removed.

**Stemming**: A standard, public-domain implementation of the Porter stemmer was used on the lowercased, stop words-free text.

With the above transformations complete, all four algorithms were applied in turn to the complete text of each play and the outputs were recorded.

## 3. EVALUATION

As in the case of other machine learning problem domains like classification, it is not feasible to directly apply the precision/recall framework, or, for that matter, other similar devices to text summarization without any reference summaries to serve as labeled data. Human intervention is inevitably necessary in the evaluation of results related to research in most Natural Language Processing tasks. It is especially true in the case of evaluation of summaries of classic works that date back hundreds of years, and therefore reflect the niceties of language use and social norms in that period of human history. A good summary should distill the main theme of the document while keeping redundancy to a minimum.

A number of useful measures for evaluation of text summaries have been suggested in [5], but linguistic quality is best judged by humans. Consequently, results of the project were presented to a person who is intimately familiar with the works of William Shakespeare, and therefore has the right background[1] to evaluate the extracts. In addition to assigning grades to the outputs of the various algorithms, precision/recall and $F_1$ metrics were computed based on a gold extract created by the evaluator for one of the plays.

## 4. RELEVANCE RANKING

There are several algorithms employed by Information Retrieval systems to rank the relevance of a document in the context of a specific query [6]. A popular one, the *bag of words* model, assigns a weight to every term that is proportional to the number of occurrences of that term in the document. This weighting scheme is referred to as *term frequency* and is denoted $\mathrm{tf}_{t,d}$, with the subscripts denoting the term and the document in order. Although this model has its shortcomings (e.g., word order is ignored), it is simple and intuitive in that the content of the document is captured by the frequently occurring material[2] words.

At the same time, in order to attenuate the effect of words that occur too often in the collection of documents, the weight of a term is scaled by a factor proportional to the inverse of its *document frequency*, denoted $\mathrm{df}_t$. The composite weight of a term is thus defined as the product of its term frequency and inverse document frequency:

$$\begin{aligned}\text{tf-idf}_{t,d} &= \mathrm{tf}_{t,d} \times \mathrm{idf}_t \\ &= \mathrm{tf}_{t,d} \times \log \frac{N}{\mathrm{df}_t}\end{aligned}$$

where $N$ is the total number of documents in the collection.

Relevance Ranking for summarization then proceeds as described in algorithm 1.

```
Input : The text of a document, and a positive
        integer $l$
Output: A subset of sentences from the document
        with the highest tf-idf weights
1 Split the text of each play into sentences;
2 Compute the sentence score as the sum of the tf-idf
  weights of the stemmed version of the sentence;
3 Sort the sentences by score in non-increasing order;
4 Select the top $k$ sentences so that the total number
  of words is $l$;
```

**Algorithm 1:** Sentence ranking using tf-idf weights.

## 5. SENTENCE CLUSTERING

The $K$-Means clustering algorithm can be used to partition the set of sentences in a document into clusters so as to minimize the in-cluster distance metric, typically the square of the $L_2$ norm, captured in the following expression:

$$\underset{S}{\operatorname{argmin}} \sum_{i=1}^{K} \sum_{x_j \in S_i} \|x_j - \mu_i\|^2$$

The parameter $K$ is determined using *silhouette scoring*; we also take into account the constraint imposed by the desired summary size. A cluster count of 20 was found to be reasonable.

$K$-Means for sentence clustering is outlined in algorithm 2. The aim is to identify distinct topics in the document corresponding to the different clusters. A summary of the document is subsequently created by selecting the sentences closest to the centroids of the clusters.

## 6. SINGULAR VALUE DECOMPOSITION

We begin by stating the following theorem.

THEOREM 1. *Let $r$ be the rank of an $M \times N$ matrix $C$. Then, there is a Singular Value Decomposition (SVD) of $C$ of the form $\mathbf{C} = \mathbf{U\Sigma V^T}$ where*

1. *The eigenvalues $\lambda_1, \ldots, \lambda_r$ of $CC^T$ are the same as the eigenvalues of $C^TC$;*
2. *For $1 \le i \le r$, let $\sigma_i = \sqrt{\lambda_i}$, with $\lambda_i \ge \lambda_{i+1}$. Then the $M \times N$ matrix $\Sigma$ is composed by setting $\Sigma_{ii} = \sigma_i$ for $1 \le i \le r$, and zero otherwise.*

The values $\sigma_i$ are called the *singular values* of $C$. SVD can be used to compute a *low-rank approximation* to $C$.

[1]A graduate degree in English Literature.
[2]Stop words are excluded.

```
Input  : The text of a document, a positive integer
         $l$, and a positive integer $K$
Output: A subset of sentences from the document
         that represent the centroids of the
         sentence clusters
1  Split the text of each play into sentences;
2  Compute the *sentence score* as the sum of the
   normalized tf-idf weights of the *stemmed* version of
   the sentence;
3  Pick $K$ sentences $f_1, \ldots, f_k$ at random from the
   document as the initial seeds for the clusters;
4  while *stopping criterion is not true* do
5      for *all clusters* do
6          assign sentences to closest cluster;
7          re-compute cluster centroids;
8      end
9  end
10 Select the $K$ sentences closest to the centroids of the
   clusters;
11 Select $k \leq K$ sentences so that the total number of
   words is $l$;
```
**Algorithm 2:** Sentence clustering using $K$-Means.

This construct can be used to summarize a document as outlined in algorithm 3 [4]:

```
Input  : The text of a document, and a positive
         integer $l$
Output: A subset of salient sentences from the
         document
1  Split the text of each play into sentences;
2  Construct the (stemmed) terms-by-sentences matrix
   $A$ for the text of the document;
3  Perform SVD on $A$ to obtain the singular value
   matrix $\Sigma$, and the right singular vector matrix $V^T$;
4  Select the top $k$ sentences in order of non-increasing
   singular values so that the total number of words is
   $l$;
```
**Algorithm 3:** Sentence extraction using SVD.

This procedure identifies the latent semantic structure of the prose and captures it in the singular vectors. The algorithm then selects the salient sentences that correspond to underlying patterns of semantically related content.

## 7. RANDOM SURFER MODEL

The PageRank algorithm [11] is widely used in Web search engines as a ranking criterion. PageRank importance is determined by "votes" in the form of links from other pages on the Web. The idea is that votes (links) from important sites should carry more weight than votes (links) from less important sites, and the significance of a vote (link) from any source should be tempered (or scaled) by the number of sites the source is voting for (linking to). Computing PageRank thus boils down to solving a recursive equation, which can be expressed as the following eigenvector problem:

$$\pi^T G = \pi^T$$

Here, $G$ is a stochastic matrix, also known as the adjusted Google matrix. In general, the dominant eigenvalue for every stochastic matrix is 1. Consequently, the PageRank iteration converges to the normalized left-hand eigenvector $\pi^T$ which is the stationary (or steady-state) distribution of the Markov chain. A "teleportation" or damping factor of 0.85 was used in the implementation.

We adapt the PageRank algorithm for our purposes by creating a graph from the document we wish to summarize, where the vertices correspond to the sentences, and an edge between every pair of vertices having a weight that equals the cosine similarity between the two sentences it connects.

$$cos(\overrightarrow{u_1}, \overrightarrow{u_2}) = \frac{\overrightarrow{u_1} \cdot \overrightarrow{u_2}}{\|\overrightarrow{u_1}\| \ \ \|\overrightarrow{u_2}\|}$$

Running the PageRank algorithm on this graph then identifies the sentences with the highest importance. This is shown in algorithm 4.

```
Input  : The text of a document, and a positive
         integer $l$
Output: A subset of sentences from the document
         with the highest importance scores
1  Split the text of each play into sentences;
2  Express each sentence as a (sparse) vector in the
   multi-dimensional space of words;
3  Compute the cosine similarity between every pair of
   sentences;
4  Construct a graph $M = (V, E)$ from the document
   where each sentence is a node $u \in V$ and there is an
   edge $e = (u, w) \in E$ where $u$ and $w$ are sentences,
   and the weight of the edge is the cosine similarity
   between the sentences;
5  Compute the PageRank of each node;
6  Select sentences with the largest importance scores
   so that the total number of words is $l$;
```
**Algorithm 4:** Sentence selection using PageRank.

## 8. IMPLEMENTATION NOTES

The majority of the code was written in Python 2.7 using routines from NumPy [12] and NetworkX [13]. Some ad hoc scripts were written in `bash` to preprocess the raw text and apply the various algorithms to the normalized document contents.

The complete code and data for the project is available at the author's Github repository [14].

## 9. RESULTS AND ANALYSIS

The results of the extraction algorithms for six plays are included in this report. Table 1 shows the evaluation of the results from the different algorithms for the play *Julius Caesar* using unigrams, and table 2 shows the same using trigrams. These metrics are not unlike ROGUE, or Recall-Oriented Understudy for Gisting Evaluation [9], that is traditionally

**Table 1: Evaluation of the summary extracts against the gold version for *Julius Caesar* using unigrams.**

| Algorithm | Grade | Precision | Recall | $F_1$ |
|---|---|---|---|---|
| tf-idf | B | 0.1555 | 0.3784 | 0.2205 |
| SVD | C+ | 0.0984 | 0.1622 | 0.1224 |
| K-Means | B− | 0.2549 | 0.3514 | **0.2954** |
| PageRank | B | 0.0984 | 0.1622 | 0.1224 |

**Table 2: Evaluation of the summary extracts against the gold version for *Julius Caesar* using trigrams.**

| Algorithm | Grade | Precision | Recall | $F_1$ |
|---|---|---|---|---|
| tf-idf | B | 0.0344 | 0.2500 | 0.0606 |
| SVD | C+ | 0.0000 | 0.0000 | 0.0000 |
| K-Means | B− | 0.0667 | 0.2500 | 0.1053 |
| PageRank | B | 0.1296 | 0.4375 | **0.2000** |

used in the research community to automatically evaluate summarizer performance. The generated extracts are displayed in table 3. With an average length of almost 24,000 words per play, the 100-word summary extracts amount to only 0.4% of the entire text. It is indeed a daunting task to capture the essence of any creative work of literature in a short span of one hundred words. There isn't a single compression algorithm in the world that offers the kind of summarization targeted in this research.

Table 4 shows the gold summary created by the evaluator by hand-picking sentences that were deemed salient in the play *Julius Caesar*; sentences that also appear in at least one algorithmically-generated summary are underlined in color. Precision, recall, and $F_1$ metrics for each algorithm were subsequently computed using counts of overlapping unigrams and trigrams in the target summary and the gold version. Trigrams were used in order to offset the lack of availability of multiple evaluators, and any subjective bias that the human judge may have introduced in generating the gold summary.

In the traditional Vector Space Model from the field of IR as employed in the tf-idf ranking algorithm, only similarities between sentences or between a query and the sentences in a document can be calculated within one space. If terms were to be compared to each other another space would have to be drawn. In a term space, where the terms represent the dimensions, the terms are considered to be linearly independent, which means their relations to each other are not taken into account. Furthermore in VSM the similarity calculation is based only on word matching. Each dimensions of a vector corresponds to a term. Two documents with a similar topic but different vocabulary will not be placed next to each other. Only documents that overlap in vocabulary will be considered similar.

The SVD algorithm preserves as much information as possible about the relative distances between the sentence vectors, while collapsing them down into a much smaller set of dimensions. In this collapse, information is lost, and content words are superimposed on one another. Information loss sounds like a bad thing, but here it is a blessing. What we are losing is noise from our original term-sentence matrix, revealing similarities that were latent in the document. Similar things become more similar, while dissimilar things remain distinct. This reductive mapping is what gives SVD its seemingly intelligent behavior of being able to correlate semantically related terms. We are really exploiting a property of natural language, namely that words with similar meaning tend to occur together [8]. SVD thus re-expresses a co-occurrence matrix in a new coordinate system. The idea is to uncover the latent semantic structure of a collection of sentences, i.e., to find hidden relations between terms or other text units by using high-order co-occurrence. Unlike VSM and its kin that rely on literal word overlap for similarity calculation, SVD relies on a derived semantic relatedness measure. This measure reflects the semantic similarity between words that are used in similar context, e.g., synonyms, antonyms, hyponyms or compounds.

Even though SVD didn't do all that well in terms of the $F_1$ measure, it did reasonably well in qualitative terms. I would attribute SVD's poor $F_1$ score to lack of human-generated gold extracts—ideally, not only should we have gold extracts for all documents (plays), we should have *multiple* extracts for each document to serve as reference summaries, along the lines of how machine translation output is evaluated. Unfortunately, doing so requires a serious commitment of time and effort on the part of qualified human evaluators, who are unlikely to sign up for this task without compensation.

Clustering performed better than expected in picking out the set of topics in each play. A straightforward application of an implementation of Lloyd's algorithm appears to have produced reasonably good results.

PageRank did best among the unsupervised algorithms explored in this project. This is not entirely surprising; it is already used as a measure of importance by all commercial web search engines. What is intriguing is the fact that it worked well on an undirected graph—recall that the World Wide Web is a directed graph: millions of pages on the Internet point to `www.stanford.edu`, but there are few links originating from Stanford's home page. In contrast, the links between sentences in the document are symmetric: the weight of the edge connecting a sentence to another is the cosine similarity of the two. So, in effect, we are really looking for sentences that are "highly connected" to others in the document.

In general, we seem to have obtained good mileage from unsupervised techniques without any aid of semantics. It is indeed remarkable that we could achieve reasonably good results without introducing linguistics, except in the preprocessing stages.

## 10. CONCLUSIONS AND FURTHER WORK

Not surprisingly, summarizing Shakespeare in a hundred words is an ambitious goal, and one that is hard to attain

**Table 3: Summary extracts generated by the various algorithms for the play *Julius Caesar*.**

| Using *tf-idf* Relevance Ranking | Using SVD |
|---|---|
| You all do know this mantle: remember The first time ever Caesar put it on; 'Twas on a summer's evening, in his tent, That day he overcame the Nervii: Look, in this place ran Cassius' dagger through: See what a rent the envious Casca made: Through this the well-beloved Brutus stabb'd; And as he pluck'd his cursed steel away, Mark how the blood of Caesar follow'd it, For Brutus, as you know, was Caesar's angel: Judge, you gods, how dearly Caesar loved him! come not, friends, to steal away your hearts: am no orator, as Brutus is; But, as you know me all, a plain blunt man, That love my friend; For have neither wit, nor words, nor worth, Action, nor utterance, nor the power of speech, To stir men's blood: only speak right on; tell you that which you yourselves do know; In every wound of Caesar that should move The stones of Rome to rise and mutiny. | Caesar doth bear me hard; but he loves Brutus: If were Brutus now and he were Cassius, He should not humour me. Fourth Citizen Caesar's better parts Shall be crown'd in Brutus. Third Citizen Nay, that's certain: We are blest that Rome is rid of him. Fill, Lucius, till the wine o'erswell the cup; cannot drink too much of Brutus' love. Messala, have here received letters, That young Octavius and Mark Antony Come down upon us with a mighty power, Bending their expedition toward Philippi. Exit Farewell, good Messala: Good night, Titinius. By your leave, gods:-this is a Roman's part Come, Cassius' sword, and find Titinius' heart. |
| **Using *K*-Means Clustering** | **Random Surfer Model: PageRank** |
| what! Our course will seem too bloody, Caius Cassius, To cut the head off and then hack the limbs, Like wrath in death and envy afterwards; For Antony is but a limb of Caesar: Let us be sacrificers, but not butchers, Caius. Enter reading a paper 'Caesar, beware of Brutus; take heed of Cassius; come not near Casca; have an eye to Cinna, trust not Trebonius: mark well Metellus Cimber: Decius Brutus loves thee not: thou hast wronged Caius Ligarius. Fourth Citizen Read the will; we'll hear it, Antony; You shall read us the will, Caesar's will. Fourth Citizen It is no matter, his name's Cinna; pluck but his name out of his heart, and turn him going. | Brutus and Caesar: what should be in that 'Caesar'? Enter reading a paper 'Caesar, beware of Brutus; take heed of Cassius; come not near Casca; have an eye to Cinna, trust not Trebonius: mark well Metellus Cimber: Decius Brutus loves thee not: thou hast wronged Caius Ligarius. have done no more to Caesar than you shall do to Brutus. Fourth Citizen Caesar's better parts Shall be crown'd in Brutus. Here, under leave of Brutus and the rest- For Brutus is an honourable man; So are they all, all honourable men- Come to speak in Caesar's funeral. Fourth Citizen Read the will; we'll hear it, Antony; You shall read us the will, Caesar's will. |

**Table 4: Gold extract for the play *Julius Caesar*.**

| |
|---|
| Enter the conspirators, CASSIUS, CASCA, DECIUS BRUTUS, CINNA, METELLUS CIMBER, and TREBONIUS. Caesar, beware of Brutus; take heed of Cassius; come not near Casca; have an eye to Cinna, trust not Trebonius: mark well Metellus Cimber: Decius Brutus loves thee not. Et tu, Brutus? Not that I loved Caesar less, but that I loved Rome more. Friends, Romans, countrymen, lend me your ears. For Brutus is an honourable man. So are they all, all honourable men. In every wound of Caesar that should move The stones of Rome to rise and mutiny. Revenge! About! Seek! Burn! Fire! Kill! Slay! Let not a traitor live! |

in practice even for a person with deep knowledge of the classics.

But we are not done yet. There are plenty of interesting avenues to explore. Most notably, we could perhaps use part of the dataset (i.e., a subset of the thirty-seven plays) to train a classifier that can subsequently be used to determine whether a sentence in a document from the test set should be included in its summary. In addition, there are other graph-based algorithms including *HITS* [10] that attempt to identify axial nodes in a graph using a different measure of centrality.

Finally, instead of, or perhaps in addition to, sentence extraction, research from the burgeoning field of text generation could be applied to produce abstractive summaries. This is arguably a harder road, but potentially a promising one that is well worth exploring.

## 11. ACKNOWLEDGEMENTS

I would like to thank Ms. Megha Malhotra for lending her expertise in English Literature in evaluating the machine-generated summaries, as well as in providing a reference summary for *Julius Caesar*.

## 12. REFERENCES

[1] IBM, `http://www.ibm.com/software/data/bigdata`

[2] Nenkova, Ani and McKeown, Kathleen, *Automatic Summarization*, Foundations and Trends in Information Retrieval, 2011.

[3] Bosak, Jon, *The Complete Plays of Shakespeare marked up in XML*, available for use without restrictions at `http://www.ibiblio.org/xml/examples/shakespeare`.

[4] Gong, Yihong and Liu, Xin, *Generic Text Summarization Using Relevance Measure and Latent Semantic Analysis*, SIGIR 2001.

[5] Steinberger, Josef, and Ježek, Karel, *Evaluation Measures for Text Summarization*, Computing and Informatics, Vol. 28, 2009.

[6] Manning, Christopher D., Raghavan, Prabhakar, and Schütze, Hinrich, *An Introduction to Information Retrieval*, Cambridge University Press, 2009.

[7] Erkan, G., and Radev, D. R., *LexRank: Graph-based Lexical Centrality as Salience in Text Summarization*, Journal of Artificial Intelligence Research, 2004.

[8] Yu, C., Cuadrado, J., Ceglowski, M., and Payne, J. S. *Patterns in Unstructured Data - Discovery, Aggregation, and Visualization*, 2002.

[9] ROGUE, `http://www.berouge.com/Pages/default.aspx`

[10] Kleinberg, Jon, *Authoritative sources in a hyperlinked environment*, Journal of the ACM 46 (5): 604-632, 1999.

[11] Brin, S., Page, L., Motwani, R., and Winograd, T., *The PageRank Citation Ranking: Bringing Order to the Web*, Technical Report 1999-0120, Computer Science Department, Stanford University, Stanford, CA, 1999.

[12] NumPy, `http://www.numpy.org`

[13] NetworkX, `http://networkx.github.io`

[14] Bajaj, Samir, GitHub Repository: `samirbajaj/cs224n-project`