Multi-document extraction based Summarization*

Sandeep Sripada ssandeep@cs.stanford.edu
Venu Gopal Kasturi venuk@cs.stanford.edu
Gautam Kumar Parai gkparai@cs.stanford.edu

ABSTRACT

In this paper, we present three techniques for generating extraction based summaries including a novel graph based formulation to improve on the former methods. The first method uses a sentence importance score calculator based on various semantic features and a semantic similarity score to select sentences that would be most representative of the document. It uses stack-decoder algorithm as used as a template and builds on it to produce summaries that are closer to optimal. The second approach clusters sentences based on the above semantic similarity score and picks a representative from each cluster to be included in the generated summary. The third approach is a novel graph problem based formulation where summaries are generated based on the cliques found in the constructed graph. The graph is generated by building edges between sentences which talk about similar topics but are semantically not similar.

These approaches are used to generate a 100-word summaries for the dataset available as part of DUC 2004. The paper discusses the system developed, algorithms used and their analysis along with improvements. ROUGE scores obtained are comparable to other participants in the competition.

General Terms

Multi-document Summarization, Maximal cliques, Semantic similarity, Stack decoder, Clustering

1. INTRODUCTION

With the recent increase in the amount of content available online, fast and effective automatic summarization has become more important. The need for getting maximum information by spending minimum time has led to more efforts being directed to the field of summarization.

Most current automatic summarization systems use sentence extraction, where key sentences in the input documents are selected to form the summary. Sentence scoring methods utilize both purely statistical and purely semantic features, for example as in [12], [17], [19]. Systems that go beyond sentence extraction, reformulating or simplifying the text of the original articles, must decide which sentences should be simplified, compressed, fused together or rewritten [1], [6], [7], [8], [16]. Common approaches for identifying important sentences to include in the summary include training a binary classifier [9], training a Markov model [3], or directly assigning weights to sentences based on a variety of features and heuristically determined feature weights [11], [15].

In this paper, we employ three different techniques to generate automatic summaries from a set of related topic documents. The first approach is based on picking the sentences with maximum importance score to form the summary. Instead of greedily picking the sentences based on maximum scores, we build on the stack decoder algorithm proposed by [14] to pick a solution that is close to optimal based on our constraints. The solution, though not optimal, is 'close' to optimal because of the way we limit the stack sizes in the decoder algorithm to avoid exponential blowup. In the second method, we cluster the documents based on the semantic similarity score using K-means clustering algorithm. The obtained clusters are then processed to get the clusteroid which is then placed in the summary.

The third approach is a novel graph based formulation where the task of generating summaries is converted to the problem of finding cliques in the sentence distance graph. A clique in a graph is a strongly connected component formed by a subset of its vertices. The cliques are processed to obtain a summary from the vertices and the best summary based on the importance scores is selected as the final output.

The performance of these methods is closely attached to the performance of the semantic similarity and sentence importance scorers. We tried various similarity measures to look at how duplicates can be eliminated.

- Bag of words model:
  - Jaccard similarity
  - Cosine similarity
  - Tf-Idf distance
- Semantic similarity:

*CS 224N: Final Project

- Jiang and Conrath
- Lesk
- Resnik
- Lin

We used a linear combination of the feature values as our importance function. The weights for the feature values were calculated by training on the DUC 2002 data and then using them to get the importance score of sentences in the test data. The features used and analysis of the weights is mentioned in Section 3.2. We also tried various normalization schemes and our error analysis showed that were effective and improved the performance.

The paper is organized as follows: Section 2 explains the dataset, Section 3 explains the various features used and methods for importance, similarity scores, Section 4 describes the algorithms used in detail, Section 5 describes the analysis and improvements carried out, Section 6 shows the results obtained and finally Section 7 talks about the future work that could help improve the methods described in this paper.

2. DATA DESCRIPTION

We conducted the experiments on the topics taken from DUC 2004. In the multi-document summarization task in DUC 2004, participants are given 50 document clusters, where each cluster has 10 news articles discussing the same topic, and are asked to generate summaries of at most 100 words for each cluster.

In DUC 2002, 60 sets of approximately 10 documents each were provided as system input for the single document summarization task. A generic abstract of the document with a length of approximately 100 words or less was created using these input documents. These documents were used for training purposes.

3. SCORES AND FEATURES

The section describes the various measures, features considered.

3.1 Similarity Scores

As mentioned earlier, we experimented with the following similarity measures (bag of words model):

- Jaccard Similarity: The similarity between two sentences is defined the amount of word overlap normalized by the union of the sets of words present in the two sentences. It is calculated using the formula:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \quad (1)$$

- Cosine Similarity: It gives the cosine of the angle between the vectors represented by the word-frequency vectors of the two sentences. It is calculated using:

$$sim = \frac{A \cdot B}{||A||||B||} \quad (2)$$

- TF-IDF similarity: This is also a vector based model but the words are weighted by their TF-IDF score.

$$tf.idf(t,d) = (1 + log(tf_{t,d})) * log(\frac{N}{df_t}) \quad (3)$$

3.1.1 WordNet-based semantic similarity

(Using Lesk Word Sense Disambiguation)
We have used the Lesk-based semantic similarity[1] for computing the similarity between sentences. The Lesk algorithm [10] uses gloss to disambiguate a polysemous word in a sentence context by counting the number of words that are shared between two glosses. The more overlapping the words, the more related are the senses. Using this as the basis, the semantic similarity between two sentences is computed as follows.

1. Tokenization of the sentence(alongwith stop words removal).
2. Part-of-speech tagging.
3. Stemming words.
4. Find appropriate word sense for each word in sentence(Lesk Word Sense Disambiguation).
5. Sentence similarity based on word pair similarity.

Once the most appropriate sense has been found for each word using the Lesk Algorithm, the semantic similarity between synsets is determined using path length similarity using is-a(hypernym/hyponym) hierarchies in WordNet. The notion being that the shorter the path length, the higher is the semantic similarity between synsets. The path length similarity between synsets is computed using the following.

$$Sim(synset_1, synset_2) = 1/Distance(synset_1, synset_2)$$

Given two sentences, the semantic similarity between appropriate word senses for word pairs at different positions is precomputed in a semantic similarity relative matrix $[X_m, Y_n]$ where m,n are the respective lengths of sentence X and Y. The semantic similarity between sentences is computed as the maximum total matching weight of a bipartite graph, as disjoint sets of nodes, where the first set would have words from the first sentence and the second set would have words from the second sentence. This bipartite graph is used by the Hungarian algorithm to compute the best assignment corresponding to the global minimum. The results from matching pairs are combined into a single similarity value for two sentences. To compute the overall combined similarity value for sets of matching pairs the following scores can be used.

1. Matching Average $\frac{2*Match(X,Y)}{|X|+|Y|}$ where Match(X, Y) are the matching word tokens between X and Y.
2. Dice Coefficient $\frac{2*|X \cap Y|}{|X|+|Y|}$ returns the ratio of the number of matched word tokens to the total number of word tokens.

[1] http://opensource.ebswift.com/WordNet.Net/Default.aspx

This is the final semantic similarity score computed for the given sentence pair $[X, Y]$.

For example the wordnet based similarity between the sentences:

"Adolf Hitler was a very violent ruler and as a result Germany suffered during his reign."

and

"Hitler ruled Germany with what was its worst dictatorship and this severely damaged the economy of the country."

is around 0.75 whereas the normal cosine similarity is only 0.22. This shows that this method is highly effective when the words used are different but have a similar mode in the synsets.

We used WordNet[2] to obtain semantic information about the various words comprising a sentence and used this information to obtain a semantic score. The methods mainly consider synset expansion of the words using WordNet. The similarity of the sentence is given by:

$$sim(s_1, s_2) = \frac{1}{2} * (\sum_{w_i \in s_1} \arg\max_{w \in s_2} maxSemScore(w_i, w) + \sum_{w_j \in s_2} \arg\max_{w \in s_1} maxSemScore(w_j, w))$$

We used the semantic measures mentioned earlier like Jiang and Conrath, Resnik, Lin present in he Java WordNet similarity library[3]. After a few initial experiments, we decided to drop these measures as they were not normalized and because they were more useful in cases where comparison between pairs of sentences is required.

3.2 Sentence Importance Scores

Each sentence is given an importance score and this acts as a goodness measure for the sentence. The scores can be used to order sentences and pick most important sentences. The probability of a sentence to be present in the summary is proportional to its score. Each sentence is represented by a set of features and the score is a function of the weighted sum of the individual feature values.

The features we have used are:

- TF-IDF sum: The goodness of a sentence is usually represented by the importance of the words present in it. TF-IDF is a simple but powerful heuristic for ranking the words according to their importance. This feature is the sum of the TF-IDF scores of the individual words of the sentence.
- Sentence length: This feature is the number of words present in the sentence. Longer sentences usually contain more information about the documents.

[2]http://wordnet.princeton.edu/

[3]http://www.cogs.susx.ac.uk/users/drh21/

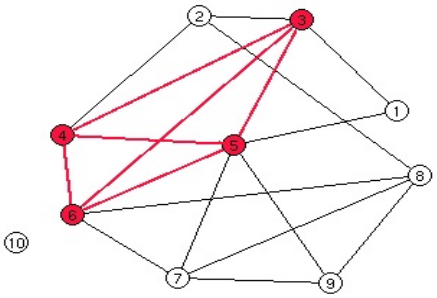

Figure 1: A clique in a graph of 10 nodes

- Named Entities count: Sentences which contain named entities are usually more important as these sentences indicate information of the entities participating in the documents. This feature is a count of the named entities present in the sentence as indicated by the Stanford NER library[4].
- Top-K Important words: The TF-IDF sum feature might lead to the selection of long sentences with many insignificant words. To prevent this, we introduced the Top K Important words feature. It counts the number of words of this sentence present that are present in the top K words ranked by their TF-IDF scores.
- Sentence Position: News articles tend to contain most of the important sentences in the first paragraph itself [2]. Articles which are opinions by individuals tend to contain the summaries at the end of the document in a concluding paragraph. Hence sentence position tends to be a good indicator of the importance of a sentence across different classes of documents.
- Numerical Literals count: Sentences which contain numerical literals usually indicate attributes of the events like death toll, time of occurrence, statistical information etc. This feature counts the number of Numerical terms present in the sentence.
- Upper case letters count: Words which contain upper case letters are usually entities and hence we are using this feature to count the number of such instances.
- Nouns count: Represents the number of noun classes in the sentence.
- Verbs count: The count of the number of verbs and its various forms in the sentence.
- Adjectives count: The count of the number of adjectives in the sentence.

The feature vectors were generated for training data and then logistic classification was employed to get the weights for features that get best accuracy. These weights were then used to get the importance for sentences using the features for the test data.

[4]http://nlp.stanford.edu/ner/index.shtml

3.3 Normalizations

We also tried experimenting with various normalization schemes. We started by generating the feature values and normalizing some of them based on the sentence length and others based on the number of sentences in the document. The features considered for length normalization were: (sentence length was not normalized in this mode)

- TF-IDF sum, Named Entities count, Top-K Important words, Numerical Literals count, Upper case letters count, POS tags counts: These feature were divided by the length of the sentence being considered to get a normalized value.
- Sentence Position: This feature is normalized by the number of sentences in the document. This gives us the % position of the sentence in the document.

These normalizations did not perform better than the unnormalized feature values because of the way the information is lost when dividing by the sentence length. Longer sentences which contain important information are length normalized and hence lose information. Also, short sentences get unnecessary bumps in the scores because of the length normalization.

We analyzed various other normalization schemes and found the one described in [18] to be quite effective. The feature values are normalized based on a sigmoid function which effectively looks at how many standard deviations away the value is from the mean. In this method, the features are normalized as follows:

- TF-IDF sum, Named entities count, Top-K important words, Numerical literals count, Upper case letters counts, POS tags count, Sentence length: The feature value is normalized using the sigmoid function:

$$T = \frac{1 - e^{-\alpha}}{1 + e^{\alpha}}, where\ \alpha = \frac{t(s) - \mu}{\sigma} \quad (4)$$

- Sentence Position: This feature is normalized by the number of sentences in the document. This gives us the % position of the sentence in the document.

This normalization proved to be highly effective when compared to the no normalization and length normalization cases.

4. ALGORITHMS

In this section we describe the three techniques used to generate extraction based multi document summaries.

4.1 Stack decoder based formulation

Stack decoder formulation was used to generate summaries 'close' to global optimal as the algorithm can test multiple summary lengths. Potential summaries are given an overall score based on the scores of the included content units, and the goal is to find the summary with the best overall importance score. The importance score of a summary is equal to the sum of importance scores of individual sentences.

$imp(summary) = \sum_{s \in sentences} imp(sen)$

The summary generation techniques takes as input the set of all sentences from the input documents along with their importance scores. The decoder has $maxlength + 1$ stacks, one for each length, up to $maxlength$ and an additional stack for all summaries with length greater than the summary length wanted which is $maxlength$. Each stack would contain the best summary for that length. As we are using a priority queue in each stack to maintain the solutions and since the length of the priority queue is limited, we can say that the solution obtained is optimal for that length i.e. there will be at most $stacksize$ different hypotheses on any given stack. New solutions would be added to the stack only if there is space or if the score of the

The algorithm proceeds by examining a particular stack. It looks at every solution on that stack (a solution is a set of sentences) and then tries to extend that solution with every sentence from the input document sentences. These extensions are then placed on the stack of the appropriate length. The exponential blowup of solutions at each stack is avoided by maintaining just $stacksize$ solutions at each length.

Some important aspects of the implementation are:

- The sentences below a *min sentence length* are not considered.
- The sentence being considered to add to a solution (extending phase) is added to the existing solution only if the similarity (a measure that helps estimate the overlap) is below a certain value. This helps in tackling the issue of redundancy in the summary.
- The number of solutions maintained in the priority queue is restrained, as described above, to counter exponential blowup.

Algorithm: Initially, all the sentences are placed at their respective length positions along with the (score, id) pair in a priority queue, that has stacksize number of solutions.

```
for i = 0 to maxlength do
  for all sol in Stack[i] do
    for all s in Sentences do
     newlen = maxlength+1
     if i+length(s)<=maxlength
      newlen = i+length(s)
      if similarity < threshold
         newsol = sol U {s}
      else
         next
      score = importance(newsol)
      Insert newsol, score into priority queue stack[newlen]
    end for
  end for
end for
Return best solution in stack[maxlength]
```

(Best solution here is the global minimum of the KLD value among all the solutions.)

4.2 Clustering based generation

K-means is an unsupervised learning algorithm that solves the well known clustering problem. The procedure classifies a given data set through a certain number of clusters (assume k clusters) fixed a priori. The main idea is to define k clusteroids, one for each cluster. These clusteroids are chosen to place them as much as possible far away from each other. The next step is to take each point belonging to a given data set and associate it to the nearest clusteroid. When all points have been classified, we re-calculate k new clusteroids as new centers of the clusters resulting from the previous step. After we have these k new clusteroids, a new association is generated between the same data set points and the nearest new clusteroid. The k clusteroids change their location in each step until no more changes occur. Although the K-means algorithm will always terminate, it does not necessarily find the most optimal configuration, corresponding to the global objective function minimum. The algorithm is also significantly sensitive to the initial randomly selected cluster centers. The K-means algorithm can be run multiple times to reduce this effect.

We used the K-Means clustering algorithm to cluster together similar sentences from multiple documents. We initially, choose k sentences as the clusters which are at maximum distance from each other. We used different non-euclidean distance metrics e.g. jaccard similarity, cosine similarity, wordnet based similarity (lesk disambiguation). At the end of each iteration, a new clusteroid is chosen that minimizes the average distance from the clusteroid. Once the algorithm has terminated, we choose the clusteroid of each cluster as the potential sentences for the summary. We then rank those sentences by their importance score which is a linear combination of the sentence features viz tf-idf score, sentence length, named entity recognition count, sentence position, parts-of-speech tag counts etc. (The score computation is described in detail in Section 3.2) We start by choosing sentences according to their rank (best first), until we reach the required summary length of 100 words. We truncate the last sentence so that the summary fits within the 100 word boundary. The sentences can be reordered, to produce coherent summaries. However, this does not make a difference with ROUGE scores, hence we did not experiment with sentence reordering.

4.3 Graph based formulation

In [4], an intra-sentence cosine similarity graph was constructed and an eigen vector centrality of a graph was used to compute the ranking of sentences. In [13], the authors used a SumGraph and the ranking of the sentences was obtained by using the Pathfinder algorithm. The mentioned systems use techniques to compute the relative ranking of nodes (sentences) by using the properties of the nodes in the graph like degree of the nodes, page rank etc.

In our approach, we have formulated the problem of extractive text summarization as a 'clique' (Figure 1) finding problem. We build a sentence dis-similarity graph on sentences picked from documents to summarize using one of the above mentioned similarity scores to get the similarity values. With sentences as nodes we build edges if the similarity between the two sentences (nodes) is less than a threshold. As the edges in the graph are present when the sentences are dis-similar, we need to find a subset of nodes where every pair is connected to each other (these connection imply low redundancy in the content). This corresponds to a clique in the graph and hence every clique is a possible candidate for a summary. We now run the clique finder to find all maximal cliques in the constructed graph using java graph library[5].

Table 1: Graph statistics - clique finding problem

Graph	Components	Number
Original	Vertices	250
	Edges	15000
Reduced	Vertices	60
	Edges	1400

The formulation for graph G = (V, E) is as follows:

$V = \{S | S \in \{sentences\ to\ summarize\}\}$
$E = \{(U \rightarrow V) | sim(U, V) < threshold\}$

Once a clique is obtained, the nodes are ordered based on the importance scores and a summary is generated by adding these sentences in that order until the constraints are violated.

As the problem of finding cliques is NP-hard, getting all the biggest maximal cliques is not easy. It is computationally expensive and when we considered the whole set of sentences with similarity < 0.5, the number of edges formed were around 15000 for a 250 sentence document as shown in Table 4.3. The worst-case time complexity for finding the biggest maximal clique is $O(3^{n/3})$ for an n-vertex graph. This made it impossible to work on such a dense graph. We therefore used a subset of the sentences and performed the decoding on that set. Also, to prevent exponential blowup in the execution time (as the problem is NP-Hard) while finding cliques we have reduced the density of the graph by only considering top 60 sentences from the documents after ordering them based on their importance scores.

5. ANALYSIS & IMPROVEMENTS

5.1 Similarity scores

This module was one of the most important base components and we spent considerable amount of time to check out various measures and to see the fault points. After looking at various sentences and their similarities, we picked simple cosine similarity, Tf-Idf distance and wordnet based similarity as the final candidate measures. The performance of cosine and tf-idf distance measure was similar and these measures were fast. However the performance of word net based similarity was not as high as expected mostly because of the use of similar language among various sentences in DUC data. Also, the problem we have observed is that these measures are not normalized. Hence we were not able to stumble upon a threshold, above which we can flag two sentences as being similar. And we have realized that these methods are better at ranking a set of sentences according to their relative similarity with a given sentence rather than solely compar-

[5]JGraphT - http://jgrapht.sourceforge.net/

ing two sentences. This measure is also very slow and time consuming.

Wordnet based similarity measure performed well at times but in most of the occassions it assigned high similarity scores to not so sematically similar sentences. This coupled with the fact that the similarity measure is slow pushed us to take up simple cosine similarity as the default measure. Stemming and stop word removal analysis was also done and since cosine similarity looks for exact matches, we decided to take up stemming. As tf-idf was one of our measures, this assigns appropriate weights to both common words and frequent but important words. As a result, sentences with many stop words would automatically get a low score when compared to the sentence with important words. Also, the presence of other features like NER and top-k also makes sure that long sentences with common and unimportant words do not dominate small and sentences with important words.

For example, the similarity between the sentences:

"Christian conservatives argue that hate crime laws restrict freedom of speech, while gay rights activists and others say these laws send a message that attacks on minorities will not be tolerated."

and

"But when he stopped, he found the burned, battered and nearly lifeless body of Matthew Shepard, an openly gay college student who had been tied to the fence 18 hours earlier."

is 0.69 where as the cosine similarity is 0.08 (without stemming) and 0.28 (with stemming).

5.2 Features

As all the input documents are related to a particular event, most of them talk about the important entities present in the topic. Hence the TF-IDF score of the important words would tend to zero. To circumvent this problem, we considered the logarithm of the term frequency alone if the word is present in all the documents.

We used a sigmoid function over the hypothesis value of the sentence to compute its importance. But this lead to cases where two smaller sentences together had more importance value than a longer and more important sentence. This is because, a sigmoid function maps the hypothesis over $[0, 1]$ and hence the differences in the hypothesis of pairs of sentences becomes very less. To avoid this, we are using a linear function of the hypothesis value to compute the importance score of the sentence.

We also tried running feature selection methods on the set of features we considered and the output from the 'Information Gain' feature selection method showed that most of the features are useful except for the Sentence position feature. This could be because of the setup of our training data. Since DUC 2002 data is for a single document summarization task, we split the main document into 10 sub documents consisting of an equal proportion of sentences and generated the training samples. So we filtered out this feature in our final set of features and ran the experiments.

Table 2: Similarity scores on sentences (A) and (B) ($*$: Un-normalized scores)

Similarity Measure	Score
Lesk	0.9
Cosine(stem)	0.62
Cosine(no-stem)	0.56
Jaccard(stem)	0.378
Jaccard(no-stem)	0.33
Resnik*	128.7

5.3 Algorithms

We initially started with the idea as mentioned in [5], but upon examining the problem at hand we decided that building a sentence dis-similarity graph and finding maximal cliques is better and more appropriate than building a sentence similarity graph and applying maximum flow solution. The problem with the latter method was that the solution or threads obtained contained a long chain of all the document sentences. This happened because the min cost solution added all possible sentences to the solution set owing to the negative weights in the flow graph.

6. RESULTS

We experimented on the DUC 2004 data and used various similarity measures, importance scores, normalizations to generate summaries. We then evaluated the summaries using ROUGE[6] with the mentioned settings.

As the similarity measure is one of the most important core components, we experimented with various similarity measures. The results of the output of various similarity measures is as shown in Table 6. The sentences considered for measuring the similarities reported in the above table are:

"Opposition parties lodged no confidence motions Wednesday against Prime Minister Mesut Yilmaz after allegations he interfered in the privatization of a bank and helped a businessman linked to a mobster."

and

"Following charges that he interfered in a privatization contract and helped a businessman with mob ties, Turkish Prime Minister Mesut Yilmaz was forced to resign."

As seen in the table, resnik similarity measure is not normalized and hence was difficult to decide on a threshold for filtering. After tuning the threshold parameter for sentence similarity, we settled for a value of 0.5.

Generation techniques: As mentioned earlier, dealing with dense graphs is difficult and hence we decided to take the top 60 sentences from each topic and build the sentence dis-similarity graph using them. The statistics for this graph are as shown in Table 4.3. The time taken for generating a

[6]ROUGE-1.5.5.pl -n 2 -x -m -2 4 -u -c 95 -r 1000 -f A -p 0.5 -t 0 -a

Table 3: Time taken to generate a summary by each method (not including the data loading time)

Method	Time(s)
Stack decoder	0.78s
Clustering based	1.62s
Graph decoder	6.42s

Table 4: ROUGE scores of summaries generated using the clustering mechanism with various similarity measures.

Similarity	ROUGE-2 (95% cinf-interval)
TFIDF	0.03371 (0.03046 - 0.03702)
Cosine	0.02918 (0.02665 - 0.03164)
Jaccard	0.03468 (0.03144 - 0.03821)
Lesk	0.03 (0.0283 - 0.0321)

summary for the graph method is as shown in Table 6 (also present are times of the other generation techniques).

Table 6 shows the ROUGE scores of the summaries generated using various similarity measures. The weights for the feature values were set based on the training output on DUC 2002 data. This is used to calculate the importance scores. As can be seen from the Table 6, the cosine and lesk similarity scores perform equally well, whereas the Jaccard similarity performs slightly better than the two. For clustering, small changes in similarity scores do not change the clusteroids significantly. Hence, the sentences which are selected for the final summary do not change a lot causing near similar scores.

Table 6 shows the ROUGE scores for various normalization schemes on summaries generated using the Stack Decoder method. The normalizations tried are sigmoid normalization and length normalization. The similarity scores tried are: TF-IDF and Cosine similarity measures. As can be seen from the Table 6, the Sigmoid norm has the best performance followed by length normalization. As expected, normalization helps to capture more information from the feature values which leads to better classification and subsequent summary generation.

Table 6 shows the final comparison of all the methods using the following parameters:

- Normalization: Sigmoid
- Similarity: Cosine
- Importance: Weights trained on DUC 2002 data

The results show that the Stack decoder method and the Graph decoder method perform nearly the same. Better similarity metrics and better training data to estimate the parameters would help in achieving higher scores. One of the entries in Table 6 which uses hand set parameters to the weights performs better showing that the parameter estimation method could be vastly improved.

7. FUTURE WORK

Some of the items that could be easily done to build upon this system are:

- As cosine similarity was fast and effective, building upon that seemed most logical. So using synset expansion based on the POS tag of the word and then performing cosine similarity might prove to be more effective than simple cosine similarity.
- Sentence compression could be used to remove/trim unnecessary parts and improve the effectiveness of the summary.
- Using discourse analysis to trim sentences is useful as we can identify extraneous phrases like elaboration etc. This would be useful in improving the performance in the clustering based technique.
- We plan to augment the feature set of the sentences with the attributes of the nodes like Page Rank, degree as mentioned in [4] and [13]. We believe that these would further help us in understanding the effect of the relationship among the sentences in summarization.
- Improving the training dataset would help us in achieving better weights that would in turn help the importance scores.
- Generate coherent summaries by reordering sentences.

8. CONTRIBUTIONS

The contributions of the team are as follows:

- All: Discussed the ideas for summary generation techniques, features for importance scores, report writing, results and analysis. Also, pair programmed common areas of the project.
- Sandeep: Worked on the stack decoder method, core component for loading, preprocessing data.
- Venu: Worked on the graph decoder based method, implementation of features.
- Gautam: Worked on the clustering based method, word net similarity measure.

9. REFERENCES

[1] R. Barzilay and K. McKeown. Sentence fusion for multidocument news summarization. *Proceedings of the Human Language Technology Conference*, pages 31–33, 2005.

[2] P. Baxendale. Machine-made index for technical literature - an experiment. *IBM Journal of Research Development*, 1958.

[3] J. Conroy, J. Schlesinger, J. Goldstein, and D. O'Leary. Left-brain/right-brain multidocument summarization. *Proceedings of the 4th Document Understanding Conference*, 2004.

[4] Erkan and D. Radev. Lexrank: Graph-based lexical centrality as salience in text. *JAIR*, pages 457–479, 2004.

[5] R. Guha, R. Kumar, D. Sivakumar, and R. Sundaram. Unweaving a web of documents. *KDD*, 2005.

Table 5: ROUGE scores of summaries generated using the Stack decoder mechanism with various normalization schemes.

Normalization	Similarity	ROUGE-2 (95% cinf-interval)
No norm	TFIDF	0.0403 (0.03821 - 0.04263)
	Cosine	0.04152 (0.03825 - 0.04499)
Length norm	TFIDF	0.04053 (0.03743 - 0.04361)
	Cosine	0.04382 (0.03991 - 0.04797)
Sigmoid norm	TFIDF	0.05359 (0.04924 - 0.05828)
	Cosine	0.05205 (0.04776 - 0.05668)

Table 6: ROUGE scores of summaries generated using all the methods ($*$ - hand set parameters).

Mechanism	ROUGE-2 (95% cinf-interval)
Stack	0.05205 (0.04776 - 0.05668)
Graph	0.05150 (0.04815 - 0.05506)
Cluster	0.02918 (0.02665 - 0.03164)
Stack$*$	0.0697 (0.06512 - 0.0732)

[6] H. D. III and D. Marcu. Bayesian multi-document summarization at mse. *Proceedings of the Workshop on Multilingual Summarization Evaluation (MSE), Ann Arbor, MI*, 2005.

[7] H. Jing and K. McKeown. Cut and paste based text summarization. *Proceedings of the 1st Conference of the North American Chapter of the Association for Computational Linguistics NAACL*, 2000.

[8] K. Knight and D. Marcu. Summarization beyond sentence extraction: A probabilistic approach to sentence compression. *Artificial Intelligence*, pages 139–139, 2002.

[9] J. Kupiec, J. Perersen, and F. Chen. A trainable document summarizer. *Research and Development in Information Retrieval*, pages 68–73, 1995.

[10] M. Lesk. Automatic sense disambiguation using machine readable dictionaries: how to tell a pine cone from an ice cream cone. pages 24–26, 1986.

[11] C.-Y. Lin and E. Hovy. Automated multi-document summarization in neats. *Proceedings of the Human Language Technology Conference*, 2002.

[12] A. Nenkova, L. Vanderwende, and K. McKeown. A compositional context sensitive multidocument summarizer: exploring the factors that influence summarization. *SIGIR*, pages 573–580, 2006.

[13] K. Patil and P. Brazdil. Sumgraph: Text summarization using centrality in the pathfinder network. *IADIS*, 2007.

[14] D. Paul. An efficient a* stack decoder algorithm for continuous speech recognition with a stochastic language model. *ICASSP*, 1991.

[15] B. Schiffman, A. Nenkova, and K. McKeown. Experiments in multidocument summarization. *Proceedings of the Human Language Technology Conference*, 2002.

[16] L. Vanderwende, M. Banko, and A. Menezes. Event-centric summary generation. *DUC*, 2004.

[17] L. Vanderwende, H. Suzuki, and C. Brockett. Task focused summarization with sentence simplification. *DUC*, 2006.

[18] Z. Xie, X. Li, and B. D. Eugenio. Using gene expression programming to construct sentence ranking functions for text summarization. *International Conference On Computational Linguistics, Proceedings of the 20th international conference on Computational Linguistics*, 2004.

[19] W. Yih, J. Goodman, L. Vanderwende, and H. Suzuki. Multi-document summarization by maximizing informative content words. *IJCAI*, 2007.