**MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY**

**BHOPAL (M.P.)**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**MINOR PROJECT**

**ON**

**TEXT SUMMARIZATION: ABSTRACTIVE AND EXTRACTIVE**

SUBMITTED IN PARTIAL FULFILLMENT FOR THE DEGREE OF BACHELOR OF TECHNOLOGY

|  |  |
| --- | --- |
| SUBMITTED BY:  RAJAT AGRAWAL (141112004)  BARUN SINGHANIA(141112006)  SIVA LIKITHAVALLURU(141112009)  ANCHIT PANDEY( 141112017)  DIVYANSH KHATRI(141112018) | UNDER THE GUIDANCE OF:  Dr. S K SARITHA  (SESSION 2016-2017) |

**MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY**

**BHOPAL (M.P.)**

****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**CERTIFICATE**

This is to certify that **Rajat Agrawal**, **Barun Singhania**, **Siva Likitha Valluru, Anchit Pandey** and **Divyansh Khatri** students of B.Tech 3rd Year (Computer Science & Engineering), have successfully completed their project “**TEXT SUMMARIZATION: ABSTRACTIVE AND EXTRACTIVE**” in partial fulfillment for the degree of Bachelor of Technology in Computer Science & Engineering.

**Dr. S K SARITHA Dr. SANYAM SHUKLA**

(Project Guide) (Project Coordinator)

**MAULANA AZAD NATIONAL INSTITUTE OF TECHNOLOGY**

**BHOPAL (M.P.)**

****

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING**

**DECLARATION**

We, hereby, declare that the following report which is being presented in the Minor Project Documentation Entitled “**TEXT SUMMARIZATION USING ABSTRACTIVE AND EXTRACTIVE APPROACH**” is the partial fulfillment of the requirements of the third year (sixth semester) Minor Project in the field of Computer Science And Engineering. It is an authentic documentation of our own original work carried out under the able guidance and the dedicated co-ordination of Dr. S K Saritha. The work has been carried out entirely at Maulana Azad National Institute of Technology, Bhopal. The following project and its report, in part or whole, has not been presented or submitted by us for any purpose in any other institute or organization.

We, hereby, declare that the facts mentioned above are true to the best of our knowledge. In case of any unlikely discrepancy that may possibly occur, we will be the ones to take responsibility.

**RAJAT AGRAWAL(141112004)**

**BARUN SINGHANIA(141112006)**

**SIVA LIKITHA VALLURU(141112009)**

**ANCHIT PANDEY( 141112017)**

**DIVYANSH KHATRI(141112018)**

**ACKNOWLEDGEMENT**

With due respect, we express our deep sense of gratitude to our respected guide and coordinator Dr. S K Saritha, for her valuable help and guidance. We are thankful for the encouragement that she has given us in completing this project successfully. Her rigorous evaluation and constructive criticism was of great assistance.

It is imperative for us to mention the fact that this minor project could not have been accomplished without the periodic suggestions and advice of our project

guide Dr. SK Saritha and project coordinator Dr. Sanyam Shukla.

We are also grateful to our respected director Dr. N. S Choudhary for permitting us to utilize all the necessary facilities of the college.

Needless to mention is the additional help and support extended by our respected HOD, Dr. R. K. Pateriya, in allowing us to use the departmental laboratories and other services.

We are also thankful to all the other faculty, staff members and laboratory attendants of our department for their kind co-operation and help. Last but certainly not the least; we would like to express our deep appreciation towards our family members and batch mates for providing the much needed support and encouragement.

**ABSTRACT**

Every day, people rely on a wide variety of sources to stay informed -- from news stories to social media posts to search results. Being able to develop Machine Learning models that can automatically deliver accurate summaries of longer text can be useful for digesting such large amounts of information in a compressed form, and is a long-term goal. It is not always possible for humans to manually read through every text. This can also be very time-consuming. Developing an algorithm to summarize text can save a lot of time and also prove to be effective. For large texts, developers and users often cannot read every document and code. They may only rely and understand just the relevant parts corresponding to their task. Such quick skimming may lead to misinterpretation and confusion of context. Reading just the titles or headers doesn’t tell enough about the meaning. There is another possible case where users may read the entire document, disregarding the size of it. This takes too long. To resolve these two problems, we provide a solution which depicts a document into a concise and fluent summary.

Extractive and abstractive summarization are two classes of text summarization. In extractive summarization, we consider most valued (weighted) sentences and words. These sentences together combine to form the summary. Here, we have used statistical method like tf-idf to generate the extractive summary. Further more interesting is abstractive summarization which is same as how a human summarizes. The summary understands the context, which is a revolution. We are using LSTM cell which is a type of recurrent neural network to train our model. Although the training requires very high computing devices, it is very efficient. As the dataset size increases and training becomes more effective. Thus, the result takes context into consideration.

However, generating long summary for articles is challenging. There are many ongoing researches going on in this field.

**TABLE OF CONTENTS**

**Certificate ii**

**Declaration iii**

**Acknowledgement iv**

**Abstract v**

**Introduction 8**

**Chapter 1. Theoretical Aspects 10**

1.1 Word Embeddings 10

1.2 Global Vectors 11

1.3 Recurrent Neural Network 11

1.4 Long Short Term Memory (LSTM) 12

1.5 Sequence to Sequence Model 13

1.6 Naïve Bayes Classifier 14

**Chapter 2. Proposed Work & Methodology 16**

2.1 Extractive Summarization 16

2.1.1 Algorithm: Extractive Summarization 21

2.2 Abstractive Summarization 22

2.2.1 Algorithm: Abstractive Summarization 25

**Chapter 3. Result Analysis 26**

**Chapter 4. Software and Hardware requirements 28**

4.1 Software 28

4.2 Hardware 28

**Conclusion and Future work 29**

**References 30**

**LIST OF FIGURES**

**Figure 1.1** Semantic relationships **10**

**Figure 1.2** An unrolled recurrent neural network. **11**

**Figure 1.3** Information persistence in RNN **11**

**Figure 1.4** Problem of long term dependency in RNN **12**

**Figure 1.5** The repeating module in a standard RNN contains a single layer. **13**

**Figure 1.6** The repeating module in an LSTM contains four interacting layers. **13**

**Figure 1.7** Sequence to sequence model **13**

**Figure 1.8** Posterior probability naïve bayes classifier **14**

**Figure 2.1** Preprocessing steps **16**

**Figure 2.2** Feature Sets **20**

**Figure 2.3** Creating embedding matrix **23**

**Figure 2.4** Encoder decoder architecture **24INTRODUCTION**

**OBJECTIVE**

The objective of the project is to provide summaries of document using different summarization techniques and to compare the quality of the summaries generated by Abstractive and Extractive models. This involves application of Neural Networks (NN) and Classification techniques to achieve the same.

**ABOUT THE MODEL**

We would be using Naïve Bayes Classifier and feature extraction techniques to generate extractive summary of the document. And in abstractive approach we would be using sequence to sequence learning model to generate headlines out of the news articles. In this case, the model reads the article text and writes a suitable headline.

**APPLICATIONS**

Text summarization can be used by personal or specialized assistants, which would be the ultimate use case. Apart from that it can be used for many personalized devices or applications. Mail clients, report generation, news feed, summarizing medical data for doctors, Multimedia News Summarization, Smart Reply for Inbox etc. Due to the limited size of displays on WAP phones and palm-top computers, it is useful to condense text found in web pages browsed. Convenient Text-to-Speech for Blind people- The idea here is to scan in a page from a book, and then read out a summary of the page rather than the entire text. Collating Search Engine hits rather than read all the pages returned by a search, it would be better to read a summary of the top N hits.

**OVERVIEW**

A summary is an informative way to cover important and relevant information of the text. Whether it is used for summarizing some parts or all parts of the text, a summary greatly reduces the reading time. A summary should be able to cover the different topics in a text without being completely redundant and wordy. Sentences shouldn’t be lengthy just to fill up the page. Summarization tools may search for headings and titles that reflect important topics in a document.

One approach to summarization is to extract parts of the document that are deemed interesting by some metric (for example, inverse-document frequency) and join them to form a summary.

Algorithms of this flavor are called extractive summarization.

**Original Text:**

*Alice and Bob took the train to visit the zoo. They saw a baby giraffe, a lion, and a flock of colorful tropical birds.*

**Extractive Summary**: *Alice and Bob visit the zoo. saw a flock of birds.*

Above we extract the words bolded in the original text and concatenate them to form a summary. As we can see, sometimes the extractive constraint can make the summary awkward or grammatically strange.

Another approach is to simply summarize as humans do, which is to not impose the extractive constraint and allow for rephrasing. This is called abstractive summarization.

**Abstractive summary**: *Alice and Bob visited the zoo and saw animals and birds.*

In this example, we used words not in the original text, maintaining more of the information in a similar amount of words. It’s clear we would prefer good abstractive summarizations.

**1. THEORETICAL ASPECTS**

**1.1 WORD EMBEDDINGS**

"Word embeddings" are a family of natural language processing techniques aiming at mapping semantic meaning into a geometric space. This is done by associating a numeric vector to every word in a dictionary, such that the distance (e.g*. L2 distance or more commonly cosine distance*) between any two vectors would capture part of the semantic relationship between the two associated words. The geometric space formed by these vectors is called an **embedding space**.

For instance, "coconut" and "polar bear" are words that are semantically quite different, so a reasonable embedding space would represent them as vectors that would be very far apart. But "kitchen" and "dinner" are related words, so they should be embedded close to each other.

Ideally, in a good embedding space, the "path" (a vector) to go from "kitchen" and "dinner" would capture precisely the semantic relationship between these two concepts. In this case the relationship is "where x occurs", so you would expect the vector kitchen - dinner (difference of the two embedding vectors, i.e. *path to go from dinner to kitchen) to capture this "where x occurs" relationship. Basically, we should have the vectorial identity: dinner + (where x occurs) = kitchen (at least approximately)*. If that's indeed the case, then we can use such a relationship vector to answer questions. For instance, starting from a new vector, e.g. *"work", and applying this relationship vector, we should get sometime meaningful, e.g. work + (where x occurs) = office, answering "where does work occur?"*

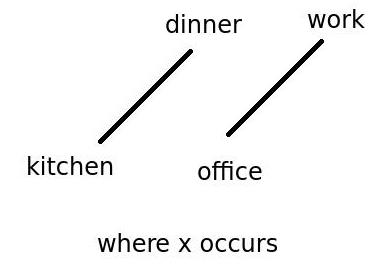


Figure 1.1:Semantic relationships

W(‘‘kitchen") –W(‘‘dinner") ≃ W(‘‘office")−W(‘‘work")

**1.2 GLOBAL VECTORS**

GloVe stands for "Global Vectors for Word Representation". It is an unsupervised learning algorithm for obtaining vector representations for words. Training is performed on aggregated global word-word co-occurrence statistics from a corpus, and the resulting representations showcase interesting linear substructures of the word vector space.

**1.3 RECURRENT NEURAL NETWORKS**

A recurrent neural network (RNN) is a class of artificial neural network where connections between units form a directed cycle. This creates an internal state of the network, which allows it to exhibit dynamic temporal behavior. Unlike feed forward neural networks, RNNs can use their internal memory to process arbitrary sequences of inputs.

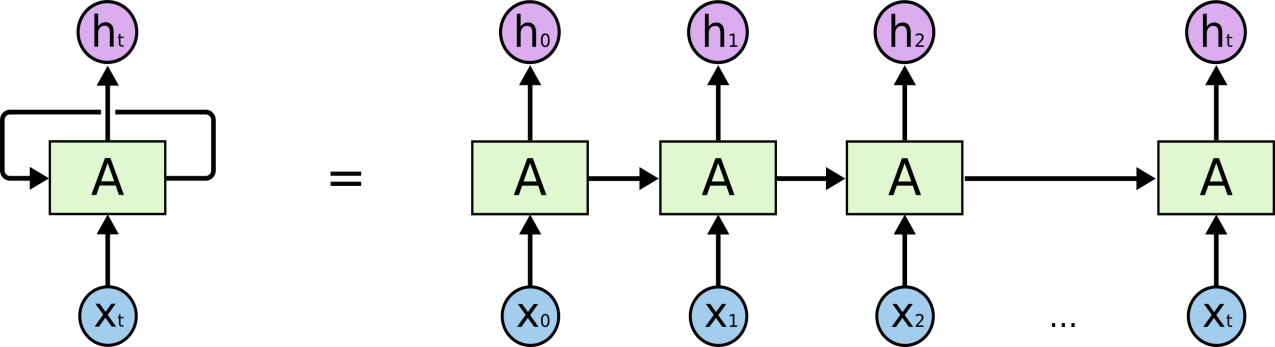


Figure 1.2: An unrolled recurrent neural network.

RNNs might be able to connect previous information to the present task.

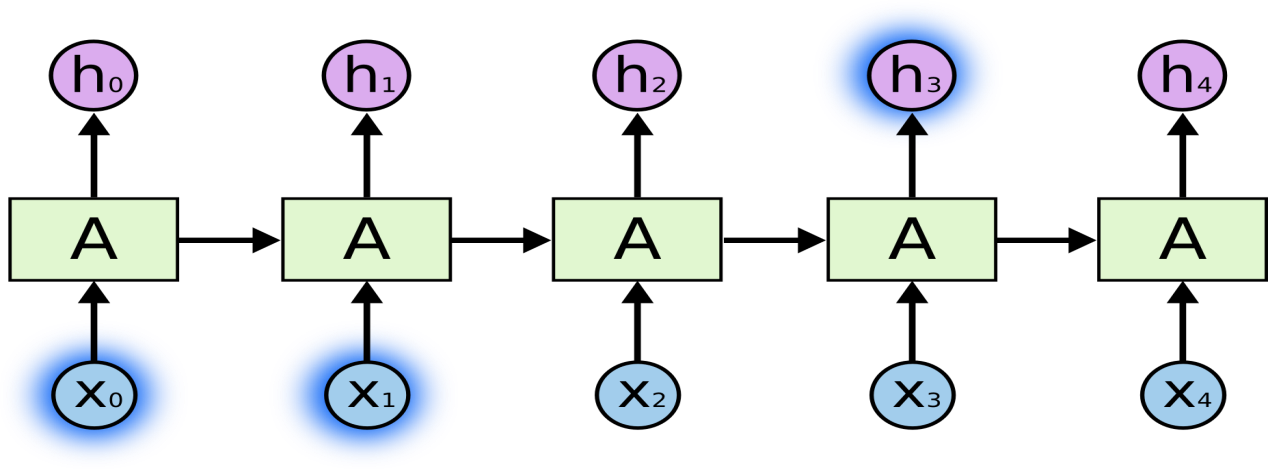


Figure 1.3: Information persistence in RNN

But there are also cases where we need more context. It’s entirely possible for the gap between the relevant information and the point where it is needed to become very large. Unfortunately, as that gap grows, RNNs become unable to learn to connect the information.

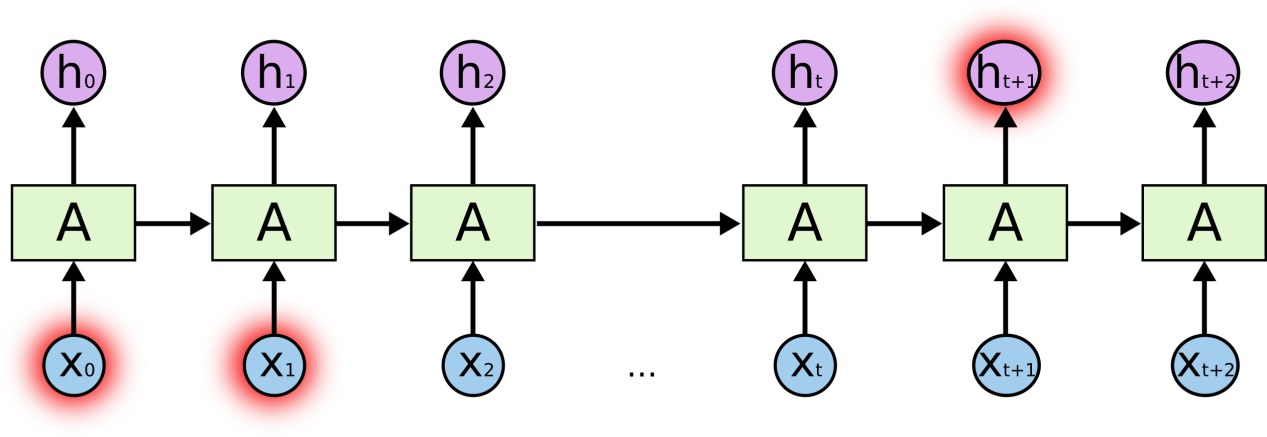


Figure 1.4: Problem of long term dependency in RNN

**1.4 LONG SHORT TERM MEMORY (LSTM)**

Long Short Term Memory networks – usually just called “LSTMs” – are a special kind of RNN, capable of learning long-term dependencies. They were introduced by Hochreiter & Schmidhuber (1997), and were refined and popularized by many people in following work. They work tremendously well on a large variety of problems, and are now widely used.

LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behavior, not something they struggle to learn.

All recurrent neural networks have the form of a chain of repeating modules of neural network. In standard RNNs, this repeating module will have a very simple structure, such as a single tanh layer.

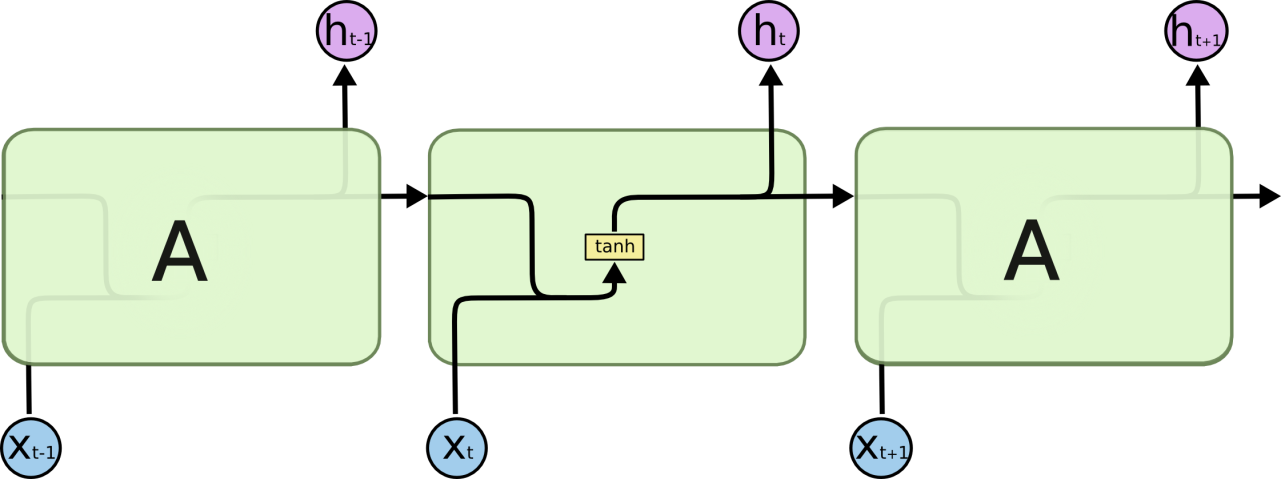


Figure 1.5: The repeating module in a standard RNN contains a single layer.

LSTMs also have this chain like structure, but the repeating module has a different structure. Instead of having a single neural network layer, there are four, interacting in a very special way.

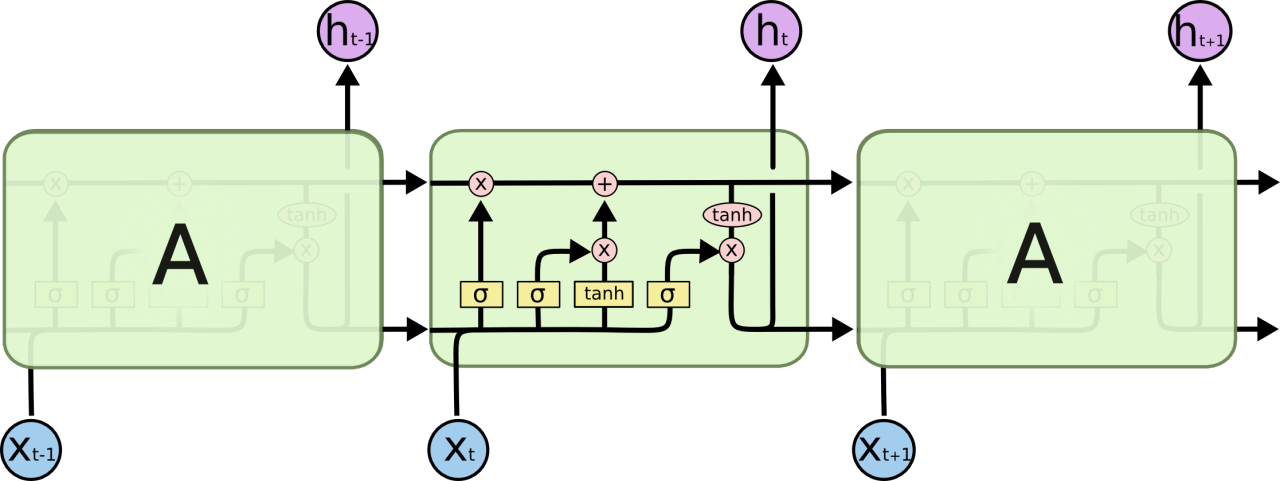


Figure 1.6: The repeating module in an LSTM contains four interacting layers.

**1.5 SEQUENCE TO SEQUENCE MODEL**

A basic sequence-to-sequence model, consists of two recurrent neural networks (RNNs): an encoder that processes the input and a decoder that generates the output. This basic architecture is depicted below.

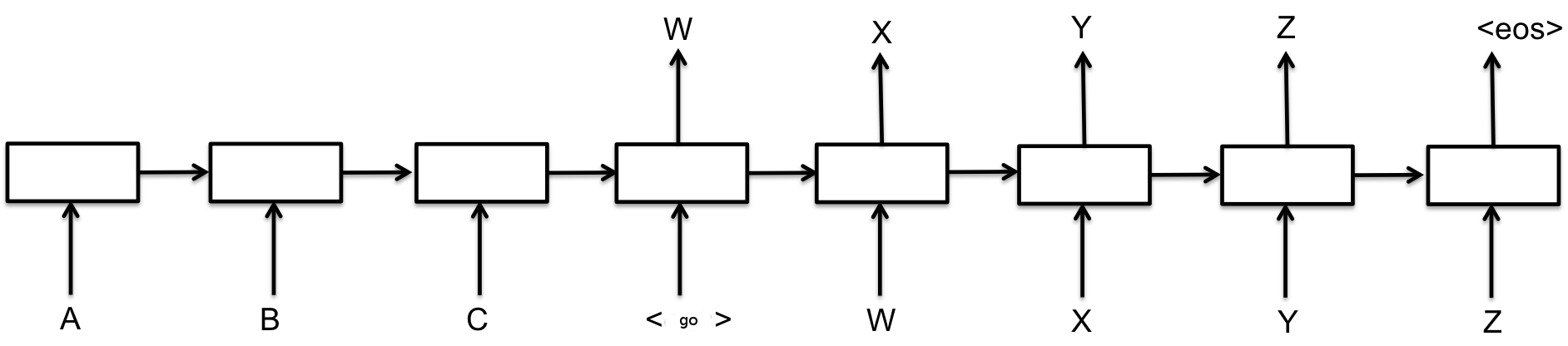


Figure 1.7: Sequence to sequence model

Each box in the picture above represents a cell of the RNN, most commonly a GRU cell or an LSTM cell. Encoder and decoder can share weights or, as is more common, use a different set of parameters. One RNN encodes a sequence of symbols into a fixed-length vector representation, and the other decodes the representation into another sequence of symbols. The encoder and decoder of the proposed model are jointly trained to maximize the conditional probability of a target sequence given a source sequence.

**1.6 NAÏVE BAYES CLASSIFIER**

The Naive Bayesian classifier is based on Bayes theorem with independence assumptions between predictors. A Naive Bayesian model is easy to build, with no complicated iterative parameter estimation which makes it particularly useful for very large datasets. Despite its simplicity, the Naive Bayesian classifier often does surprisingly well and is widely used because it often outperforms more sophisticated classification methods.

Bayes theorem provides a way of calculating the posterior probability, *P*(*c|x*), from *P*(*c*), *P*(*x*), and *P*(*x|c*). Naive Bayes classifier assumes that the effect of the value of a predictor (*x*) on a given class (*c*) is independent of the values of other predictors. This assumption is called class conditional independence.

****

Figure 1.8: Posterior probability naïve bayes classifier

*P*(*c|x*) is the posterior probability of *class* (*target*) given *predictor* (*attribute*).

*P*(*c*) is the prior probability of *class*.

*P*(*x|c*) is the likelihood which is the probability of *predictor* given *class*.

*P*(*x*) is the prior probability of *predictor*.

**2. PROPOSED WORK & METHODOLOGY**

**2.1 EXTRACTIVE SUMMARIZATION**

Extractive summarization is divided into two categories: preprocessing and processing.

**I. Preprocessing**

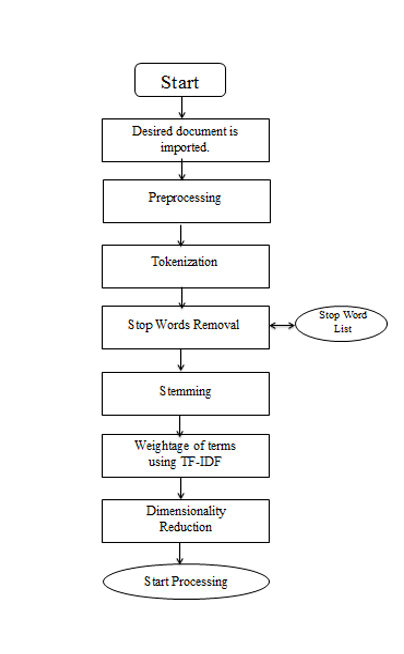
******Preprocessing is the process of turning original document into a structured and well-defined representation. It is the first and most important step done in any field of text mining.

Figure 2.1: Preprocessing steps

**Data is preprocessed using the following methods:**

**A. Tokenization**

Desired document is imported into the program. Then the entire file is word tokenized. Word tokenization is the process of dividing a flow of text into meaningful elements also known as tokens. This list of words becomes the input for further processing.

**B. Stop Words Elimination**

Stop words are those words which contain little lexicon meaning. Most common stop words are articles and prepositions. They don't give any additional meaning to the text. Examples of stop words include words such as a, an, the, in, etc. Since stop words are not considered as key words in text summarizing, they're removed. However, this paper analyzes the classical method. In this method, stop words are already present in some predefined lists. From these lists, stop words are eliminated from the document.

**C. Stemming**

Stemming is the method of identifying the root or the *stem* of a word. For example, the words run, running, ran, and runner, all have a common stem: “run”. This method is used because it removes complexity from the program by removing prefixes and suffixes. An additional plus point is that stemming also saves space. Again, there are different ways to stem a certain word. There are different kinds of *stemmers* such as Lovins stemmer, Dawson stemmer, Porters stemmer,WordNetLemmatizer etc.In this paper, we will be using the WordNetLemmatizer. In this lemmatizer, rules are applied on each word. Whichever word passes all conditions, it is converted to its root.

**D. Term Weighting**

For every document, each term has its own importance level. For each term, a weight is mapped to it. We consider three major components that affect a term's importance:

1) Term frequency (TF) – expresses importance of a word by analyzing how it's distributed in the text

2) Inverse document frequency (IDF) – expresses importance of a word by analyzing how it's distributed in a *database* of texts

3) Document normalization – adjusts TF too see how the term is dispersed. For example, there is a difference between the same term being repeatedly used in a single paragraph and the same term frequently occurring in a large document.

**E. Dimensionality Reduction**

Dimensionality reduction is used to remove the infrequent occurrences of a word. It is similar to stop word removal, although this method removes frequent words which are irrelevant to the document. For example, infrequent occurrences could be those words which occur only once or twice.

So, the constituents that make up preprocessing are tokenizing, removing stop words, stemming, calculating TF/IDF, and applying dimensionality reduction. Now, data is ready to be processed.

**II. Analyzing Features**

Summarizing a document means identifying meaningful contexts from the text, or in other words, identifying the *features* of the text. Following are the features that are considered while generating a summary:

**1) Title word feature:**

The title is one of the most prominent features in the text because it gives an idea about the document. There are great chances that sentences containing keywords from the title could be returned as part of the summary.

**2) Sentence location feature:**

Usually the first and last sentences also have a fair chance of being returned as prominent sentences. The first sentence usually sums up the rest of the document and is also known as thesis. The last sentence usually acts as a conclusion.

**3) Sentence length feature:**

Very large and very short sentences are usually not found in summaries.

**4) Proper noun feature:**

Proper noun is highly distinguished from a common noun, which usually refers to a group of entities. Proper nouns are names given to people, places, organizations, etc. Proper nouns are capitalized regardless of where they exist in a sentence.

**5) Upper-case word feature:**

Similar to proper noun feature described above, abbreviations, jargon, acronyms, etc. are also included in a summary.

**6) Cue-Phrase feature:**

Sentences containing cue phrases, such as, “in conclusion”, “therefore”, etc. also have a reasonable chance of being in a summary.

**7) Font-based word feature:**

If sentences contain bold, italic, or underlined words, then they're important.

**III. Method of Extractive Summarization**

For this project, one of the most popular algorithms used is the Naive Bayes Classifier. Naive Bayes classifier is technique based on Bayes theorem. It gets the name 'naive' because it assumes that presence or absence of a particular feature is unrelated to the presence or absence of any other feature in a given class.

Drawbacks: A drawback of the algorithm is that it makes assumptions that may or may not be true. Another drawback is that in case one feature depends on another, then results obtained Naive Bayes may not be true.

Advantages: This algorithm is fairly easy to implement. It requires only a small training dataset to form relationships between given parameters. Accurate results are obtained in most cases.

**A. Feature Sets**

When using any classifier, the first step is to decide what features of the text are relevant. Relevant features can be extracted through preprocessing. A feature set is a dictionary which maps the names of features to their values. Values are simple data types, such as booleans, strings, integers, etc. Next, we use a *feature extractor* by defining a function to process the data. By applying function on the data, we get *feature sets*. Feature sets are divided into a *training set,* a *validation set* and a *test set*. A training set is a set of data that’s used to discover predictive relationships. A validation set is a set of data which is used to determine a stopping point. Validation set is also known as tuning set. However, we will not be considering a validation test since the training set is already large. A test set is a set of data that is used to weigh the performance of the relationships using some classifier X.

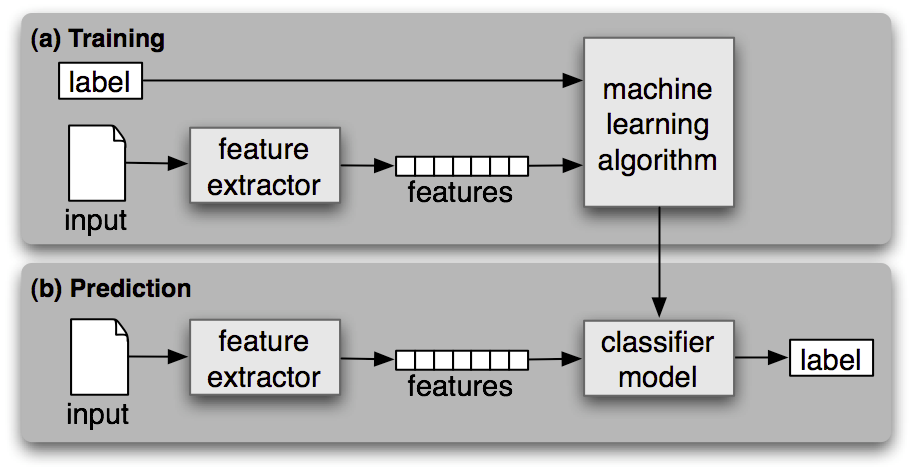


Figure 2.2: Feature Sets

Naïve Bayes classifier is *supervised* since it’s based on some labeled training corpora. Also, it is better suited for large sets of data. It is not recommended for small datasets. Data is reliable when it consists of variety and volume. Data is not generated at just one place. It must be extracted from different sources and databases.

**B. Training and Testing Sets**

After getting feature sets, we shuffle them to increase randomness. Feature sets are divided into training set and testing set. Usually, training set is about 70% of the feature sets. The remaining 30% is used as the test set.

**C. Most Informative Features**

After applying the algorithm on train and test sets, we extract the *most informative features.*

**D. Synsets**

WordNet is lexical database for the English language. It is used to group English words into synonyms known as*synsets*. Synsets provide short definitions of a word and also its usage in different sentences. Synsets are also known as synonym rings. WordNet is also referred as a dictionary and thesaurus.

By extracting synsets, we can find out the immediate synonyms of most frequently identified features.

**E. Extraction of Data**

Extract desired input text into the program. Then, text is preprocessed according to the given steps above.

**F. Weighted Sentences**

After preprocessing has been done, weights are assigned to each sentence. This means that sentences are ranked according to their importance. The sentences which are ranked the highest are returned. These sentences together constitute the summary.

**2.1.1 ALGORITHM: EXTRACTIVE SUMMARIZATION**

Program is implemented using Python 2.7, although it is compatible in Python 3.x except for a few minute pieces of syntax. For the implementation, since text processing is done, NLTK is required.

**Input: Text file**

**Output: Subset of sentences**

1. Import libraries: math, NLTK, random
2. Import one of NLTK’s default libraries: movie\_reviews
3. Categorize each of the files in movie\_reviews according to their file id and category (whether review is positive or negative). Prepare a list of each of the documents according to the id and category.
4. Using FreqDist, traverse words in movie\_reviews for the most frequent word list:

fori in movie\_reviews.words():

words.append(i.lower())

words = nltk.FreqDist(words)

word\_features = list(words.keys())[:1000]

5. Create feature sets using a function:

def features (files):

sets=set(files)

feature={}

fori in word\_features:

feature[i]=(i in sets)

return feature

6. Out of the feature sets, create a training set and test set. Apply the Naïve Bayes classifier on both of the sets. Show the most informative features from the sets using classifier\_var\_name.most\_informative\_features(n). Seven features are used in this program.

7. Store the words in a separate list and derive synonyms using wordnet.

8. Now, import URLs of different movie reviews as sample texts. Preprocess the texts using the steps for preprocessing listed above.

9. Set weights for each sentence of text using TF-IDF (prominently) and return the sentences ranked highest.

10. The printed sentences constitute the summary

**2.2 ABSTRACTIVE SUMMARIZATION**

A collection of news articles is used as training data. This dataset is converted to pickle format, which essentially means converting it into a raw byte stream. Pickling is a way of converting a Python object into a character stream. It helps to easily reconstruct that object in another Python script. The resultant data is saved as a tuple with the heading, description pair. The heading and description are the list of headings and their respective articles in order. Then the text is tokenized into individual words. The pre-trained GloVe word vectors are used to initialize an embedding matrix with the generated tokenized vocabulary from the training data. The embedding matrix is initialized with random numbers at first then all the GloVe weights of words that show up in our training vocabulary are copied in the matrix. And for every word outside this embedding matrix, find the closest word inside the matrix by measuring the cosine distance of GloVe vectors. This matrix of word embeddings is fed to first layer of the encoder.

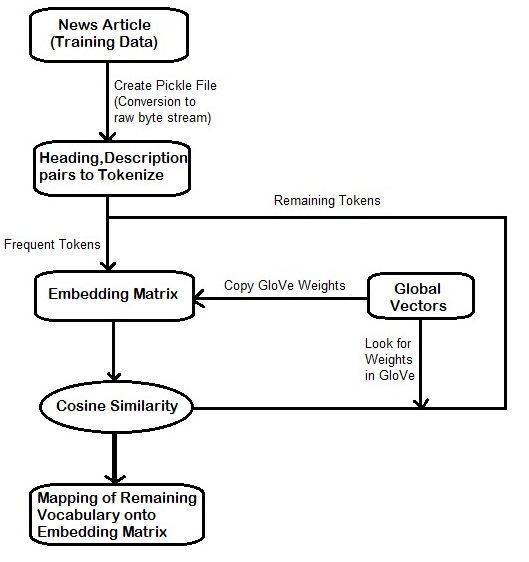


Figure 2.3: Creating embedding matrix

The word embeddings generated after the preprocessing step are used to create a summary headline for a article that is fed using sequence to sequence neural architecture. It's called sequence to sequence because it takes an input sequence and generates an output that is not a single value, but a sequence as well. So two recurrent networks one for each sequence are required. The first is the encoder network. It takes an input sequence and creates an encoded representation of it. The second is the decoder network. Feed it as its input that same encoded representation and it will generate an output sequence by decoding it. There are different ways to approach this architecture. One approach would be to let the encoder network learn these embeddings from scratch by feeding it the training data. But a less computationally expensive approach is considered, because the model has already learned embeddings from GloVe.

When encoder LSTM network is build, the pre-trained embeddings are set as the first layer's weights. The embedding layer is meant to turn input integers into fixed size vectors. And when this model is trained, it just fine tunes or improve the accuracy of the embeddings as a supervised classification problem where the input data is the set of vocabulary words and the labels are their associated headline words. The decoder generates headlines. It has the same LSTM architecture as the encoder and its weights are initialized using the same pre-trained GloVe embeddings. It takes the vector representation as an input generated after feeding in the last word of the input text. So at first it generates its own representation using its embedding layer. And the next step is to convert this representation into a word. Then the decoder generate a word as its output and that same word is fed in as input when generating the next word until a headline is generated.

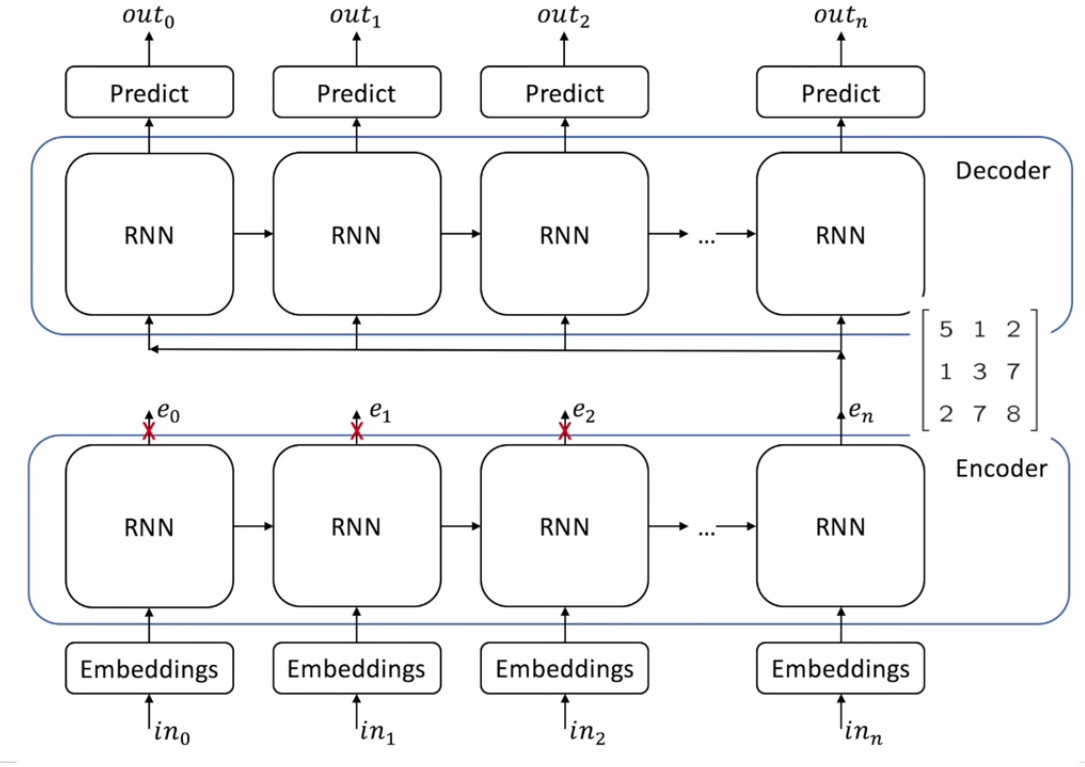


Figure 2.4: Encoder decoder architecture

**2.2.1 ALGORITHM: ABSTRACTIVE SUMMARIZATION**

This work describes to build a text summarizer that can generate a headline from a short article using Keras (Python library for Deep Learning).

**Input: News articles as heading, description pair**

**Output: Summarized headline**

1. A collection of news articles is used as the training data. Now first tokenize, or split up the text.
2. GloVe short for global vectors, a model by Stanford University is used to find word embeddings.
3. These word embeddings are fed to neural network architecture called sequence to sequence model. Thus two recurrent networks (LSTM) are used, one for each sequence.
4. The first is the encoder network. It takes an input sequence and creates an encoded representation of it.
5. The second is the decoder network. Decoder is fed as its input that same encoded representation and it will generate an output sequence by decoding it.
6. The decoder will generate a word as its output and that same word will be fed in as input while generating the next word until a headline is obtained.

**3. RESULT ANALYSIS**

**Extractive summarization results**:

Given the news article, the generated output is as follows:

**Article:** There are many ways you can help your child in school. By law you are responsible for making sure he or she goes to school regularly, and on time. But you can also help by supporting the school's rules, and its arrangements for homework. Make sure your child knows that you support the school's policies.

**Generated:** There are many ways you can help your child in school. Make sure your child knows that you support the school's policies.

The results produced in this method are quite satisfying in some cases but generally the sematic relationship between the sentences is poor.

**Abstractive summarization results**:

Given the headline and article details, we produce the generated headline during the testing phase.

**Original:** Finally time to put the college football season to rest.

**Article:** Ok, we are finally here. The college football season, at last, ends tonight in New Orleans. You know its an important game in a SERIOUS^ locale, because they’ve put the cheerleaders to work. Its Ohio State vs. LSU, for something weird thing we’re calling a `` national championship. ‘’ or something. Enjoy.

**Generated:** football play game.

**Original:** EU Says Google Must Amend^ Privacy Policy Because of Legal Flaws

**Article**: By Claire Davenport BRUSSELS^ (Reuters) – European Union data protection authorities have found legal problems with Google’s nBy^ Claire Davenport BRUSSELS^ (Reuters)- European Union data protection authorities have found legal problems with Google’s new privacy policy and asked the company to make changes, a letter from a majority of the bloc’s national regulators see

**Generated:** problem privacy Union with Google

Since there is no fair scheme to analyze the semantic correctness of the output headline but the results produced are quite relevant in the context of given description. From the above results the accuracy of this model can be assumed around 15-20%. But the accuracy is bound to increase with rigorous training.

**4. SOFTWARE & HARDWARE REQUIREMENTS**

* 1. **SOFTWARE**

The following software’s were used for this project:

**Operating System**  : Microsoft® Windows® 8, Ubuntu 14.04 or above.

**Anaconda** : Anacondais a open source distribution of the Python and

programming language, Anaconda2-4.3.1-Linux-x86\_64.

**Keras** : Keras is a high-level neural networks API, written in

Python and capable of running on top of either TensorFlow or Theano.

**Tensorflow1.0** : An open source software library for machine intelligence.

**3.2 HARDWARE**

The following hardware configuration were used to run the various softwares for this project:

**Processor** : Intel® Core™ i7 CPU

**Memory** : 8GB RAM

**Graphics card**: : NVIDIA GTX 660M 2GB

**Storage required** : Maximum of 6GB

**CONCLUSION & FUTURE WORKS**

Extractive and abstractive types of summarization both have their advantages and disadvantages. Extractive is better in some ways since it doesn’t deal with semantics and has a higher chance of success for this reason. It only selects important sentences based on statistical features. But a major drawback is that it suffers from inconsistency of contexts and proper balance in a summary.

Abstractive, on the other hand, reduces text size and generates a summary that satisfies semantic meaning. This means that the sentences are semantically related instead of just random sentences being together. A drawback is that it this doesn’t work well for more difficult datasets where reading the entire document is necessary to produce good summaries also training this model is a rigorous process and require high end hardware.

The future work in this regards include looking at more difficult datasets and hope this model serve as a baseline of summarization of these datasets and training the model in distributed environment.

In conclusion, extractive text summarization is easier to build and compute. But abstractive text summarization is better although difficult, because summary is semantically related.

**REFERENCES**

1. Patil, Aarti, Komal Pharande, Dipali Nale, and Roshani Agrawal. "Automatic Text Summarization." International Journal of Computer Applications 109.17 (2015): 18-19. Web.
2. Wong, Kam-Fai, Mingli Wu, and Wenjie Li. "Extractive summarization using supervised and semi-supervised learning." Proceedings of the 22nd International Conference on Computational Linguistics - COLING '08 (2008): 985-92. Web.
3. Siddique, Tanveer J., Dr. Multi - Document Summarization ( MLTA 2013). Powerpoint. N.p.: n.p., n.d. Print.
4. S, Vijayarani, Dr, J. Ilamathi, and Nithya. "Gaigole, Pritam C., L. H. Patil, and P. M. Chaudhari. "Preprocessing Techniques in Text Categorization." National Conference on Innovative Paradigms in Engineering & Technology (n.d.): 1-3. Web." International Journal of Computer Science & Communication Networks 5-17-16 (n.d.): 7-16. Web.
5. A, Anil Kumar, and Chandrasekhar S. " Text Data Pre-processing and Dimensionality Reduction Techniques for Document Clustering ." International Journal of Engineering Research & Technology (IJERT) 1.5 (July 2012): 1-6. Web.

[6] Generating News Headlines with Recurrent Neural Networks “https://arxiv.org/pdf/1512.

1512.01712.pdf

[7] “Sequence to Sequence Learning with Neural Networks”https://papers.nipes.cc/paper

/5346-sequence-to-sequence-learning-with-neural-networks.pdf

[8] “Tensorflow documentation” https://www.tensorflow.org/tutorials/

[9] “GloVe: Global Vectors for Word Representation” https://nlp.stanford.edu /pubs/glove

.pdf

[10]“Keras documentation” https://keras.io/

[11] “LSTM Network” http://colah.github.io/posts/2015-08-Understanding-LSTMs/

[12] “Google research blog” https://research.googleblog.com/2016/08/text- summarization-

with-tensorflow.html