# Extractive Fiction Summarization Using Sentence Significance Scoring Models

Stanford University
Eric Holmdahl | holmdahl@stanford.edu | CS 221 and CS 229
Ashkon Farhangi | ashkonf@stanford.edu | CS 224N and CS 221
Lucio Tan | lctan21@stanford.edu | CS 221

## Abstract

This paper describes experiments in generating extractive summaries of fictional short stories using supervised and unsupervised methods. We first experiment with different sentence significant scoring models. We describe the use of TextRank and word frequency analysis focusing on verbs, nouns and adjectives to determine the significance of individual words within sentences. We then experiment with sentence clustering using both standard and modified versions of the K-Means algorithm to find sentences to which many others are similar. The final result of these experiments yielded an ensemble model, using the outputs of our other sentence significance scorers as features provided to a Gaussian Naive Bayes sentence significance classifier. To use the output of the sentence significance scoring models, we experiment with multiple sentence extraction algorithms. We first implement a greedy sentence extraction algorithm, and then experiment with modeling the sentence extraction problem as a constraint satisfaction problem. We evaluate the performance of this system with the ROUGE summary evaluation library, which compares system-generated summaries to human-generate gold summaries and uses the similarity between them as a proxy for quality.

## 1 Introduction

The nuance of human language has always made text summarization a difficult problem, especially with regards to fiction. Modern attempts to tackle this challenging task often focus on unsupervised methods to identify key topics, but generally focus on nonfiction texts. Some more experimental and cutting edge systems have begun to utilize machine learning and natural language understanding methods, but have done so with varying degrees of success.

In this paper we will describe our approach to the problem of automated text summarization, the theory behind models we built to pursue it and our implementation of algorithms necessary to complete the task. Because the task of generating text from scratch is extremely difficult, we chose to avoid attempting to create generative summaries and instead focused on extractive summarization. The primary challenge in extractive summarization is the identification of the most important sentences within a text, and we focus our efforts on tackling this challenge.

To limit the length of summaries produced by our system, we impose a constraint on the extractive summaries produced by our algorithm: we require that summaries be fewer than 500 words in length.

## 2 Current State of Field

There is no dearth of projects in the field of text summarization. Some more promising works we came across during our research pointed towards the use of Latent Dirichlet Allocation (LDA) and variations of the PageRank algorithm for generating topic models for texts. These methods generate lists of key topics and then extract sentences from original texts that correlate with those topics. Of the several PageRank derivatives that have been employed, TextRank appears to be the best performing and most commonly used, with LexRank also being quite popular. There has been very little work done in the field of summarizing fiction, and almost all of the literature describing implementations of the aforementioned algorithms have focused on the summarization of scientific abstracts.

## 3 Dataset

We decided to restrict the scope of our project to the summarization of short stories, in general no longer than 10 pages. To acquire training and test data, we scraped a number of public web sites. We obtained more than 170 short stories and 3-4 associated human-generated summaries for each story. These stories contain an average of around 100 sentences each, resulting in a data set of roughly 17000 sentences. Due to copyright laws, only significantly old stories are readily available free of charge online. As a result, our dataset consists almost exclusively of classical stories from the 1800s. Authors such as Edgar Allen Poe and Mark Twain appear repeatedly in our data set.

## 4 Sentence Significance Scoring Methods

Our system consists of two primary components. Because our system generates extractive summaries, we must first identify important sentences from within stories to extract. In the following section we discuss the different sentence significance scoring models we implemented. Afterwards, we will discuss the specific ways in which we use those significance scores to generate summaries. We implemented several different supervised and unsupervised algorithms for sentence significance scoring, achieving mixed results.

### 4.1 TextRank

TextRank is an unsupervised keyword extraction algorithm with the primary purpose of determining the most relevant keywords in a text. It models a given text by representing it as a graph (with unique words as nodes and edges representing word co-occurrence) and scoring the keywords by running the PageRank algorithm on the aforementioned graph [5].

TextRank is one of the top keyword extraction methods for documents such as scientific papers by f-measure, but has not yet been extensively tested for the purpose of fiction keyword extraction.

In constructing and modeling the graph, we have each node represent each unique word in the text. Note that we only care about words whose significance we want to represent to other words, so we filter out words that are not nouns and adjectives. Edges are then added by iterating over the filtered text word by word, adding a directed edge (if there is not already one) from each word iterated over to the n words that succeed it, where n is our window size.

Subsequently, the PageRank algorithm is run on the graph and runs as follows:

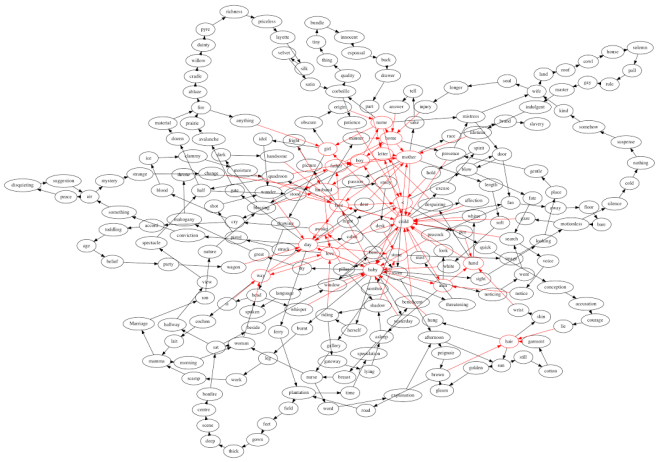
**Until convergence:**
**For $S_i$ in TextRank graph:**

$$Score(S_i) = (1-d) + d * \sum_{S_j} \frac{1}{|S_j|} * Score(S_j)$$

Note that $S_j$ are the nodes that have an edge pointing to $S_i$, and $|S_j|$ is the number of edges out of $S_j$. PageRank was designed to score the importance of webpages by scoring pages by the importance of links pointing to a given webpage, and TextRank works analogously for words in a text. A node's score is dependent on the scores of the nodes that have edges pointing into it, giving higher scores to nodes that co-occur with nodes that commonly co-occur.

After generating the node scores, TextRank selects the 20 nodes that have the highest scores as potential keywords and then concatenates any of the potential keywords that have an edge between them, into one keyword such that it can pick up multi-word keywords.

Below, we can see the graph generated by our implementation of TextRank on an example story, with the nodes selected as keywords highlighted in red.



Intuitively, this algorithm is effective since it leverages the idea that units of text 'recommend' one another. If two words co-occur frequently, then there is probably some relation between them. If a word has a lot of other words related to it, then it is most likely an important word in the text and its relations should reflect that.

## 4.2 Noun and Adjective Significance Scoring

The nouns and adjectives a sentence contains can strongly indicate the significance of that sentence. To capture this intuition, we build a model that predicts the significance of a sentence based on the nouns and adjectives it contains, as well as the location of that sentence relative to the times at which the nouns and adjectives it contains appear the most in the story it appears in.

### 4.2.1 Linguistic Intuition

Some nouns and adjectives tend to be very characteristic of the stories in which they appear. Character names, for instance, are almost completely unique to the stories to which they belong, as authors frequently invent new character names for each of their works. However, not all nouns are characteristic of the stories in which they appear. Common nouns such as "person" and "food" appear in most works of fiction. To identify nouns and adjectives that are especially significant to a story, we look for nouns and adjectives that appear much more frequently in the story than they do in the English language as a whole.

However, the importance of a sentence depends on more than just the nouns and adjectives it contains. Rare nouns and adjectives tend to appear in clumps in stories. Certain events described in stories relate more closely to certain characters, locations or other rare nouns occurring frequently in the story. Consequently, the significance of a noun or adjective found within a sentence also depends on the location of the sentence relative to the epicenter of mentions of that noun or adjective.

We assume that rare noun and adjective occurrences conform to a Truncated Gaussian distribution. As mentioned before, mentions of rare nouns and adjectives in a story tend to be concentrated around a single point in that story at which they become extremely relevant. Leading up to such a point, mentions increase in frequency and after such a point, mentions decrease in frequency. This observation led us to believe the Gaussian distribution would fit the distributions of rare noun and adjective mentions well. Assuming mentions of nouns and adjectives are normally distributed may be deeply flawed, but anecdotal observations suggested that a normal distribution would represent word occurrence data reasonably more often than any other standard distribution.

We use the Truncated Gaussian distribution instead of the standard Gaussian distribution because stories contain a fixed number of sentences, and words cannot appear before the first sentence or after the last one. Imposing these bounds on the Gaussian distribution complicates the distribution's probability distribution function (PDF). The standard Gaussian distribution exhibits the following PDF:

$$\mathcal{N}(x) = \frac{\exp\left(\frac{-(x-\mu)^2}{2\sigma^2}\right)}{\sigma\sqrt{2\pi}}$$

$\sigma$ is the standard deviation of the Gaussian distribution and $\mu$ is its mean. The Truncated Gaussian distribution's more complicated PDF appears below:

$$\mathcal{N}(x) = \frac{2 * \exp\left(-\left(\frac{x-\mu}{\sigma}\right)^2\right)}{\sigma * \left[\left(1 + \operatorname{erf}\left(\frac{\beta-\mu}{\sigma\sqrt{2}}\right)\right) - \left(1 + \operatorname{erf}\left(\frac{\alpha-\mu}{\sigma\sqrt{2}}\right)\right)\right] * \sqrt{2\pi}}$$

The PDF of the Truncated Gaussian distribution is slightly more complicated and requires two additional parameters. $\alpha$ represents the beginning of the distribution's bounds and $\beta$ represents the end of its bounds. $\alpha$ and $\beta$ are the Truncated Gaussian distribution's points of truncation.

### 4.2.2 Unsupervised Learning for Word Distribution Fitting

For each noun and adjective, we fit a Truncated Gaussian distribution to the locations of mentions of that lemma. Note that we use the lemmas of nouns and adjectives and not the raw forms of those words. For each mention of such a lemma, we consider the index of the sentence in which it appears within the story to be its location. To estimate the mean and variance of the Truncated Gaussian distribution, we use Maximum Likelihood Estimation (MLE). The maximum likelihood estimate of the mean of a Gaussian distribution is the sample mean of the observed points we believe belong to it. The maximum likelihood estimate of the variance of a Gaussian distribution is the sample variance of the observed points we believe belong to it. The formulas for calculating sample mean and variance appear below:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \qquad s^2 = \frac{\sum_{i=1}^{n}(x_i - \bar{x})^2}{n}$$

To estimate α and β, we used a much simpler method. We know exactly where stories begin and end, and know that words can appear within the bounds of a story but not outside of them. As such, set α to 0 (representing the first sentence) and β to one less than the number of sentences in the story (representing the last sentence). This process of parameter estimation leaves us with a distribution representing the locations of mentions of each noun and adjective lemma appearing in a story.

#### 4.2.3 Noun and Adjective Significance Scoring

To predict the significance of the mention of a noun or adjective within a story, we consider two factors: its location within the story, and its commonality in the story relative to its commonality in the English language. We have already learned the distribution of each noun and adjective within the story we consider. Consequently, to derive the significance score of the mention of a noun or adjective we first obtain the PDF value of the mention's location (the index of the sentence in which it appears), which we will call the location-based relevance of the mention. This closer the mention is to the epicenter of equivalent mentions, the larger this PDF value will be. We then multiply this PDF value by what we call the absolute relevance of the word to the story: its unigram probability of occurring within the story divided by its unigram probability of occurring in the English language. To obtain the unigram probability that the word appears in the English language, we train a standard language model on the Brown University Standard Corpus of Present-Day American English (commonly referred to as the Brown Corpus). In calculating both unigram probabilities we use Laplace smoothing. An expression for the final significance of a word w (noun or adjective) mention appears below:

$$\text{Score}(w) = \mathcal{N}(\ell) * \frac{\text{P}_{story}(w)}{\text{P}_{English}(w)} = \mathcal{N}(\ell) * \frac{\left[\frac{c_{story}(w)+1}{\text{N}_{story}+\text{n}_{\text{story}}}\right]}{\left[\frac{c_{Brown}(w)+1}{\text{N}_{Brown}+\text{n}_{\text{Brown}}}\right]}$$

$\ell$ is the location of the word in the story, $\mathcal{N}(\ell)$ is the PDF of the Truncated Gaussian distribution, N is the number of words in the specified corpus, and n is the number of unique words in the specified corpus.

#### 4.2.4 Sentence Significance Prediction

Finally, we aggregate the significance scores corresponding to all the nouns and adjectives contained in a sentence to attain the cumulative significance score of the sentence according to this model. We use an extremely simple aggregation method, using the arithmetic mean of the significance scores of the nouns and adjectives within a sentence as its significance score. Note that this model completely ignores all words that are not nouns or adjectives.

### 4.3 Verb Significance Scoring

The verbs a sentence contains can strongly indicate the significance of that sentence as well. To capture this intuition, we build a model that predicts the significance of a sentence based on the verbs it contains.

#### 4.3.1 Linguistic Intuition

Unlike nouns and adjectives, verbs are rarely unique to the stories in which they appear. People talk, walk, eat, and fight in almost every story imaginable. There are certainly exceptions to this trend, but for the most parts it holds. This is not to say that all stories contain the exact same verbs in the exact same quantities. Distributions of verbs deviate substantially from one story to the next, and we take this into account.

While verbs may not be characteristic of stories in which they appear, verbs can, much more easily than nouns and adjectives, be classified as significant or unimportant. Verbs that describe common actions, such as speaking or eating, in general describe minimally important events. However, other verbs such as "to fight" or even more so "to kill" indicate extremely important events almost every time they are mentioned. Of course, one cannot cleanly label verbs as important or unimportant. Every verb's significance falls along some theoretical continuous scale of significance.

However, a verb's significance within a story depends on more than its significance in an absolute sense. Some stories include a lot of very significant verbs, while some do not. Every story has a different average verb significance score, and the significance of a sentence to a story depends not on the absolute significance of the verbs it contains but the significance of verbs it contains relative to the story's average.

#### 4.3.2 Supervised Learning for Absolute Verb Significance Scoring

First, we learn absolute verb lemma significance scores in a supervised manner. Note that we use the lemmas of verbs instead of their raw forms. We base this learning process on the intuition that a verb that appears much more often in a summary than it does in a story must have been extremely important to that story, while a verb that appeared many times in a story but never in its summary must have been too menial to deserve mention in the summary. We learn the absolute significance score of a verb by comparing that verb's commonality in stories to its commonality in summaries, as observed in our training set of stories and associated gold summaries. To be precise, we consider a verb's absolute significance score to be its probability of occurring in a summary divided by its probability of occurring in a story. In calculating these probabilities, we use Laplace smoothing. The expression for arbitrary verb v's absolute significance score given training set D appears below:

$$\text{AbsoluteScore}(v, D) = \frac{\text{P}_{summary}(v, D)}{\text{P}_{story}(v, D)} = \frac{\left[\frac{\text{c}_{summary}(v, D)+1}{\text{N}+\text{n}}\right]}{\left[\frac{\text{c}_{story}(v, D)+1}{\text{N}+\text{n}}\right]} = \frac{\text{c}_{summary}(v, D)+1}{\text{c}_{story}(v, D)+1}$$

N is the number of verb mentions in the training set (across both stories and summaries) and n is the number of unique verbs found in the training set. c is the count function, that represents the number of instances in training in which its input condition is met.

#### 4.3.3 Unsupervised Learning for Story-Specific Verb Significance Scoring

To generate story-specific verb significance score, we compare a verb's significance to that of other verbs appearing in the story in a very simple fashion. We consider a verb mention's story-specific significance score to be the percentile at which its absolute significance score falls in comparison to the other verb mentions in the story. An expression for this score for verb v, in story d, given training set D, follows:

$$\text{StorySpecificScore}(v, d, D) = \frac{\sum_{v' \in d} 1[\text{AbsoluteScore}(v', D) < \text{AbsoluteScore}(v, D)]}{|d|}$$

#### 4.3.4 Sentence Significance Prediction

Finally, we aggregate the significance scores corresponding to all of the verbs contained in a sentence to attain the cumulative significance score of the sentence, according to this model. We use

the same aggregation method used by the noun and adjective scoring method described previously, using the arithmetic mean of the significance scores of the verbs within a sentence as its significance score. Note that this model completely ignores all words that are not verbs.

## 4.4 Sentence Clustering

Stories often contain many similar sentences that express relatively similar ideas. Unlike nonfiction, works of fiction do not strive for maximum possible conciseness. Intuitively, sentences to which many other sentences in a story are extremely similar tend to include important ideas. To utilize this intuition, we cluster sentences.

To model sentences in a form acceptable as input to a clustering algorithm, we represent sentences by their TF-IDF vectors. The TF-IDF metric is a score representing the significance of a word to a document within a corpus. A word's Term Frequency (TF) is the number of times it appears within the document. A word's Inverse Document Frequency is the reciprocal of the number of documents within the greater corpus within which the word appears. An expression representing the TF-IDF metric for term t, document d and corpus D follows:

$$\text{TF-IDF}(t,d,D) = \text{TF}(t,d) * \text{IDF}(t,D) = \frac{\sum_{t' \in d} 1[t'=t]}{\sum_{d' \in D} 1[t \in d']}$$

We treat sentences as documents and the story in which those sentences reside as the corpus. A sentence's TF-IDF vector is the vector of the TF-IDF scores of the words it contains.

### 4.4.1 Standard K-Means

Because of the sparsity of TF-IDF vectors, we implement each TF-IDF vector a dictionary mapping words in the sentence to their associated TF-IDF scores. We cluster sentences' representative TF-IDF vectors using the K-Means algorithm. The algorithm clusters objects into K clusters, where K is a positive integer provided to it as a parameter. We chose a value of K equal to the quotient of the number of sentences in a story and 5, to obtain an average of 5 sentences per cluster. This standard K-Means algorithm attempts to minimize the Reconstruction Loss function. That function follows:

$$\text{ReconstructionLoss}(z,\mu) = \sum_{i=1}^{n} \left\| \phi(x_i) - \mu_{z_i} \right\|_2^2$$

$\phi(x_i)$ is the TF-IDF vector corresponding to the ith sentence, $\mu$ is the vector of centroids and z is the vector of sentence assignments to clusters.

### 4.4.2 K-Means with Angular Loss Function

K-mean clustering using standard Reconstruction Loss minimizes the sum of the squared Euclidean distances between sentences and their centroids. This results in clusters of similar sentences, for which similarity is defined by the Euclidean distances between their TF-IDF vectors. However, similarity between sentences is measured most frequently as the angle between the corresponding TF-IDF vectors, and not the Euclidean distance between them. To cluster sentences based on the definition of angles between TF-IDF vectors as similarity, we altered the standard K-mean algorithm. To account for this, we replaced the standard Reconstruction Loss function with a modified loss function, which we will call "Angular Loss". The definition of Angular Loss follows:

$$\text{AngularLoss}(z,\mu) = \sum_{i=1}^{n} \left( angle\left(\phi(x_i), \mu_{z_i}\right)\right)^2 = \sum_{i=1}^{n} \left( \cos^{-1}\left( \frac{\phi(x_i) \circ \mu_{z_i}}{\|\phi(x_i)\|_2 * \left\|\mu_{z_i}\right\|_2} \right)\right)^2$$

It represents the sum of the squares of the angles between points and the centroids of their clusters. Instead of minimizing the squared Euclidian Distance between TF-IDF vectors, this loss function will result in the minimization of the squared angles between those vectors.

The use of this loss function presented a new challenge. To find the optimal centroid given a cluster of points, we could not simply take the average of those points because doing so would minimize Reconstruction Loss and not Angular Loss. As a result, we had to find a different method to minimize Angular Loss in calculating centroids as part of the K-Means algorithm. Given the complexity of the loss function, we decided to use Stochastic Gradient Descent (SGD) to find the centroid that minimizes the loss with respect to all the point in its associated cluster. The gradient of Angular Loss follows:

$$\text{Loss}(z,\mu,i) = \left( \cos^{-1}\left( \frac{\phi(x_i) \circ \mu_{z_i}}{\|\phi(x_i)\|_2 * \left\|\mu_{z_i}\right\|_2} \right)\right)^2$$

$$\text{Let A} = \frac{\phi(x_i) \circ \mu_{z_i}}{\|\phi(x_i)\|_2 * \left\|\mu_{z_i}\right\|_2}$$

$$\nabla \text{Loss}(z,\mu,i) = \nabla(\cos^{-1}(A))^2 = \frac{-2 * \cos^{-1}(A)}{\sqrt{1-A}} * \nabla A$$

$$= \frac{-2 * \cos^{-1}(A)}{\sqrt{1-A}} * \frac{\left[ \left\|\mu_{z_i}\right\|_2 \|\phi(x_i)\|_2 * \phi(x_i) + \frac{\mu_{z_i} \circ \phi(x_i) * \|\phi(x_i)\|_2}{\left\|\mu_{z_i}\right\|_2} * \mu_{z_i} \right]}{\left( \left\|\mu_{z_i}\right\|_2 \|\phi(x_i)\|_2 \right)^2}$$

To find the centroid that minimizes Angular Loss with respect to the point in its cluster, we run 25 iterations of SGD, updating the centroid on each iteration according to the gradient of the Angular Loss function with respect to each point in the cluster one at a time. We use a step size equivalent to $1/\sqrt{iteration\ index}$. Instead of randomly initializing centroids at the start of SGD, we initialize centroids to the average of all of the points within those centroids' associated clusters.

### 4.4.3 Use of Clusters

Finally, we define the significance score for each sentence according to this model as the number of sentences in its cluster.

## 4.5 SVR with Estimated Training Data

### 4.5.1 Gold Sentence Significance Estimation

We do not have explicit training data for the task of important sentence identification, as we do not have stories with sentences labeled as important or otherwise. However, we do have a large data set of stories and associated summaries. Based on this data set, we attempt to estimate sentence importance. For each sentence in a story, we measure the angle between its TF-IDF vector and that of each sentence in the story's summary. We take the inverse of the lowest angle between the sentence and any of the sentences in the summary to be the sentence's significance score.

### 4.5.2 Support Vector Regression (SVR)

To train a Support Vector Regressor (SVR), we calculate significance scores as previously described for a training set of sentences, featurize those sentences and feed them into a standard SVR implementation as training data. To use the trained SVR, we later featurize sentences for which we don't have associated summaries (i.e. those we want to summarize) and use the SVR's significance prediction to score sentences in terms of importance. Of the approaches we tried, this one performed by far the worst. It was immediately obvious upon testing this regression model that it would perform terribly. It produced an average squared error of 0.018 while the variance of values it attempted to predict was

0.023. Such a small difference in variance of underlying values and squared mean squared error indicates poor predictive power. As a result, we decided to drop this model from consideration.

## 4.6 Multinomial Naive Bayes with Simple Features

### 4.6.1 Motivation and Strategy

As noted, we do not have explicit training data for the task of important sentence identification. However, we did obtain a sizeable data set of stories and associated summaries. Previously we discussed using gold summaries to estimate the importance of sentences in their associated stories. Upon the failure of this approach, we decided to try a different strategy; using sentences in summaries themselves as examples of important sentences. We made the assumption that every sentence in a story would be insignificant, and that every sentence in its associated summary would be significant. The assumption that every sentence in a story would be insignificant was of course flawed (in fact the idea of summarization via extraction relies on its falsehood), but the average story sentence would certainly be much less significant than the average summary sentence. We used this assumption to treat the placement of sentences as a proxy for their importance, and to train a classifier to predict whether an input sentence belongs in a story or summary. Note that ultimately, we needed to predict the significance of sentences in stories. We intended to treat the classifier's confidence that such an input sentence belongs in a summary as a proxy for that sentence's significance, despite the fact that we would know with complete certainty that every sentence fed to the classifier in testing would belong in a story.

### 4.6.2 Supervised Learning

We decided to use a Multinomial Naive Bayes classifier to learn to classify sentences as belonging in a story or summary, and used its estimated probability of a sentence belonging in a summary as that sentence's significance score according to this model. We featurize sentences using a relatively simple method that captures some of their binary characteristics. Features we used include boolean features indicating whether or not a sentence includes a question mark, whether or not it includes an exclamation point, whether or not it includes at least one verb, whether or not it includes at least one proper noun, and a myriad of other similar features. The intuition behind these features was that sentences in summaries almost exclusively do not contain question marks of exclamation points, almost exclusively contain verbs and contain proper nouns more often than not, while the same cannot be said for sentences found in stories. As a result, such features should provide rich insights into the extent to which sentences are prototypical of those found in summaries, and by extension their likelihood of being sufficiently significant to be extracted into a summary.

Our Multinomial Naive Bayes model uses Laplace Smoothing in estimating probabilities. Ultimately, this model's goal is to estimate $P(S \mid \phi)$, the probability that a sentence belongs in a summary given its feature vector, because we use this probability as a sentence's significance score. The steps leading the to final expression for $P(S \mid \phi)$ according to the Multinomial Naïve Bayes appear below, preceded by definitions.

$$D = Training\ Set \qquad \phi = Feature\ Vector$$

$$x_j \in D = Training\ Example\ j$$

$$T_j => x_j\ in\ text \qquad S_j => x_j\ in\ summary$$

$$c(T) = \sum_j 1[x_j\ in\ text] \qquad c(S) = \sum_j 1[x_j\ in\ summary]$$

$$c(\phi_i, T) = \sum_j 1[x_j \text{ in text and has feature } \phi_i = 1]$$

$$c(\phi_i, S) = \sum_j 1[x_j \text{ in summary and has feature } \phi_i = 1]$$

$$Y \in \{S, T\} \qquad P(T) = \frac{c(T)}{|D|} \qquad P(S) = \frac{c(S)}{|D|}$$

$$P(S|\phi) = \frac{[P(S) * P(\phi|S)]}{[P(T) * P(\phi|T) + P(S) * P(\phi|S)]}$$

By the Naïve Bayes assumption:

$$\approx \frac{[P(S) * \prod_{i=1}^{n} P(\phi_i|S)]}{[P(T) * \prod_{i=1}^{n} P(\phi_i|T) + P(S) * \prod_{i=1}^{n} P(\phi_i|S)]}$$

$$= \frac{[c(S)^{n-1} * \prod_{i=1}^{n} (C(\phi_i, S) + 1)]}{[(c(T)^{n-1} * \prod_{i=1}^{n} (c(\phi_i, T) + 1)) + (C(S)^{n-1} * \prod_{i=1}^{n} (c(\phi_i, S) + 1))]}$$

### 4.6.3 Sentence Significance Scores

We use the computed values of these probability distributions to estimate the probability that each sentence in a story belongs in a summary according to the Multinomial Naive Bayes model. We use such an estimated probability of a sentence belonging in a summary as that sentence's significance score according to this model

### 4.6.4 Examples

To further illustrate the performance of this Multinomial Naive Bayes model, Below is a subset of the binary features we used in our Multinomial Naive Bayes model, and their associated conditional probabilities. When $P(\phi \mid S)$ is larger than $P(\phi \mid T)$, the presence of such a feature in a sentence will reduce the Naive Bayes model's estimated probability that the sentence is in a summary by less than it does the estimated probability that the sentence is in a story (refer to the derivations of the Multinomial Naive Bayes probabilities above). For each feature, the larger probability is bolded for clarity. Notice that most of the features below have significantly different conditional probabilities for classes S and T, indicating their strength.

| Feature | P($\phi$ \| S) | P($\phi$ \| T) | Indicates |
|---|---|---|---|
| Sentence contains 0 nouns | 0.0412 | **0.1408** | Sentence in original story text: probably unimportant. |
| Sentence contains at least one significant verb (as defined by verbs with English Language unigram probabilities of less than 0.0001) | **0.7871** | 0.6323 | Sentence in summary: sentence is important. |
| Sentence contains at least 1 proper noun | **0.46** | 0.3308 | Sentence in summary: sentence is important. |
| Sentence contains more than 2 verbs | **0.5865** | 0.3816 | Sentence in summary: sentence is important. |
| Sentence is question (contains a question mark) | 0.0098 | **0.07** | Sentence in original story text: probably unimportant. |

In summary, we found the Multinomial Naive Bayes classifier to be very useful in combining all the Sentence Significance Scoring Methods in the previous modules and giving a total probability score to feed into the CSP in the Summary Generation Stage. However, we also found that the Multinomial Naive Bayes classifier does not accept continuously or discretely ordered features and thus, we decided to extend upon this implementation with a variant of the classifier known as the Gaussian Naive Bayes classifier does.

## 4.7 Ensemble Model with Gaussian Naive Bayes

Each sentence significance model described so far interprets the importance of a certain set of sentence features well, but ignores the majority of sentence characteristics from which it could derive predictive power. To combine the insights provided by each model into a more well-rounded one, we consolidated all of the models previously discussed into a single ensemble model. Our intention was to use the output of the models discussed previously as features for a single classifier that could predict the probability that sentence would be found inside a summary, and by extension the significance of the sentence, better than any of the models could individually. To provide the backbone of this stacking methodology we chose to use a Gaussian Naive Bayes classifier as an extension of our Multinomial Naive Bayes classifier.

### 4.7.1 Mathematics of the Gaussian Naive Bayes Classifier

The Gaussian Naive Bayes classifier expands upon the principles underlying the design of the Multinomial Naive Bayes classifier, but overcomes one of its primary shortcomings. The standard Multinomial Naive Bayes classifier is incapable of handling continuous, ordered features. The Gaussian Naive Bayes classifier, however, is able to do so. The standard Gaussian Naive Bayes classifier assumes that continuous features conform to a Gaussian distribution. It derives its name from this characteristic. It estimates a specific conditional Gaussian distribution to which it believes each feature it encounters in training conforms to, when conditioned upon each class to which training examples can belong. It uses maximum likelihood estimation, described earlier in this paper, and the training examples with continuous feature values provided to determine the parameters of these Gaussian distributions. When estimating the probability that an example encountered in testing belongs to a certain class, the classifier cannot use the probability that a continuous feature takes on a certain value because the probability that a continuous feature takes on a specific value is infinitesimally small. To circumvent this issue, the Gaussian Naive Bayes classifier uses the PDF (of the conditional distribution that it has learned a continuous feature conforms to) output of a value as a proxy for the probability that the feature takes on that value. We demonstrate the mathematical basis for doing so below:

$$\mathcal{N}_{iY}(x) = Gaussian\ PDF\ of\ \phi_i \mid Y$$

$$P(\phi_i = x|Y) = \int_{\phi_i-\epsilon}^{\phi_i+\epsilon} \mathcal{N}_{iY}(x)\,dx = \varepsilon * \mathcal{N}_{iY}(x)$$

$$f(\phi_i|Y) = \mathcal{N}_{iY}(\phi_i)$$

$$P(S|\phi) \approx \frac{[P(S) * \prod_{i=1}^{n} P(\phi_i|S)]}{[P(T) * \prod_{i=1}^{n} P(\phi_i|T) + P(S) * \prod_{i=1}^{n} P(\phi_i|S)]}$$

$$= \frac{\left(\frac{c(S)}{N} * \prod_{i=1}^{n} \varepsilon * \mathcal{N}_{iS}(x)\right)}{\left(\frac{c(S)}{N} * \prod_{i=1}^{n} \varepsilon * \mathcal{N}_{iS}(x) + \frac{c(T)}{N} * \prod_{i=1}^{n} \varepsilon * \mathcal{N}_{iT}(x)\right)}$$

$$= \frac{(c(S) * \prod_{i=1}^{n} \varepsilon * \mathcal{N}_{iS}(x))}{(c(S) * \prod_{i=1}^{n} \varepsilon * \mathcal{N}_{iS}(x) + c(T) * \prod_{i=1}^{n} \varepsilon * \mathcal{N}_{iT}(x))}$$

### 4.7.2 Problems with Gaussian Naive Bayes

While Gaussian distributions may model many variables reasonably well, they by no means model every potential variable optimally. More importantly, they model the outputs of many of our other sentence significance scoring models extremely poorly. TextRank scores conform to a roughly exponential distribution. Most sentences receive TextRank scores of 0, with exponentially smaller fractions of sentences receiving increasing positive scores. In contrast, Noun and Adjective significance scores conform to a roughly Log-Normal distribution. Because we use Laplace smoothing in calculating noun and adjective probabilities, and remembering the way we calculated significance scores (refer to previous section), all scores must be greater than 0. Furthermore, the likelihood of increasingly larger scores increases until a certain critical point (the mode of significance scores), after which likelihoods of larger significance scores decreases. Most simply, story-specific verb significance scores are by design percentiles, values distributed uniformly between 0 and 1.

Furthermore, we decided to take advantage of the Gaussian Naive Bayes classifier's intelligent handling of ordered inputs to add ordered simple features. We provided several such features and feature templates. Sentence length, noun count, proper noun count, adjective count and verb count were among them. We modeled these five features using the Poisson distribution on our professor's recommendation. The intuition behind doing so is that word, noun, proper noun, adjective and verb appearances within a sentence are all essentially random events, and which the Poisson distribution was designed to model.

Lastly, providing discrete features to a Multinomial Naive Bayes classifier, as discussed previously, and feeding the output of this classifier into a Gaussian Naive Bayes classifier constitutes an unnecessary segmentation of prediction based on the features underlying the Multinomial Naive Bayes model, and would result in compounding of prediction errors. We consequently wanted a way to combine the discrete features handheld will by the Multinomial Naive Bayes classifier with the continuous features handled by the Gaussian Naive Bayes classifier.

### 4.7.3 Modified Gaussian Naive Bayes Classifier

Fortunately, we found a modification of the Gaussian Naive Bayes classifier that allowed it to model ordered features using arbitrary distributions, and could combine continuous and discrete features. We combined the logic behind the standard Multinomial and Gaussian Naive Bayes classifiers to create a hybrid, which we will refer to as the modified Gaussian Naive Bayes classifier. Our

mathematical logic for doing so follows, where $f_{iY}$ is the arbitrary PDF of $\phi_i$ | Y:

$$P_{discrete}(\phi_i|Y) = P_d(\phi_i|Y) = \frac{P(\phi_i, Y)}{P(Y)} = \frac{\left[\frac{(c(\phi_i, Y)+1)}{(|D|+|\phi|)}\right]}{\left[\frac{c(Y)}{|D|}\right]} = \frac{(c(\phi_i, Y)+1) * |D|}{(|D|+|\phi|) * c(Y)}$$

$$P_{contninuous}(\phi_i|Y) = P_c(\phi_i|Y) = \int_{\phi_i-\epsilon}^{\phi_i+\epsilon} f_{iS}(x)\,dx = \epsilon * f_{iS}(x)$$

$$P(S|\phi) = \frac{[P(S) * P(\phi|S)]}{[P(T) * P(\phi|T) + P(S) * P(\phi|S)]}$$

By the Naïve Bayes assumption:

$$\approx \frac{[P(S) * \prod_{i=1}^{n} P(\phi_i|S)]}{[P(T) * \prod_{i=1}^{n} P(\phi_i|T) + P(S) * \prod_{i=1}^{n} P(\phi_i|S)]}$$

$$= \frac{P(S) * \prod_{i=1}^{n} \begin{cases} P_d(\phi_i|S)\ if\ discrete \\ P_c(\phi_i|S)\ otherwise \end{cases}}{\left[P(S) * \prod_{i=1}^{n} \begin{cases} P_d(\phi_i|S)\ if\ discrete \\ P_c(\phi_i|S)\ otherwise \end{cases} + P(T) * \prod_{i=1}^{n} \begin{cases} P_d(\phi_i|T)\ if\ discrete \\ P_c(\phi_i|T)\ otherwise \end{cases}\right]}$$

$$= \frac{[P(S) * \prod_{i=1}^{d} P_d(\phi_i|S) * \prod_{i=d+1}^{n} P_c(\phi_i|S)]}{[P(S) * \prod_{i=1}^{d} P_d(\phi_i|S) * \prod_{i=d+1}^{n} P_c(\phi_i|S) + P(T) * \prod_{i=1}^{d} P_d(\phi_i|T) * \prod_{i=d+1}^{n} P_c(\phi_i|T)]}$$

$$= \frac{P(S) * [\prod_{i=1}^{d} P_d(\phi_i|S)] * [\prod_{i=d+1}^{n} \epsilon * f_{iS}(\phi_i)]}{P(S) * [\prod_{i=1}^{d} P_d(\phi_i|S)] * [\prod_{i=d+1}^{n} \epsilon * f_{iS}(\phi_i)] + P(T) * [\prod_{i=1}^{d} P_d(\phi_i|T)] * [\prod_{i=d+1}^{n} \epsilon * f_{iT}(\phi_i)]}$$

$$= \frac{P(S) * [\prod_{i=1}^{d} P_d(\phi_i|S)] * [\prod_{i=d+1}^{n} f_{iS}(\phi_i)]}{P(S) * [\prod_{i=1}^{d} P_d(\phi_i|S)] * [\prod_{i=d+1}^{n} f_{iS}(\phi_i)] + P(T) * [\prod_{i=1}^{d} P_d(\phi_i|T)] * [\prod_{i=d+1}^{n} f_{iT}(\phi_i)]}$$

$$= \frac{\left[\frac{C(S)}{|D|}\right] * \left[\prod_{i=1}^{d} \frac{\left(\frac{c(\phi_i, S)+1}{|D|+n}\right)}{\left(\frac{c(S)}{|D|}\right)}\right] * [\prod_{i=d+1}^{n} f_{iS}(\phi_i)]}{\left[\frac{C(S)}{|D|}\right] * \left[\prod_{i=1}^{d} \frac{\left(\frac{c(\phi_i, S)+1}{|D|+n}\right)}{\left(\frac{c(S)}{|D|}\right)}\right] * [\prod_{i=d+1}^{n} f_{iS}(\phi_i)] + \left[\frac{C(T)}{|D|}\right] * \left[\prod_{i=1}^{d} \frac{\left(\frac{c(\phi_i, T)+1}{|D|+n}\right)}{\left(\frac{C(T)}{|D|}\right)}\right] * [\prod_{i=d+1}^{n} f_{iT}(\phi_i)]}$$

$$= \frac{c(S)^{1-d}[\prod_{i=1}^{d} c(\phi_i, S)+1] * [\prod_{i=d+1}^{n} f_{iS}(\phi_i)]}{c(S)^{1-d}[\prod_{i=1}^{d} c(\phi_i, S)+1] * [\prod_{i=d+1}^{n} f_{iS}(\phi_i)] + c(T)^{1-d}[\prod_{i=1}^{d} c(\phi_i, T)+1] * [\prod_{i=d+1}^{n} f_{iT}(\phi_i)]}$$

Implementing the modified Gaussian Naive Bayes classifier thus allowed us to simultaneously use discrete features and continuous features modeled using arbitrary distributions. For each continuous feature, our featurizer supplies a distribution type. The modified Gaussian Naive Bayes classifier then learns each distribution's parameters in training using MLE.

# 5 Summary Generation

To select the combination of sentences from within a story that form the optimal summary, we implemented two different algorithms. First, we implemented a simple greedy algorithm. Subsequently, we implemented a more complicated constraint satisfaction algorithm, that while more appropriate in theory performed poorly in practice.

## 5.1 Greedy Sentence Selection

First, we implemented a greedy sentence selection algorithm. This algorithm functions exactly as one would expect, iteratively selecting the sentence that would increase the summary's expected score by the highest amount and adding that sentence to the growing summary until the summary's word limit is met. The expected addition a sentence will provide to a summary's value is calculated by taking the product of the following three factors:

1. Sentence significance score: This score is calculated by any of the previously discussed sentence significance scoring models. This is the most important factor.
2. Redundancy penalty: The is the reciprocal of the sum of the TF-IDF angular similarities between the sentence and each sentence already selected for placement in the summary. This factor penalizes sentences that are similar to those already in the summary, encouraging a broad range of topic coverage in the summary.
3. Length penalty: This is the reciprocal of the length of the sentence in question, and penalizes longer sentences that eat up larger portions of the summary's word limit.

We found that this simple greedy algorithm, using the sentence significance scores provided by previously discussed models as well as a few additional heuristics, performed quite well.

## 5.2 Constraint Satisfaction Problem

We also tried modeling the task of sentence extraction as constraint satisfaction problem (CSP). The CSP balances the value of selecting important sentences, the cost of choosing similar sentences that contain overlapping content, and the word limit constraint imposed on the task of summary generation.

### 5.2.1 CSP Construction

The first step of the construction of this CSP is to create a variable for each sentence in our summary, represented as the index in the summary. For each variable, we create a unary potential that is a function of the overall sentence significance scoring value we obtained corresponding to this specific sentence.

For every variable, we also create a binary potential between the variable and all the other variables in the summary that serves as a heuristic penalty for similarity. The value of this penalty term is equal to reciprocal of angles between the TF-IDF vectors.

### 5.2.2 Solving the CSP

After defining the potentials necessary to solving this CSP, we now need to find the optimal algorithm to solve it. At first, we attempted to solve this CSP using a Backtracking Search Method, leveraging standard backtracking search. Although this method was very effective for solving CSPs with shorter stories and summaries, we found that as the length of our stories and desired summary lengths increased, the Backtracking Search Algorithm ran exponentially slower with an execution time ranging from minutes to hours.

To speed up the process of solving our CSP, we decided to implement a Beam Search algorithm, with beam size K. Our algorithm utilizes the same underlying structure of this example. In the proceeding section, we will describe how our Beam Search algorithm works through the context of this structure. Before doing so we will define the symbols used in the above pseudo code as they relate to our problem:

- The indexes i represent the index into the summary that we are constructing (variable name = $X_i$)
- As mentioned in the preceding section, the values that we assign to our variables are the indexes into the story sentences.
- The variable C represents the current set of partial assignments that we have for our variables
- The variable v represents the next value that we are assigning when we extend our partial assignment
- $Domain_i$ is the set of possible story indexes that we can still assign to our summary
- K = the beam size of our Beam Search algorithm.

Our Beam Search Algorithm is as follows:

1. Each of our K partial assignments represents an assignment of sentences in our story to our variables (sentence index of our summary). In this step, we extend each of these K partial variable assignments by assigning the next variable $X_i$ the value v, which is the sentence we want to

assign to this variable. Note that we also extend the weights of this new partial assignment by obtaining and multiplying the current weight by the aforementioned unary potential and binary potentials relating to the assignment of $X_i$ to the value v.

2. Now that we have all of our possible new partial assignments along with their corresponding weights, we take the best K elements of each partial assignment based on these weights.

3. After obtaining our K best elements, we now prune out all the other assignment extensions and move onto the next iteration of assignment extensions with our K best partial assignments.

We continue repeating steps 1-3 until we have a completed assignment for all our variables that should represent the best possible assignments for our final variables. Unfortunately, we eventually found that beam search did not perform optimally. Stories can potentially contain hundreds of sentences, and limiting the possible set of assignments on each iteration to a reasonable beam size of less than 20 prevents the vast majority of potential summaries from being considered. Upon observing that backtracking search ran too slowly to be practical, and beam search often produced sub-optimal summaries, we decided to stick to the use of greedy sentence extraction algorithms but still included the option to use Beam Search for those who want to do so.

# 6 Evaluation

## 6.1 Evaluation Metric

For the evaluation of our generated summaries, we use the Recall-Oriented Understudy for Gisting Evaluation (ROUGE) metric. ROUGE is a publicly available text summary evaluation library and is currently the industry standard in text summary evaluation. ROUGE evaluates a summary by comparing it to a human-generated gold summary and measuring similarity. It does so similarly to the way in which BLEU evaluates translations by comparing them to gold translations. While BLEU looks only at precision however, ROUGE also takes recall into account in generating a final score.

## 6.2 Getting Baseline and Oracle

To establish a lower bound of expected performance, we implemented an extremely naive baseline algorithm. Our baseline algorithm iteratively extracts the first sentence of every paragraph in a story and concatenates these to form a summary. While this algorithm is certainly extremely naïve, it should nonetheless perform significantly better than a summarizer that randomly selects sentences. Sentences that start paragraphs tend to be relatively important.

To establish an upper bound of expected performance, or an oracle, we compared pairs of human-generated summaries of the same story to one another, treating one as the gold summary. By doing so, we basically modeled humans as our oracle, the best oracle possible.

We evaluated our baseline and oracle summaries on the subset of stories we had for which we could find at least two summaries online. We obtained the following results:

Average Baseline ROUGE Score = 0.019
Average Oracle ROUGE Score = 0.185

This experiment, and more specifically the fact that the mean oracle ROUGE score was below 0.2, demonstrated that even human-generated summaries deviate from one another significantly. However, it also demonstrated that naive baseline methods perform abysmally. The ROUGE scores of the Oracle summaries, although not amazing, still exceeds the baseline score by about a factor of 10. This demonstrates significant room for algorithmic improvement.

# 7 Results and Analysis

## 7.1 Comparison of Different Sentence Significance Scoring Models

We ran a series of tests using our various sentence significance scoring metrics as inputs to a greedy sentence selection algorithm. As previously mentioned, we discarded the model that used SVR with estimated training data and the use of CSP solving methods for summary sentence selection due to their immediately obviously weak performance. Each test was run with a training set consisting of 50 stories and associated summaries and a test set consisting of 50 stories and associated summaries. Due to the large runtimes of the algorithms used, the algorithms would have taken exceedingly long to train and test on larger data sets. The results of our tests appear in the table below:

| Significance Scoring Model | Mean ROUGE Score |
|---|---|
| TextRank | 0.041813 |
| Verb Scoring | 0.023519 |
| Noun and Adjective Scoring | 0.018195 |
| K-Means Clustering | 0.020723 |
| K-Means Clustering with Angular Loss | 0.020723 |
| Multinomial Naïve Bayes with Simple Features | 0.034535 |
| Gaussian Naïve Bayes Ensemble Model with all Features | **0.044451** |

Note that the ensemble model performed better than the others by a significant margin, with a mean ROUGE score of 0.044. As such, the ensemble model significantly outperformed our naïve baseline model. It nonetheless significantly underperformed our oracle, but this came as no surprise given the difficulty of the task of automatic summary generation.

In addition to the ensemble model, TextRank model also performed reasonably well, as did the Multinomial Naïve Bayes model with simple features. The noun and adjective scoring model, however, performed quite poorly, underperforming our naïve baseline model. The large variations in the performances of these different models raised the question of whether or not each model provided value to the ensemble model. To investigate this question, we ran further tests.

### 7.2 Comparison of Different Feature Combinations with the Ensemble Model

We ran a series of tests through which we experimented with providing different features to the stacked ensemble model backed by the modified Gaussian Naïve Bayes classifier. Throughout all of these tests, we maintained the use of simple binary sentence features and adjusted only the rich sentence scoring model features used. We tested all possible combinations rich features. As with our previous tests, we ran each test with a training set consisting of 50 stories and associated summaries and a test set consisting of 50 stories and associated summaries. The results obtained from this experiment appear below.

| TextRank Feature | Verb Score Feature | Noun & Adjective Score Feature | Clustering Feature | Average ROUGE Score |
|---|---|---|---|---|
| yes | yes | yes | yes | **0.041813** |
| yes | yes | yes | no | 0.033306 |
| yes | yes | no | yes | 0.037782 |
| yes | no | yes | yes | 0.040661 |
| no | yes | yes | yes | 0.036244 |
| yes | yes | no | no | 0.016785 |
| yes | no | yes | no | 0.029695 |
| yes | no | no | yes | 0.040661 |
| no | yes | yes | no | 0.031731 |
| no | no | yes | yes | 0.027687 |
| no | yes | no | yes | 0.036244 |
| yes | no | no | no | 0.036244 |
| no | yes | no | no | 0.030276 |
| no | no | yes | no | 0.033361 |
| no | no | no | yes | 0.039688 |
| no | no | no | no | 0.033361 |

Note that the highest mean ROUGE score was obtained through the use of all four rich features. This indicates that they all provide predictive value. After running this second series of tests, we remained convinced that all four of the sentence significance scoring models tested possess predictive power.

### 7.3 Error Analysis

To examine the performance of our system in a more qualitative manner, we will examine a sample summary it produced. We will look at the summary of "In Another Country" by Ernest Hemingway produced by the Gaussian Naïve Bayes ensemble model using all rich features:

> The major held the photograph with his good hand and looked at it very carefully. he asked. as we passed. But this was a long time ago, and then we did not any of us know how it was going to be afterward. We only knew then that there was always the war, but that we were not going to it any more. We were all a little detached, and there was nothing that held us together except that we met every afternoon at the hospital. The boys at first were very polite about my medals and asked me what I had done to get them. The major came very regularly to the hospital. The machines were new then and it was we who were to prove them. he asked me. He seemed very angry. He spoke very angrily and bitterly, and looked straight ahead while he talked. He was looking at the wall. When he came back into the room, I was sitting in another machine. My wife has just died. The doctor told me that the major's wife, who was very young and whom he had not married until he was definitely invalided out of the war, had died of pneumonia. The major did not come to the hospital for three days. In front of the machine the major used were three photographs of hands like his that were completely restored. I do not know where the doctor got them. I always understood we were the first to use the machines. The photographs did not make much difference to the major because he only looked out of the window.

While far from perfect, this summary does succeed in mentioning most of the story's important topics. It captures the central premise of the story, the death of the Major's wife, and also picks out several mentions of the photographs that form an important motif during the later parts of the story. However, the summary does not read the way a piece written by a human would. For starters, it contains several grammatically incorrect sentences. More importantly, it fails to transition from one idea to the next in a smooth manner. Sentences describing different events and ideas follow one another in abrupt fashion, often sounding like non-sequiturs. Another significant issue with this summary is that many of the pronouns it contains refer to unspecified people. In the original story, these pronouns refer to their subjects without ambiguity given the context in which they appear. However, because only a very small percentage of the story's content has been extracted to form the summary, most of that context has dissipated.

## 8 Conclusions

The performance of our algorithms exceeded our expectations. We did not expect to make any significant headway with an end goal that seemed so daunting, and were pleasantly surprised when we obtained reasonable results. While we were constrained by the inherent inaccuracy of extractive summaries and difficulty in getting an accurate summarization metric, we were able to test out a wide variety of metrics to try and represent sentence importance and laid the groundwork to continue optimizing our summary generator for human subjects.

## 9 Future Work

There is much we can do to improve the project and our results. Short term goals include applying coreference resolution to the text before summarizing it and adding generative sentences to our extractive summaries, such as a list of characters in the story. In the medium term, we would like to try replacing the Gaussian Naïve Bayes classifier with other classifiers and comparing their performances to that of our existing system. Using a Logistic Regression-based classier would be a natural next step because it is also a probabilistic classifier, but we could certainly experiment with discriminative classifiers as well. When using discriminative classifiers, instead of using the classifier's estimated probability that a sentence belongs in a summary as a proxy for sentence significance, we would simply use the classifier's confidence of

that same fact instead. The transition should be smooth. More ambitious and yet untested is the idea to model the sentence extraction process as a Hidden Markov Model (HMM) in which sentence significance probabilities would represent emission probabilities, and the probability a sentence should be extracted given the sentences already placed in the summary would represent transition probabilities.

# 10 References


[1] Automatic Summarization, by Ani Nenkova and Kathleen McKeown: http://www.cis.upenn.edu/~nenkova/1500000015-Nenkova.pdf

[2] TextRank: Bringing Order into Texts: http://digital.library.unt.edu/ark:/67531/metadc30962/m2/1/high_res_d/Mihalcea-2004-TextRank-Bringing_Order_into_Texts.pdf

[3] LexRank, by Gunes Erkan: http://www.cs.cmu.edu/afs/cs/project/jair/pub/volume22/erkan04a-html/erkan04a.html

[4] Summarization by Latent Dirichlet Allocation, by Kenton Murray: http://kentonmurray.com/thesis/thesis.pdf

[5] TextRank: Braining Order Into Text: http://web.eecs.umich.edu/~mihalcea/papers/mihalcea.emnlp04.pdf


# 11 Appendix

## 11.1 Running Our System

To run our system, execute "system.py". In its current state, it will run tests using the seven different sentence scoring models whose testing results appear in section 7.1 and report performance statistics for each. Note that because the stories placed into the test and training sets are randomly selected from a larger data set, results will vary each time the system is run.

## 11.2 Files Related to Functionality Described

Pipeline:
1. system.py
2. DocumentSummarizer.py

Preprocessing:
1. DocumentPreprocessor.py

TextRank:
1. TextRank.py

Noun and Adjective Significance Scoring:
1. WordSignificancePredictor.py

Verb Significance Scoring:
1. VerbSignificanceRegressor.py

Sentence Clustering:
1. TFIDFCalculator.py
2. Clusterer.py
3. SentenceClusterer.py

SVR with Estimated Training Data:
1. TFIDFCalculator.py
2. TFIDFSentenceSignificanceCalculator.py
3. SentenceFeaturizer.py
4. TFIDFSentenceSignificanceRegressor.py

Multinomial Naive Bayes with Simple Features:
1. SentenceFeaturizer.py
2. SentenceClassifier.py

Ensemble Model with Gaussian Naive Bayes:
1. SentenceFeaturizer.py
2. util.distributions.py
3. SentenceClassifier.py

CSP Construction and Solving:
1. SentenceWeightsCSPConstructor.py

Evaluation:
1. SummaryEvaluator.py

## 11.3 Library Dependencies

1. nltk (the library itself as well as its corpora)
2. pyStatParser
3. pattern
4. scikit-learn
5. numpy
6. pygraph
7. jsonrpc
8. corenlp-python
9. XML::DOM

## 11.4 Borrowed Code

In addition to libraries mentioned above, we imported a substantial amount of code into our project written by others. The following files and directories contain code we did not write:

1. code.stanford-ner directory (open source code
2. CSP.py and BacktrackingSearch.py (from the CS 221 Assignment 6: Scheduling)
3. rouge directory (open source code)
4. Additionally, a few lines of code here and there within the code base were also taken from external sources, and are marked as such by comments.

## 11.5 Submission

1. Ashkon Farhangi is submitting this project for CS 221 and CS 224N
2. Lucio Tan is submitting this project for CS 221
3. Eric Holmdahl is submitting this project for CS 221 and CS 229