Chapter 6: A Hybrid Approach to Text Summarization

This chapter the second model for Text Summarization has been discussed. This model uses Deep Learning concepts. The initial sections of the chapter are related to the concepts of Self-Organizing Maps and Artificial Neural Network that have been used in this Hybrid model. The ending sections deliberate on the implementation and result discussion.

6.1 Introduction

It is interesting to note that in the model developed and evaluated in the previous chapter related to Naïve Bayes algorithm, the work was more or less related to extractive summarization. However now the research and work goes further including the concept of Deep Learning which has started becoming a lucrative area of research and which is associated with abstractive summarization. Deep learning is a facet of artificial intelligence (AI) that is connected with emulating the learning approach that human beings use to gain certain types of knowledge. At its simplest, deep learning can be thought of as a way to automate predictive analytics. While traditional machine learning algorithms are linear, deep learning algorithms are stacked in a hierarchy of increasing complexity and abstraction.

In this chapter, a novel method generating single document Text Summarization using Deep Learning has been designed. The model uses - Self-Organizing Maps (SOM) which is an unsupervised method and Artificial Neural Networks (ANN) which is a supervised method. The work involves investigating the effect of adding mapped sentences from SOM visualization and re-training the inputs on ANN for ranking the sentences. In each individual experiment of the hybrid model, a different mapping of SOM is added to the ANN network as the input vector.

The proposed Hybrid model uses Stochastic Gradient Descent update set of parameters in an iterative manner to minimize the cost function. In addition, using back-propagation, weight is being adjusted for the input vector. The

empirical results show that the hybrid model using mapping clearly provides a comprehensive result and improves the F-score on ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-SU4. This novel method has been implemented on different documents, which are publicly available like Opinosis(Kavita Ganesan et al, 2010), DUC 2004, DUC 2006 and DUC 2007. The ROUGE toolkit has been used to evaluate summaries, which are generated from the proposed model and other existing algorithms versus human-generated summary.

6.2 An Insightinto Deep Learning

Deep learning is a machine learning technique that teaches computers to do what comes naturally to humans which is: learn by example. In today's fast developing world it is a key technology behind driverless cars, enabling them to recognize a stop sign, or to differentiate a pedestrian from a streetlight. It is the key to voice control in consumer devices like phones, tablets, TVs, and hands-free speakers. In fact it is getting fast recognition and popularity in the field of Data Mining and Text Mining. As shown in Figure 6.1, a Machine Learning process goes through a number of steps before it achieves its goal.

In deep learning, a computer model learns to perform classification tasks directly from images, text, or sound. Deep learning models can achieve state-of-the-art accuracy, sometimes exceeding human-level performance. Models are trained by using a large set of labeled data and neural network architectures that contain many layers as shown in Figure 6.2. Deep Learning is part of Artificial Neural Network. Deep Learning was introduced in 1986 by Rina Dechter. Research conducted using deep learning has been done in many fields like medical imaging, Bioinformatics, speech recognition, including Text mining and Natural language processing (NLP).

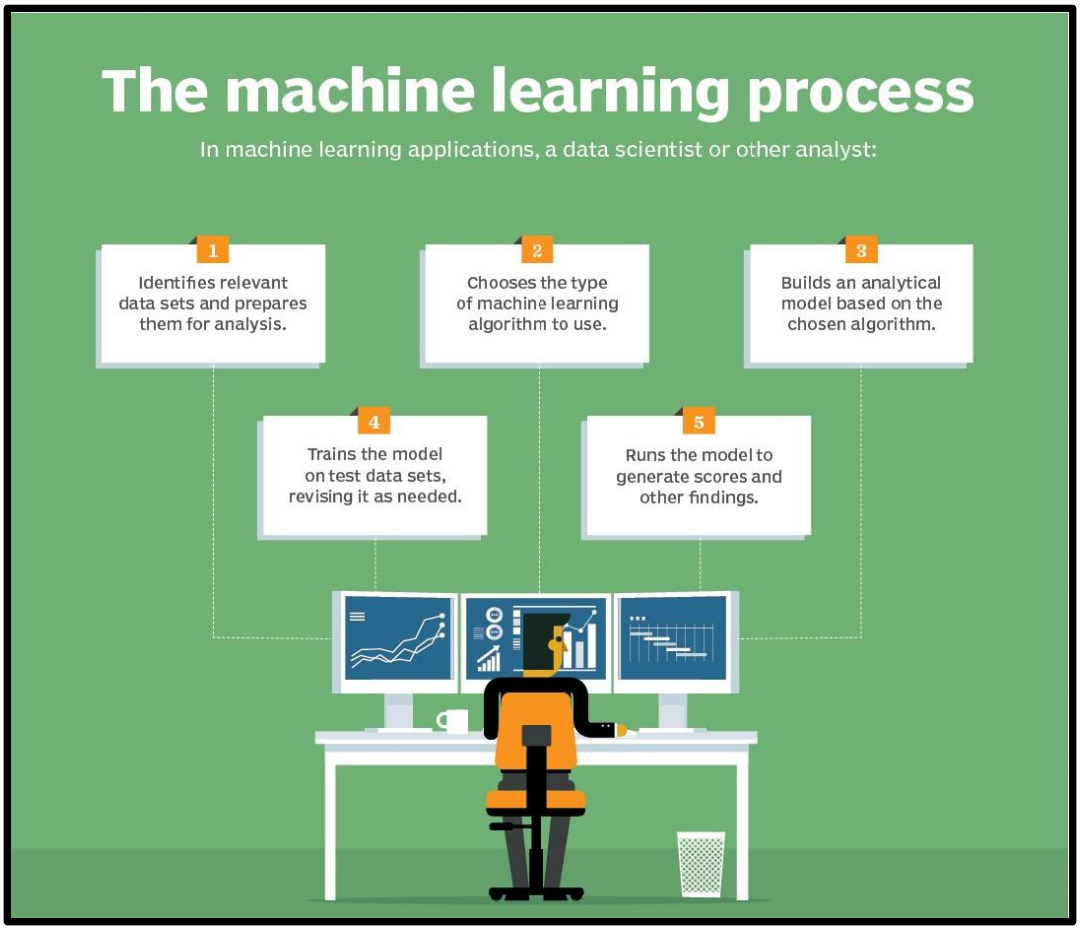


Figure 6-1 The Machine Learning Process(SearchCloudComputing, 2019)

Text mining and NLP are two fields of researches where Deep Learning has started becoming very popular. In both of these fields, processing text from unstructured documents, and generating information from it is the core of the research that is being done currently. Both text mining and NLP are required to handle this document surge. Deep learning is associated with the machine-learning concept and the working of the human brain. These methods show excellent performance in areas related to Natural Language Processing (NLP) tasks (Nitish Srivastava and Ruslan R Salakhutdinov, Ronan Collobert et al).

Working with unstructured text data is hard especially when you are trying to build an intelligent system which interprets and understands free flowing natural language just like humans. You need to be able to process and transform noisy, unstructured textual data into some structured, vectorized formats which can be understood by any machine learning algorithm. Principles from Text Mining, Natural Language Processing, Machine Learning or Deep Learning all of which fall under the broad umbrella of Artificial Intelligence are effective tools of the trade.

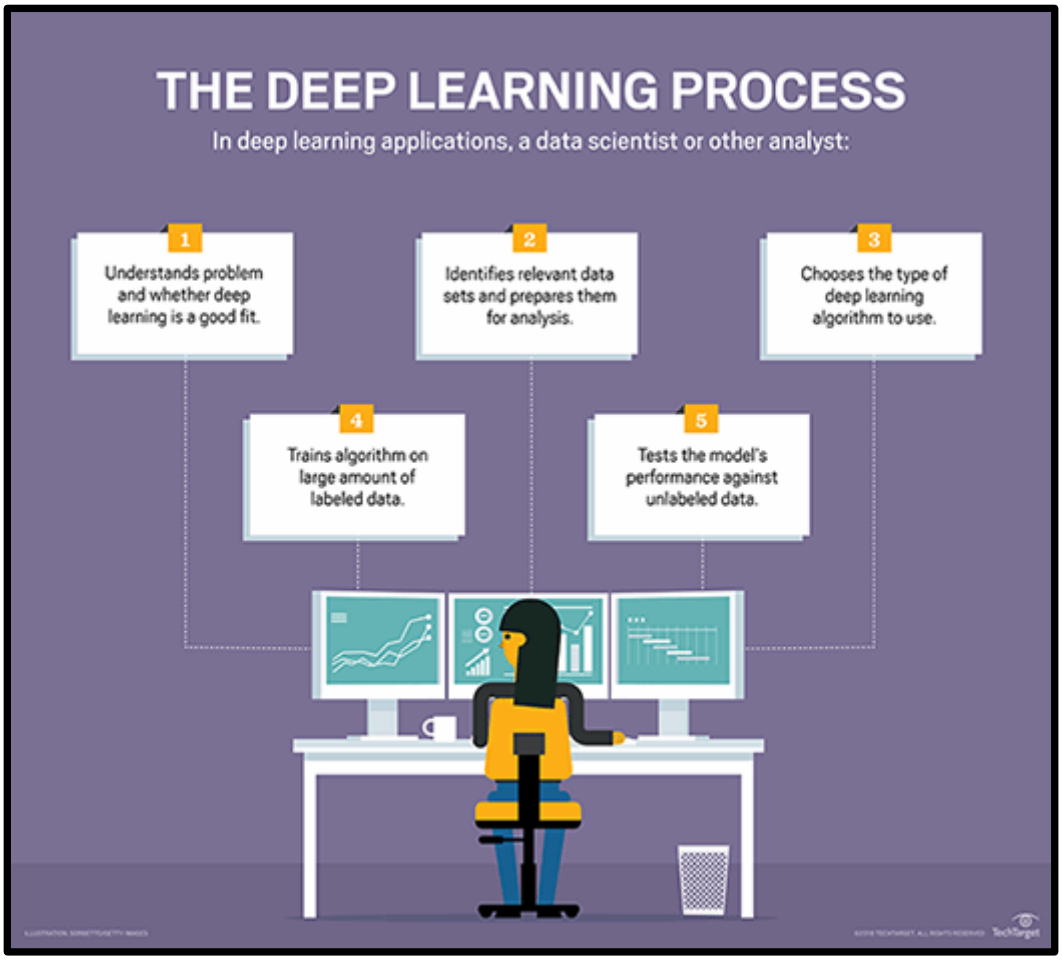


Figure 6-2 The Deep Learning Process(SearchCloudComputing, 2019)

In this chapter, we have presented a method of extracting single document Text Summarization using Deep Learning method - Self-Organizing Maps (SOM) which is an unsupervised method and Artificial Neural Networks (ANN) which is a supervised method. The work involves investigating the effect of adding mapped sentences from SOM visualization and re-training the inputs on ANN for ranking the sentences. In each individual experiment of the hybrid model, a different mapping of SOM is added to the ANN network as the input vector.

The proposed Hybrid model uses Stochastic Gradient Descent update set of parameters in an iterative manner to minimize the cost function. In addition, using back-propagation, weight is being adjusted for the input vector. The empirical results show that the hybrid model using mapping clearly provides a comprehensive result and improves the F-score on ROUGE-1, ROUGE-2, ROUGE-L, and ROUGE-SU4. This novel method has been implemented on different documents, which are publicly available on Opinosis (Kavita Ganesan et al, 2010) Dataset which is a standard Text Summarization dataset and the DUC

2004 dataset about which has been explicitly documented in the previous chapter. The ROUGE toolkit has been used to evaluate summaries, which are generated from the proposed model and other existing algorithms versus human-generated summary.

6.3 The Proposed Model

It is very important to remember that any machine learning or deep learning algorithm is based on principles of statistics, math and optimization. Hence they are not intelligent enough to start processing text in their raw, native form. It is important and pertinent that this unstructured data is pre-processed as per the algorithm and method to be applied before actually implementing the Text Summarizing task on it.

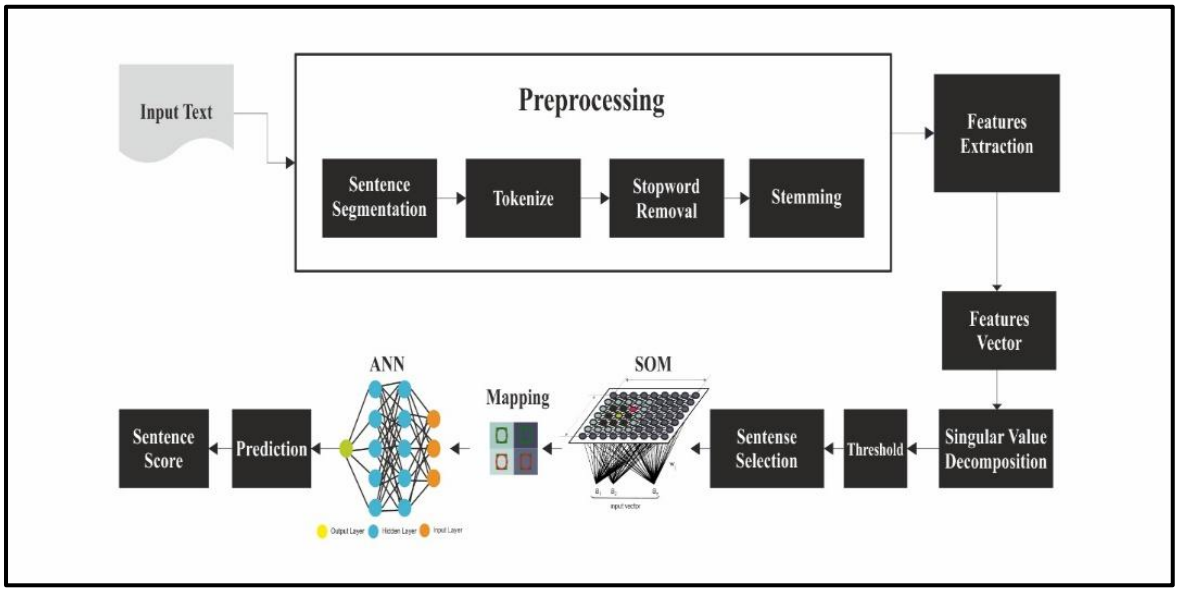


Figure 6-3 The Hybrid Model

As described before in previous chapters, traditional feature engineering strategies for textual data involve models belonging to a family of models popularly known as the Bag of Words model. This includes term frequencies, TF-IDF (term frequency-inverse document frequency), N-grams and so on. While they are effective methods for extracting features from text, due to the inherent nature of the model being just a bag of unstructured words, we lose additional information like the semantics, structure, sequence and context around nearby words in each text document. This forms as enough motivation for us to explore more sophisticated models which can capture this information and give us

features which are vector representation of words, popularly known as embedding.

Implementing the Bag of Words model leads to huge sparse word vectors for textual data and thus if we do not have enough data, we may end up getting poor models or even overfitting the data due to the curse of dimensionality. To overcome the shortcomings of losing out semantics and feature sparsity in bag of words model based features, we need to make use of Vector Space Models (VSMs) in such a way that we can embed word vectors in this continuous vector space based on semantic and contextual similarity. In fact the distributional hypothesis in the field of distributional semantics tells us that words which occur and are used in the same context are semantically similar to one another and have similar meanings.

Count-based methods like Latent Semantic Analysis (LSA) which can be used to compute some statistical measures of how often words occur with their neighboring words in a corpus and then building out dense word vectors for each word from these measures. LSA has already been discussed in detail and now in over here the model tries to improvise on it using deep learning techniques.

Predictive methods like Neural Network based language models try to predict words from its neighboring words looking at word sequences in the corpus and in the process, it learns distributed representations giving us dense word embedding.

The Hybrid model has been divided into a number of steps. The model is shown in the form of a diagram in Figure 6.3. The algorithm of this model is described as follows:

Step 1: Analyze the text

- Sentence segmentation, based on the boundary
- Tokenize, broken into words
- Stop word removal
- Stemming

Step 2: Feature Extraction & Feature Vector

- Compute frequency of occurrence(f_i) of each term (t_i) which is appearing in the document (TF)

- Computer Inverse document frequency (IDF)

Step 3: Latent Semantic Analysis

- Input matrix creation, where columns are sentences and rows are words
- Calculate Singular Value Decomposition, $A=U\sum V^T$, where A is input matrix, U is Extracted concepts X Words, ∑ Scaling values. V^T Sentences X Extracted Concepts

Step 4: Threshold

- Apply specific threshold on V, where t < 0.5

Step 5: Self Organizing Maps

- Initialize each node's weights
- Select a random vector
- Consider Best Matching Unit (BMU), using Euclidean distance find similarity between two sets

  $D = \sqrt{\sum_{i=0}^{i=n}(Vi - Wi)^2}$

- Deterring the BMU neighborhood

  $\sigma(t) = \sigma_0 \exp(-\frac{t}{\lambda})$

- Modify Node's weights

  $W(t+1)=W(t)+ \Theta(t)L(T)(V(t)-W(t))$

Step 6: Mapping

- Get coordinates as per threshold
- Returns a dictionary Wm where Wm[(i,j)] is a list with all the patterns that have been mapped in the position(i,j)

Step 7: Deep Neural Network

- Randomly initialize weights, where weight < 0
- Input the observation
- Forward propagation
- Generate error
- Backpropagation

Step 8: Sentence Score

- Sort the sentence with scores in descending order

This model too like the Naïve Bayes model discussed in Chapter-5, was developed using Python Version 3.6.5. Python is an interpreted, high-level, general-purpose programming language. Python has a design philosophy that emphasizes code readability notably using significant white space. It provides constructs that enable clear programming on both small and large scales. Further the Machine Learning Repository: scikit-learn which is a simple and an efficient tool for data mining and data analysis was also used.

Each step of the model are described in detail in subsequent sections.

6.3.1 Pre-processing the Data

a) Split the text into sentences.
b) Remove all unnecessary characters. In this step, all unnecessary characters like punctuations, symbols will be removed.
c) Remove stop words.
d) Convert all word into lower case. All words are converted into the lower case with Python built-in function lower().
e) Stemming performed on each word using the Porter Stemmer.

Preprocessing steps are discussed in detail in Chapter-3.

6.3.2 Feature Extraction

The occurrence of a word in a file: It is known as the term-document matrix. A mathematical matrix explains the occurrence of the term in a collection of text. Word (or n-gram) frequencies are typical units of analysis when working with text collections.

From occurrence to frequencies (TF-IDF): Term frequency-inverse document frequency, is a numerical method to understand the importance a word is in Corpus. The TF-IDF score has been discussed in detail in previous chapters. Different types of TF-IDF weighting methods used for scoring and ranking a document.

$$S = TF * IDF \tag{6.1}$$

$$TFi = \frac{Ti}{\sum_{k=1}^{n} Tk}, IDF = log\frac{N}{ni} \tag{6.2}$$

Feature extraction steps have also been discussed in detail in Chapter – 3.

6.3.3 Latent Semantic Analysis

Latent Semantic Analysis (LSA) is a statistical technique for extracting the meaning of contextual-usage of words by statistical calculations applied to a huge corpus of text. The principal aim, the information about all the word perspectives in that a given word appears, delivers a set of shared restrictions that largely regulates the similarity of meaning of words and a set of words related to each other. The competence of LSA's reflection of human information has been proven in a variety of ways.

Detailed discussion of LSA has been done in Chapter – 4.

6.3.4 Threshold Generation

Latent Semantic Analysis (LSA) is a statistical theory for extracting and representing words from the large dataset by contextual-usage meaning (Landauer, T. K., & Dumais, S. T. 1996). Latent Semantic Analysis is used for dimension reduction also. After performing the pre-processing steps in Text Summarization and once the corpus is ready, the document matrix and TF-IDF matrix is built on the corpus for each term. Singular Value Decomposition (SVD) is applied to the TF-IDF matrix and different concepts are generated. In the hybrid approach, based on the threshold, the value given in sentences for prediction is either 1 (part of the summary) or 0 (not part of the summary). After retrieval of relevant sentences, the model will visualize sentences using an unsupervised technique of deep learning, i.e. SOM (Self-Organizing Maps). During visualization, sentences are overlapped with each other, so the model will map those sentences and retrain all sentences using Artificial Neural Network (ANN), which is a supervised method of Deep Learning. In ANN, each sentence will get a score according to the importance of the sentence and the model will sort the sentences as part of a summary generation.

6.4 Self-Organizing Maps (SOM)

The Self-Organizing Maps (SOM) was first introduced by T. Kohonen (1982). SOM provides a data visualization technique which helps to understand high dimensional data by reducing the dimensions of data to a map. It also

represents clustering concept by grouping similar data together. Therefore it can be said that SOM reduces data dimensions and displays similarities among data.

With SOM, clustering is performed by having several units compete for the current object. Once the data have been entered into the system, the network of artificial neurons is trained by providing information about inputs. The weight vector of the unit which is closest to the current object becomes the winning or active unit. During the training stage, the values for the input variables are gradually adjusted in an attempt to preserve neighborhood relationships that exist within the input data set. As it gets closer to the input object, the weights of the winning unit are adjusted as well as its neighbors.

It is an unsupervised deep learning algorithm which is used in text mining, data mining, visualization for data, image & speech recognition, medical & medicine industry and natural language processing (T. Kohonen 1982). The SOM maps M-dimensional input vector a_j to two-dimensional neurons as per features, which converts the high-dimensional facts into a map which groups similar data together and it help us to understand high-dimensional data. The SOM has two layers. The first includes input space, whereas in the second, which consist of output space.

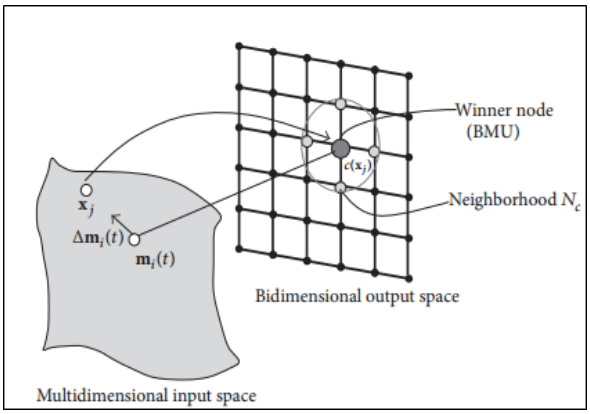


Figure 6-4 SOM Training Algorithm (Pacella, Grieco and Blaco, 2016)

Figure 6.4 explains the grid view of SOM through output nodes in two-dimensional space. SOM includes P units, the index of each unit is associated with M-dimensional vector m_i in the input space and vector on a low-dimensional regular grid, r_i, in the output space. The algorithm of SOM is given below.

1. Initialize each node weights with random values if any prior information is not available. This value of a node can be linear or random and, adjusted after the network learns.

2. Choose an arbitrary vector from giving the training data. It can be randomly selected.
3. Find the Best Matching Unit (BMU), calculate the distance between node weights (w1, w2, w3….,wk) and an input vector (V1,V2, V3 …, Vj). Vector age is compared to all possible vectors a_j and Index c (a_j) of the Best Matching Unit (BMU). Then find smallest Euclidian Distance (measurement of similarity between two datasets) of BMU, that is, chose original vector $n_{c,}$ which is closest to a_j as follows:

$$\left\| a_j - n_c \right\| = \min_i \left\| a_j - n_i \right\| \tag{6.3}$$

4. Determining the BMU neighborhood, on every iteration, an exponential decay function shrinks the size of the neighborhood until it becomes BMU itself.
5. Update the Best Matching Unit (BMU) and its neighbors. Tuning of vector and winning node and its neighbor updates as

$$n_i\left(t+1\right) = n_i(t) + \Delta n_i(t) \tag{6.4}$$

Where $t = 0,1,2....$ is an index of time. The value of $\Delta n(t)$ is computed as follows:

$$\Delta n_i(i) = \alpha(t) h_{ci}(t)(a_j(t) - n_i(t)) \tag{6.5}$$

Where α(*t*) is the learning rate, $h_{ci}(t)$ the neighborhood function. The proportion of learning has remained between 0 & 1 and it will decrease through the learning phase. The $h_{ci}(t)$ neighborhood technique regulates the distance from nodes to keys in the output layer.

During the training phase of SOM, the step between 2 and 4 will repeat iterations until the Vector a_i represent the input patterns that are closer to the node for two-dimensional neurons. After initialization, the SOM can be trained in a sequential or batch manner. Sequential training is repetitive as batch training but instead of sending all data vectors to the map for weight adjustment, one data vector at a time is sent to the network. Once the SOM is trained, each input vector is mapped to one neuron of the map, reducing high dimensional input space to a low-dimensional output space. The map size depends on the type of application. The bigger size map reveals more details of information whereas a smaller map is being chosen to guarantee the generalization capability.

6.4.1 SOM Based Text Summarization

SOM based Text summarization is an unsupervised method used to extract sentences from the document based on the similarity between documents. Suppose α = {d1, d2… d_N} are a collection of N sentences to summarize. The determination of text summarization is extracting meaningful sentences from the corpus.

Text summarization using SOM can be divided into two main phases (T. Honkela et al, 1997 & S. Kaski et al, 198). The primary phase is document preprocessing and second, text summarization.

Document preprocessing: The preprocessing of the document is very important for text summarization. With the help of preprocessing, unnecessary characters, token will removed from the document. Segmentation, stop word removal and tokenization is performed in the preprocessing stages which is explained in the preprocessing stage.

In the second stage, TF-IDF (Term Frequency-Inverse Document Frequency) will be calculated for the document. Once it is completed, SVD (Singular Value Decomposition) is performed to extract sentences from the corpus. Details of TF-IDF and SVD have been explained before.

Training SOM: After obtaining feature vector aj from calculating Singular Value Decomposition (SVD) and threshold on concept selection associated with the text document Dj, on the selected feature vectors, SOM algorithm can be applied as discussed in the above section. Here, in text summarization, MiniSom (PyPI, 2018) of Python is used to initialize SOM. The system has to use 10 X 10 dimension of the SOM. Also in SOM, we need to provide the number of elements of the vector to be trained and that number can be random as per selection of Vector a_j. Another argument, that is, σ– the spread of the neighborhood of Gaussian method, has to be satisfactory to the scope of the map.

$$\sigma(t) = \frac{\sigma}{1 + t/T} \tag{6.6}$$

Where T is the number of iterations(t) divided by 2.

$$\text{learning\_rate(t)} = \frac{\text{learning\_rate}}{1 + t/T} \tag{6.7}$$

6.4.2 SOM Visualization

SOM is a very useful method for visualizing high-dimensional data to reduced dimensional form. SOM uses the Unified distance matrix (A. Ultsch and H. P. Siemon, 1990) and Component Planes(J. Vesanto, 1999). The distance between neighboring maps can be achieved by U-Matrix and these distances can be visualized using the different color gauge on the map [1 add to references-added at the end of chapter].

The U-matrix method is a solo graph, which displays the graph borders, giving to the differences between neighbors components. The distance range of the U-matrix that can be displayed on the map is demonstrated by different shades of gray. Large distances are depicted by white color, that is, a large gap occurs between the vector scores in space of input. Whereas gray color represents small distance, that is, units are strongly clustered together. Visualizing closely connect i.e. clusters in input data of without having a priori information is possible with U-matrix. It is a versatile tool for visualizing clusters.
Another component of visualizing input is Component Planes, which is a grid whose cell contains the value of the input vector, which is, displayed by different type of colors. Through these Component Planes, analyzing the contribution of individual variables in forming clusters and the associations between them can be found. After applying SOM Algorithms with specific arguments, the Self-Organizing map is ready for plotting.

6.5 Mapping of Input and Output

During the mapping phase, the relation between the input vector and output layer nodes can be determined, after that, all input vector or pattern can be mapped onto the output layer nodes, after the training stage of SOM.

As shown in Figure 6.5, the green square indicates that sentences are part of the summary, whereas the red circle shows that it is not part of the summary. However, as per figure, there are some sentences that fall together in the same area, which means, green squares and red circles are falling in one common area.

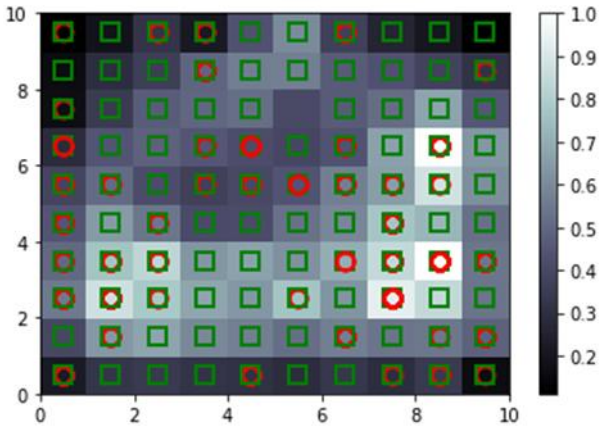


Figure 6-5 SOM Visualization

As per the model, these are the sentences which the system has to map for re-training. For same, the system has to set the value according to requirement. Initially, system selected highest value from SOM visualization, then system matches coordinates for mapping those sentences. For example, if system chooses threshold value as 1,in that case the coordinates are (8, 3) and (8, 6) are added to the dictionary.

Once the dictionary is ready, in the next stage of the model, we have to transfer from unsupervised learning to supervised learning. Therefore, we have to create a dependent variable for sentence. By default, as per the total number of sentences, first, create a vector for all sentences with value 0. During the next iteration, if the sentence is in the dictionary, then replace the value by 1.

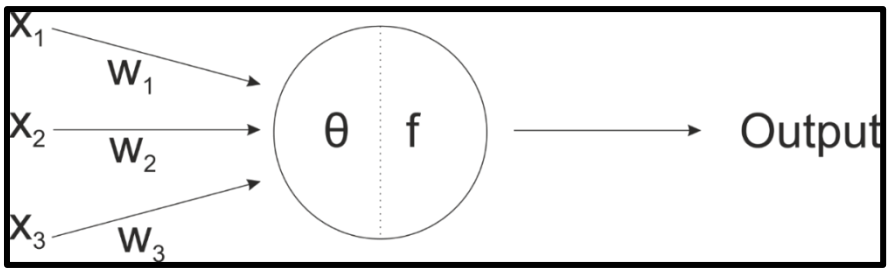
For re-training, the supervised method of deep learning that is neural networks along with hidden layer has been used. Also, the model is using the necessary optimizer and activation function for different layers, which is discussed in detail in the next section.

6.6 Introduction to Deep Neural Network

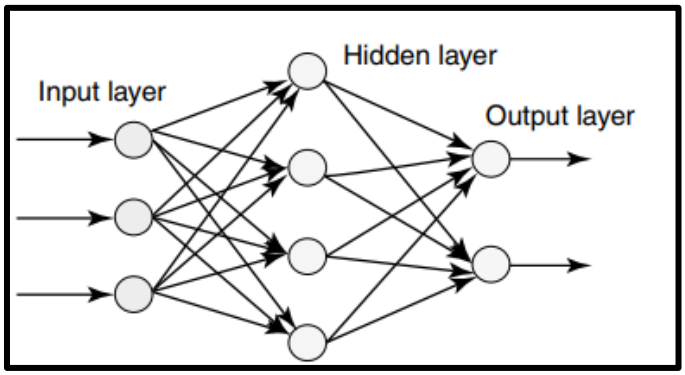
Artificial Neural Network (ANN) has been established as a generalization of scientific fields of biological neuron system. McCulloch & Pitts (1943) first introduced the idea of neural network. Artificial Network or neurons are a primary processing element of neural network.

In a simplified mathematical model of the neuron, the effects of the synapses are represented by connection weights that modulate the effect of the associated input signals, and the nonlinear characteristic exhibited by neurons is

represented by a transfer function. The neuron impulse is then computed as the weighted sum of the input signals, transformed by the transfer function. The learning capability of an artificial neuron is achieved by adjusting the weights in accordance to the chosen learning algorithm.



(a) Artificial Neuron



(b) Multilayered artificial neural network

Figure 6-6Deep Neural Network and Multilayered Deep Neural Network architecture

Figure 6.6 depicts the simple neuron system and multi-layered neural network. The inputsX1, x2, … ,xn are measured as unidirectional that are shown via arrows. The neuron output signal flow is shown as O, which is simplified in below equation.

$$o = f(net) = f\left(\sum_{j=1}^{n} w_j x_j\right) \tag{6.8}$$

The weight vector is denoted by W_j. Activation function is considered as *f(net)*. Variable net is the scalar product of the input vectors and the weight.

$$net = w^T x = w_1 x_1 + \ldots\ldots + w_n x_n \tag{6.9}$$

In the above equation, T is considered as matrix transpose. In equation 6.8,the output value O is calculated.

$$o = f(net) = \begin{cases} 1 \text{ if } & w^T x \geq \theta \\ 0 & \text{otherwise} \end{cases} \quad (6.10)$$

Here, we set the threshold level, which is indicated by Θ, is known as a linear threshold limit.

6.6.1 The Architecture of Deep Neural Network

The architecture of a neural network consists of three types of layers: input, hidden and an output layer. The signal flow in the feed-forwardnetwork goes from input to output in a direction of feed-forward. The data pre-processing divides into a number of layers, however, there are no feedback connections available. Feedback connections are available in a recurrent network. Contrary to feed-forward networks, the dynamical properties of the network are important.

A neural network is to be configured in such a manner that a set of inputs produces a requisite set of outputs. Different types of methods are available for the configuration. First one is to adjust the weight explicitly with the help of the prior knowledge. The second approach is that train the neural network by feeding it teaching patterns then after the network adjusts the weight as per the learning rate. The learning options in neural network can be divided into three categories namely supervised learning, unsupervised learning and reinforcement learning.

In supervised learning, an input vector is presented at the inputs together with a set of desired responses, one for each node, at the output layer. A forward pass is done, and the errors or discrepancies between the desired and actual response for each node in the output layer are found. These are then used to determine weight changes in the net according to the prevailing learning rule. The term supervised originates from the fact that the desired signals on individual output nodes are provided by an external teacher.

The best-known examples of this technique occur in the backpropagation algorithm, the delta rule, and the perceptron rule.

In unsupervised learning or self-organization, a output unit is trained to respond to clusters of pattern within the input. In this paradigm, the system is supposed to discover statistically salient features of the input population. Unlike the supervised learning paradigm, there is no a priori set of categories into which

the patterns are to be classified; rather, the system must develop its own representation of the input stimuli.

Reinforcement learning is learning what to do – how to map situations to actions – so as to maximize a numerical reward signal. The learner is not told which actions to take, as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward, but also the next situation and, through that, all subsequent rewards. These two characteristics, trial-anderror search and delayed reward are the two most important distinguishing features of reinforcement learning.

6.6.2 Neural Network Learning

There are different types of Neural Network Learning. The ones that are popular and which have been implemented in the model are described below.

The Hebbian learning

In this learning approach, adjustment of weight for inputs in network connections are done as per some modification rules. The best approach in connections is the Hebb approach (1949). He presented a theory of behavior based on the physiology of the nervous system.

The Hebb's postulate or his formal statement was what emerged as the most significant part of how learning can occur from his work. Learning was based on the modification of synaptic connections between neurons. Specifically, when an axon of cell A is near enough to excite a cell B and repeatedly or persistently takes part in firing it, some growth process or metabolic change takes place in one or both cells such that A's efficiency, as one of the cells firing B, is increased. The principles underlying this statement have become known as Hebbian Learning.

It can be said that most of the neural network learning techniques are a modified version of the Hebbian learning rule. The basic idea is that if two neurons are active concurrently, their interconnection must be made stronger. In a single layer net, one of the interconnected neurons will be an input unit and one an output unit. If the data are represented in bipolar form, it is easy to express the desired weight update as:

$$w_i\left(new\right) = w_i\left(old\right) + x_i o \tag{6.11}$$

Here *O* is the desired output for inputs which are i = 1 to n.

The perceptron learning

Another method is the perceptron learning wherein it is a single layer neural network whose weights and biases could be trained to produce a correct target vector when presented with the corresponding input vector. The training procedure used is termed as the perceptron learning rule. Perceptrons are especially appropriate for simple problems in pattern classification.

Suppose we have a set of learning samples consisting of an input vector x and a desired output d(k). For a classification task, the d(k) is usually +1 or −1. The perceptron learning rule is very simple and can be stated as follows:

1. Start with random weights for the connections.
2. Select an input vector x from the set of training samples.
3. If output $y_k \neq d(k)$ (the perceptron gives an incorrect response), modify all connections w_i according to:
   $\delta w_i = \eta(d_k - y_k)x_i$ where η = learning rate
4. Go back to step 2.

The difference between Hebb learning and Perceptron learning is that, once the network responds properly, the connection weight dos not change.

Backpropagation Learning

Normally it is observed that a simple perceptron is just able to handle linearly separable or linearly independent problems. To discern the direction in which the error of the network is moving, we need to take the partial derivative of the error of the network with respect to each weight. In fact, if we take the negative of this derivative (i.e. the rate change of the error as the value of the weight increases) and then proceed to add it to the weight, the error will decrease until it reaches a local minima. It is obvious from this that when the derivative is positive, it tells us that the error is increasing when the weight is increasing.

The apparent thing to do then is to add a negative value to the weight and vice versa if the derivative is negative. The reason why this algorithm is called back propagation is that after taking the partial derivatives they are applied to each of the weights beginning from the outer layer to the hidden layer weights.

After that,fro hidden layer to the input layer weights which means it's going backward. It is essential since changing these set of weights requires that we know the partial derivatives calculated in the layer downstream.

There are basically two different modes in a neural network. They are the online and the batch modes. In each of these modes, the number of weight updates for the same number of data presentations is very different. The online method weight updates are computed for each input data sample, and the weights are modified after each sample. An alternative solution is to compute the weight update for each input sample, but store these values during one pass through the training set which is called an epoch. At the end of the epoch, all the contributions are added, and only then, the weights will be updated with the composite value. This method adapts the weights with a cumulative weight update, so it will follow the gradient more closely. It is called the batch-training mode.

Training basically involves feeding training samples as input vectors through a neural network, calculating the error of the output layer, and then adjusting the weights of the network to minimize the error. To make derivative easier, the average of all square errors (E) is calculated. After the calculation of an error, the weights are updated one after the other. In the batched mode variant, the descent is based on the gradient ∇E for the total training set:

$$\Delta w_{ij}(n) = -\eta^* \frac{\delta E}{\delta w_{ij}} + \alpha^* \Delta w_{ij}(n-1) \qquad (6.12)$$

Here η is the learning rate and α is the momentum.

The momentum term determines the effect of past weight changes on the current direction of movement in the weight space. To assure the training success and the speed of the neural network learning, a perfect choice of both η and α are required. It has been proven that backpropagation learning with sufficient hidden layers can estimate any nonlinear function to subjective accurateness.

6.6.3 Gradient Descent

Gradient Descent is one of the most preferred techniques to reach optimization for neural networks. Every deep learning framework includes the

implementation of numerous techniques to enhance gradient descent such as Lasagne's[1], Caffe's[2], and Keras'[3].

Gradient descent is a method to minimize an objective function J(θ) parameterized by a model's parameters $\theta \in R^d$ by updating the parameters in the opposite direction of the gradient of the objective function$\nabla_\theta J(\theta)$w.r.t. to the parameters. The learning rate η determines the size of the steps we take to reach a (local) minimum. In other words, we follow the direction of the slope of the surface created by the objective function downhill until we reach a valley [Sebastian Ruder, 'An overview of gradient descent optimization algorithms', 2016].

Types of Gradient Descent

In all there are three alternates of gradient descent. Each one differs in how much data we use to compute the gradient of the objective function. Thus subject to the amount of data, we make a trade-off between the accuracy of the parameter update and the time it takes to perform an update.

Batch Gradient Descent

It is also known as vanilla gradient descent. It calculates the gradient cost function in reference toθ parameter for the complete training dataset.

$$\theta = \theta - n.\nabla_\theta j(\theta) \tag{6.13}$$

As we need to calculate the gradient for the whole dataset to perform just one update, batch gradient descent can be very slow and is intractable for datasets that do not fit in memory. Batch gradient descent also does not allow us to update our model online, i.e. with new examples on-the-fly.

As per provided epochs in advance, the model has to calculate loss function's gradient for the entire dataset. All deep-learning frameworks provide an automatic function to calculate gradient in regards to the parameters.

Stochastic Gradient Descent

Stochastic gradient descent works differently from the previous one. It performs a parameter update for each training example $x^{(i)}$ and $y^{(i)}$ which is the label.

$$\theta = \theta - n.\nabla_\theta j\left(\theta.x^i; y^i\right) \tag{6.14}$$

For large datasets, the Batch gradient descent performs redundant computations as it re-computes gradients for similar examples before each parameter update. In the Stochastic approach this redundancy is removed by performing one update at a time. It is therefore usually much faster and can also be used to learn online. In the proposed Hybrid Model, the Stochastic gradient descent has been implemented to achieve better results.

Mini-Batch Gradient Descent

Mini-batch gradient descent performs an update for each mini-batch of n training samples.

$$\theta = \theta - n.\nabla_\theta j\left(\theta.x^{i;i+n}; y^{i;i+n}\right) \tag{6.15}$$

Mini-batch gradient descent reduces the variance for parameter update. Therefore, the model becomes more stable in terms of convergence. Itcan make use of highly optimized matrix optimizations common to state-of-the-art deep learning libraries that make computing the gradient w.r.t. a mini-batch very efficient. Common mini-batch sizes range between 50 and 256, but can vary for different applications. Mini-batch gradient descent is typically the algorithm of choice when training a neural network and the term SGD usually is employed also when mini-batches are used. There are many gradient descent optimization algorithms used by deep learning enthusiasts. The Nesterov accelerated gradient, Adagrad, Adadelta, RMS prop, Adam, AdaMax, etc. are some. Each has its own set of advantages and limitations. The following details have been referred from [Sebastian Ruder, ‘An overview of gradient descent optimization algorithms’, 2016].

Adagrad

Adagrad is an algorithm for gradient-based optimization that does just this: It adapts the learning rate to the parameters, performing smaller updates (i.e. low learning rates) for parameters associated with frequently occurring features, and larger updates (i.e. high learning rates) for parameters associated with infrequent features. For this reason, it is well-suited for dealing with sparse data.

Adadelta

Adadelta is an extension of Adagrad that seeks to reduce its aggressive, monotonically decreasing learning rate. Instead of accumulating all past squared gradients, Adadelta restricts the window of accumulated past gradients to some fixed size w. Instead of inefficiently storing w previous squared gradients, the sum of gradients is recursively defined as a decaying average of all past squared gradients.

RMS prop

RMSprop is adaptive learning method, which is proposed by Geoff Hinton (2012). RMSprop and Adadelta have been developed at the sametime when it resolves the problem of Adgrad's diminishing learning rates. In fact, it is identical to the first update vector of Adadelta.

$$E\left[g^2\right] = 0.9E\left[g^2\right]_{t-1} + 0.1g_t^2 \tag{6.16}$$

$$\theta_{t+1} = \theta_t - \frac{n}{\sqrt{E\left[g^2\right]_t + \in}} g_t \tag{6.17}$$

It divides the learning rate by an exponentially decaying average of squared gradients. Hinton recommends setting γ at 0.9. In addition, the learning rate value should be adjusted to η is 0.001.

Adam

The full form of Adam is Adaptive Moment Estimation (Adam). It calculates the adaptive learning rate of each parameter. In addition to storing an exponentially decaying average of past squared gradients v_t like Adadelta and RMSprop, Adam also keeps an exponentially decaying average of past gradients m_t, similar to momentum. Whereas momentum can be seen as a ball running down a slope, Adam behaves like a heavy ball with friction, which thus prefers flat minima in the error surface.

The computation of decaying averages of past and past squared gradients m_t and v_t are as follows:

$$m_t = \beta_1 m_{t-1} + \left(1-\beta_1\right) g_t \tag{6.18}$$

$$\upsilon_t = \beta_2 \upsilon_{t-1} + \left(1-\beta_2\right) g_t^2 \tag{6.19}$$

The m_t is the estimate of the first moment i.e. mean of the gradient. The v_t is the second moment that is the un-centered variance of the gradient. The m_t

and v_t, are both initialized as vectors of 0's for the vector. During the initial step, the author observesthat it is biased to zero. In addition, especially the decay rates are quite small such as β_0 and β_1 are closer to one.

They respond to biases by calculating bias-corrected first and second-moment approximations.

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t} \tag{6.20}$$

$$\hat{\upsilon}_t = \frac{\upsilon_t}{1-\beta_2^t} \tag{6.21}$$

In next step these values are used to updates the parameters as per Adadelta and RMSprop, which is shown below.

$$\theta_{t+1} = \theta_t - \frac{\eta}{\sqrt{\hat{\upsilon}_t} + \in} \hat{m}_t \tag{6.22}$$

The proposed rate of β1 is 0.9. Moreover, for β2 is 0.999 and $10^{-8}$for ϵ. As per the trials conducted, the result shows that Adam optimization algorithms work better for gradient descent.

Therefore, this method has been used in the proposed work.

6.6.4 Activation Function

As discussed earlier, an Artificial Neural Network (ANN) is an information processing paradigm that is inspired the brain. ANNs, like people, learn by example. An ANN is configured for a specific application, such as pattern recognition or data classification, through a learning process. Learning largely involves adjustments to the synaptic connections that exist between the neurons.

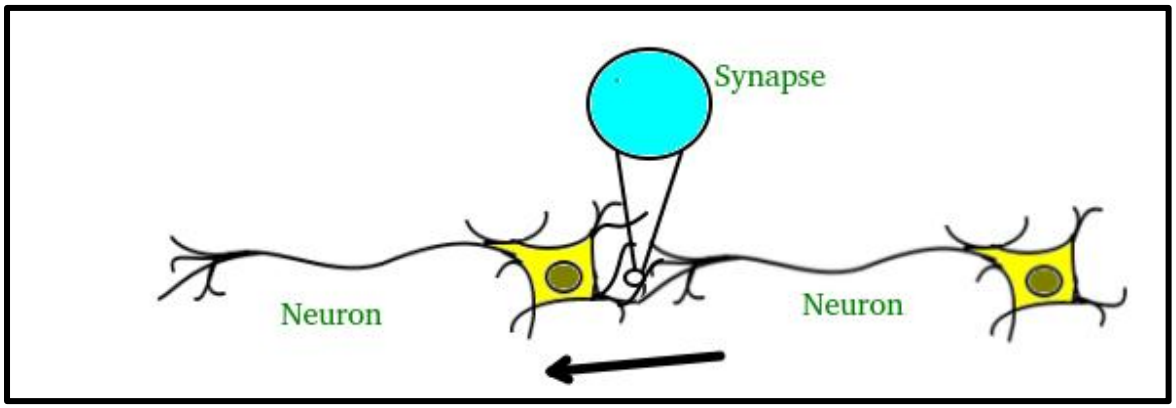


Figure 6-7 The ANN process

In the process of building a neural network, one of the choices you get to make is what activation function to use in the hidden layer as well as at the output layer of the network. There are a number of activation functions available. Few of them and the one that has been implemented are discussed below.

Identity Activation Function

The identity activation function is also known as linear activation. This function is a line and the output does not lie within any range.

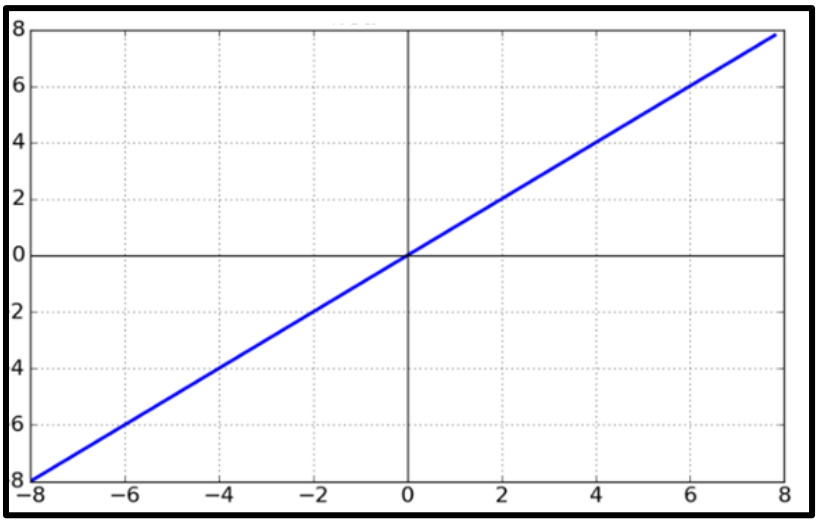


Figure 6-8 Linear Activation Function (Towards Data Science, 2019)

Equation : f(x) = x

Range : (-infinity to infinity)

It doesn't help with the complexity or various parameters of usual data that is fed to the neural networks.

Non-linear Activation Function

The Nonlinear Activation Functions are the most used activation functions. Nonlinearity helps to makes the graph look something like this

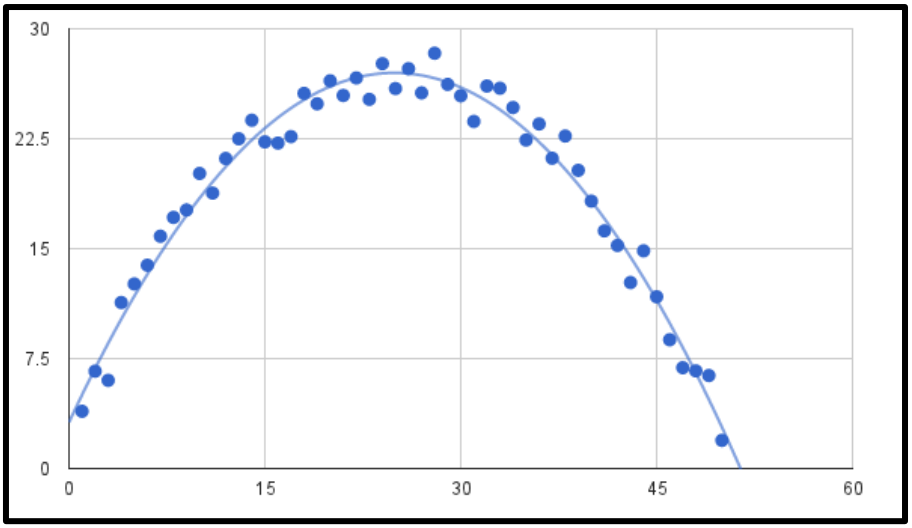


Figure 6-9 Non-Linear Activation Function (Towards Data Science, 2019)

It makes it easy for the model to generalize or adapt with variety of data and to differentiate between the outputs.

Sigmoid Activation Function

The sigmoid activation function is used where the data are not separated out linearly. In other words, a sigmoid function is used for nonlinearity. A neural network takes the input of linear grouping and uses the sigmoid function. It is quite popular for neural networks, where it satisfies the property of derivative and it is easy to calculate and perform.

$$\phi(z) = \frac{1}{1+e^{-z}} \tag{6.24}$$

The main reason is to use sigmoid function is that it predicts the result between 0 and 1. Therefore, it is specially used when the model predicts the probability of the output layer. As probability range remains between 0 and 1, so it is a good choice for neural networks.

The sigmoid function generates an ‘S’ shaped curve.

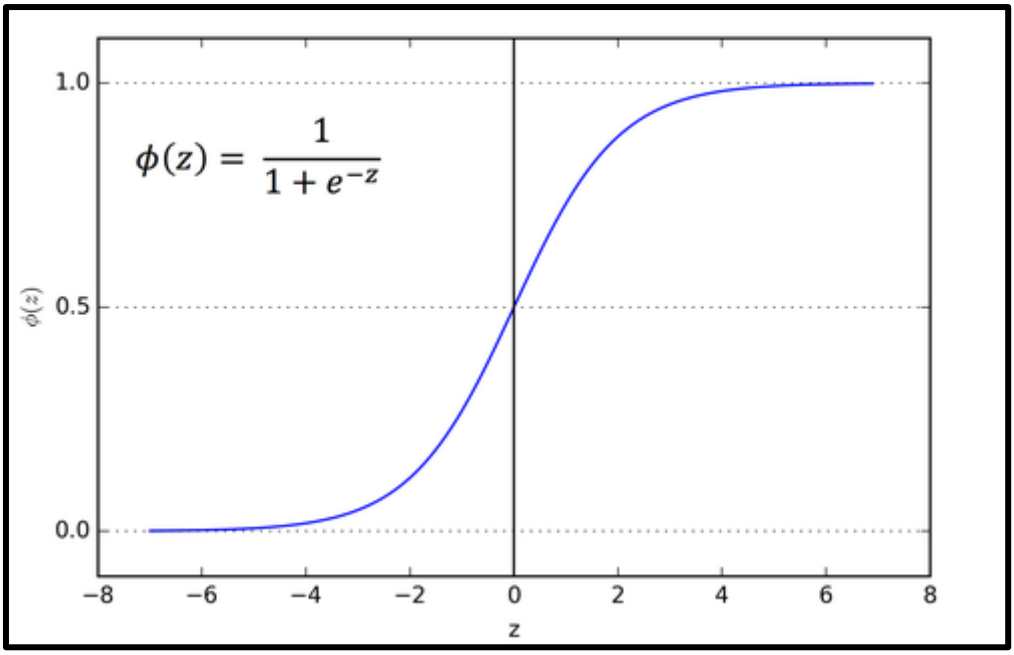


Figure 6-10 Sigmoid activation function (Learning and R), 2018

There are a few drawbacks of function. It creates a vanishing gradient problem; another problem is that it makes optimization tougher because it creates updates for the gradient is too far in another way. In addition, it has a slow convergence.

Hyperbolic Tangent Function

This activation function is also known as Tanh activation function. It is defined as.

$$f(x) = \tanh(x) = 2/(1+e^{-2x}) - 1 \qquad (6.25)$$

It works better than sigmoid function. The range of the function is -1 to 1, so hidden layer derives at 0 or near to it. In other words, the meaning of data comes to around 0. This function is differentiable, non-linear and monotonic. This function takes the approach feed-forward network.

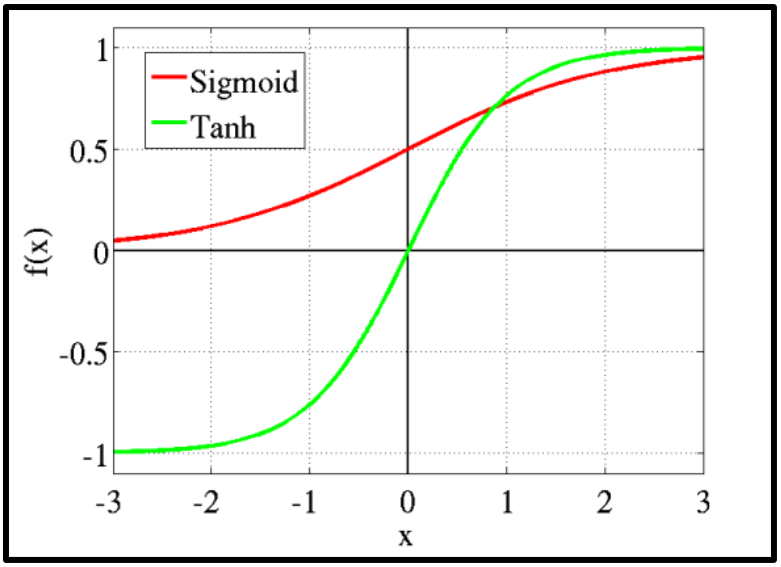


Figure 6-11 Hyperbolic tangent function (Learning and R), 2018)

ReLU Activation Function

ReLU stands for the Rectified Linear Unit. It is widely used to activate at hidden layer of neural network.

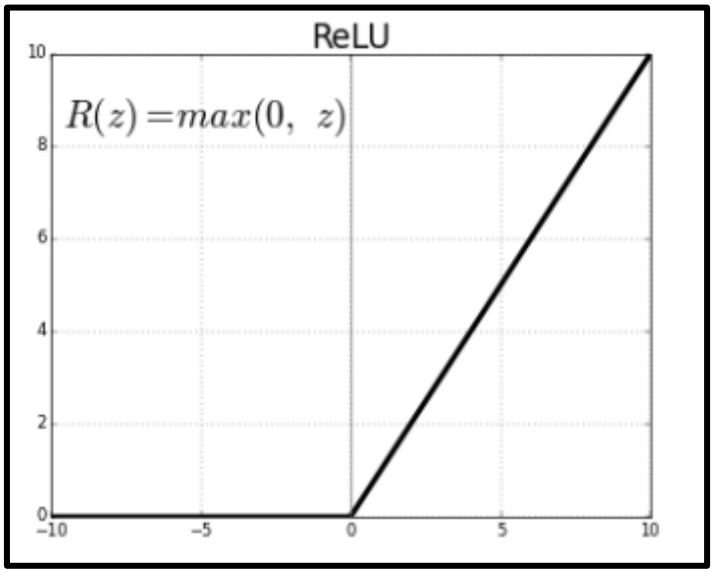


Figure 6-12 ReLU activation function (Learning and R), 2018

As we can from Figure 6.12. This activation function is half rectified at the bottom. When Z is less zero, f(z) is converted to zero. It has range 0 to infinity, instead of -1 to 1 like sigmoid function. This function and derivative both are monotonic.

Sigmoid and Tanh are common activation functions that are used in neural networks. Glorot et al (2011) demonstrate that ReLU provides faster and better performance on high-dimensional data for neural networks. Relu calculates the

function $f(x) = \max(0, x)$ and sets the threshold for input matrix to zero. One of the major advantages of ReLU over sigmoid and Tanh, is that, it does not require higher computational for comparison and multiplication. In addition, in ReLU, there is no saturation means that it has capable backpropagation without vanishing gradient, which makes the proper selection of neural network with deep layers. Considering all advantages of ReLU, which has abilities y to generate sparse activation, it is considered for optimizing deep MLP and CNN.

ReLU has one disadvantage; ReLU unit suffers from a possible problem during training, and the problem is dying neurons. The few neurons never fire from a particular point because when no gradient flows backward through the ReLU unit. When a model is selected at ahigher learning rate, a large number of neurons go into the dead state. To overcome this problem, set small learning rates for weight updates in gradient descent.

Another technique is that apply leaky ReLU, which add small negative slope into that unit, which is not active.

In the proposed model, the ReLU activation function has been applied at the input layer and hidden layer, whereas sigmoid activation function has been applied at output level.

6.7 Experimental Setup and Result Analysis

The proposed extractive summarization method is implemented on Opinosis (Kavita Ganesan, 2010) dataset. In this section, we have a discussion of the comparative analysis of existing algorithms.

6.7.1 Dataset Description

The proposed extraction based model was evaluated on standard datasets from Opinosis. This dataset is a standard dataset created by Dr. Kavita Ganesan who is a Data Scientist working especially in the field of Text Datamining. She is a Data Scientist with expertise in Natural Language Processing, Text Mining, Search and Machine Learning. Over the last decade, she has worked for various technology organizations including GitHub (Microsoft), 3M Health Information Systems and eBay. She has done work on extracting insights from data, recommendation systems, sentiment analysis, text classification, search enhancements and text summarization.

She received her Ph.D. in Computer Science with a focus on Text Mining, Machine Learning and Search from the University of Illinois at Urbana Champaign. She has authored over ten first author papers at top tier Data Mining and NLP venues such as WWW, COLING, NAACL, IEEE Big Data and Information Retrieval Journal.

The Opinosis corpus has a collection of total 51 documents, which are collected from different sources. The dataset contains extracted sentences from reviews of users on a particular topic. The topics are like “Performance of Toyota Camry”, “Battery Life Amazon Kindle”, “Features of Windows7”, “Food at Holiday Inn London” etc. There is approximately an average of 100 sentences in each document. The reviews have been obtained from various resources such as Tripadvisor for Hotels, Edmunds for car and Amazon.com for various electronic devices. This dataset contains gold standard summaries which are the human summaries for each topic. Every document / topic has 4 gold summaries which are prepared by humans.

Apart from the Opinosis dataset, the DUC 2004 dataset has also been used to evaluate this model. The DUC 2004 dataset has been discussed in detail in Chapter 5.

Different evaluation methods are used to check the performance of the extractive summarization techniques and superiority of prepared summaries. There are many techniques to assess the quality of text summarization system (Jing et al., 1998; Neto et al., 2000; Santos et al., 2004). There are two types of approaches for evaluating summaries, which is automatically generated: intrinsic and extrinsic. In intrinsic, the sentence summary is evaluated based on the content summary’s analysis, whereas in extrinsic summary, summarization quality is checked based on task-based, also, it checks the usefulness.

In this research, Recall-Oriented Understudy for Gisting Evaluation (i.e. ROUGE) toolkit has been used to compare the human and system generated summaries. ROUGE 2.0 is an easy to use evaluation toolkit for Automatic Summarization tasks. It uses the ROUGE system of metrics which works by comparing an automatically produced summary or translation against a set of reference summaries (typically human-produced). ROUGE is one of the standard ways to compute effectiveness of auto generated summaries. The proposed model is compared with Latent Semantic Analysis based text summarization

system and MSWord Auto Summary tool using the ROUGE toolkit. The latest version of ROUGE 2.0 supports the following:

- Evaluation of ROUGE-N (unigram, bigrams, trigrams, etc)
- Evaluation of ROUGE-L (summary level LCS)
- Evaluation of ROUGE-S and ROUGE-SU (skip-gram and skip-gram with unigrams)
- Evaluation of multiple ROUGE metrics at one go
- Stemming for different languages
- Stopword removal with customizable stop words
- Evaluation of unicode texts (e.g. Persian)
- Minimal formatting requirements for system and reference summaries
- Output in CSV – this makes it super easy for score analysis
- Full documentation and support via GitHub Issues

The rouge evaluation toolkit generates three types of the parameter for each metric such as average recall, average precision, and average f-score/measure.

Precision, Recall & F-Score

These parameters are discussed in Chapter – 3.

As discussed earlier, these measureshelp us to understand that system extracted summaries are how close to human summaries.

6.7.2 Result Analysis

After completing pre-processing, the summaries generated by the proposed model are compared with those generated by MSWord and Latent Semantic Analysis Summarizer. Sample summaries are given in Appendix-B

The metrics used for comparison are ROUGE-1, ROUGE-2, ROUGE-SU4 and ROUGE-L for recall, precision and F-score.

The values of the metrics obtained after applying on various summarization tools on the Opinosis dataset for average recall, average precision, and average F-score results are presented below.

Table 6-1 Comparison of various summarization tools for average recall using ROUGE-1, ROUGE-2, ROUGE-SU4 and ROUGE-L at 95% interval confidence

	ROUGE -1	ROUGE -2	ROUGE-SU4	ROUGE-L
PROPOSED	0.4	0.13	0.21	0.26
LSA	0.37	0.11	0.19	0.31
MSWORD	0.14	0.03	0.05	0.08

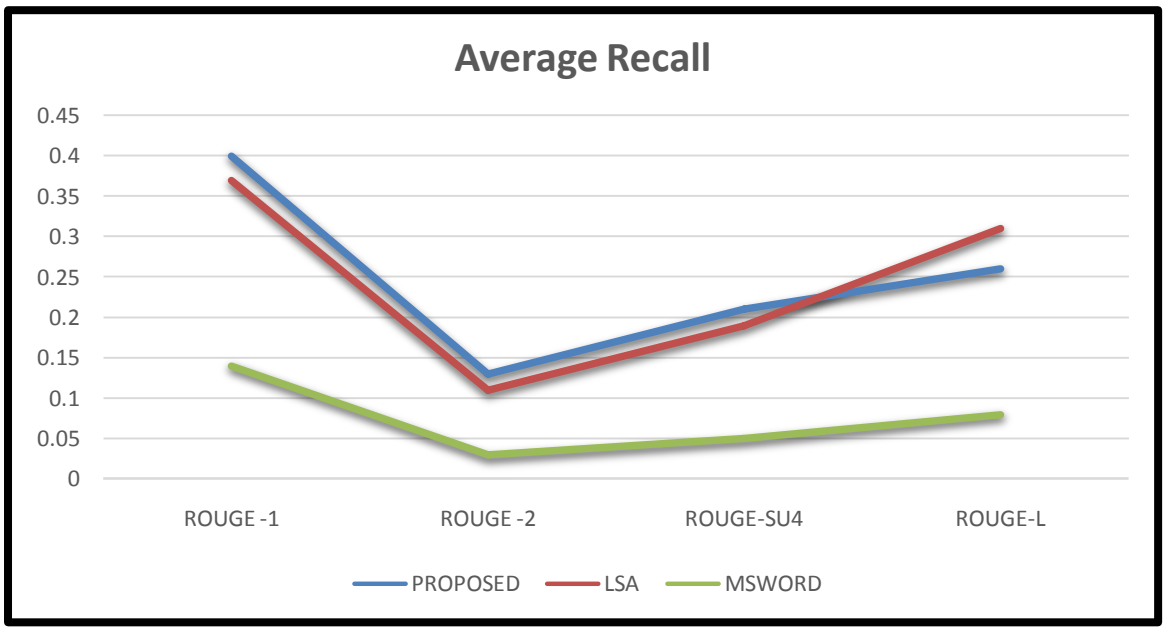


Figure 6-13 Comparison chart for average recall obtained from different summarization tools

Table 6-2 List of all various summarization tools in descending order of average recall for ROUGE-1, ROUGE-2, ROUGE-SU4 and ROUGE-L at 95% confidence interval

ROUGE -1	ROUGE -2	ROUGE-SU4	ROUGE-L
PROPOSED	PROPOSED	PROPOSED	LSA
LSA	LSA	LSA	PROPOSED
MSWORD	MSWORD	MSWORD	MSWORD

Table 6-3 Comparison of various summarization tools for average precision using ROUGE-1, ROUGE-2, ROUGE-SU4 and ROUGE-L at 95% interval confidence

	ROUGE -1	ROUGE -2	ROUGE-SU4	ROUGE-L
PROPOSED	0.35	0.11	0.18	0.22
LSA	0.2	0.05	0.08	0.13
MSWORD	0.52	0.14	0.29	0.41

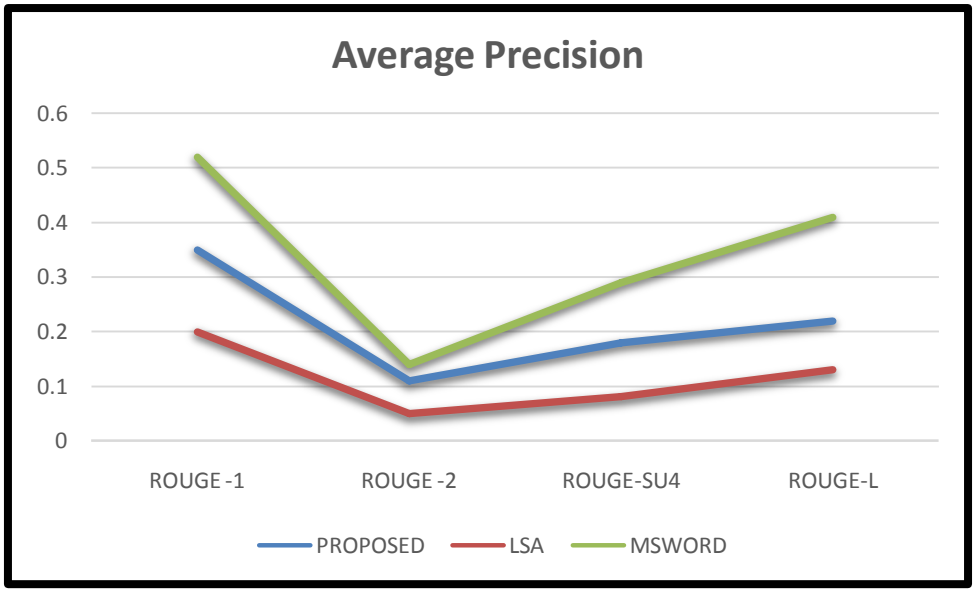


Figure 6-14 Comparison chart for average precision obtained from different summarization tools

Table 6-4 List of all various summarization tools in descending order of average precision for ROUGE-1, ROUGE-2, ROUGE-SU4 and ROUGE-L at 95% confidence interval

ROUGE -1	ROUGE -2	ROUGE-SU4	ROUGE-L
MSWORD	MSWORD	MSWORD	MSWORD
PROPOSED	PROPOSED	PROPOSED	PROPOSED
LSA	LSA	LSA	LSA

Table 6-5 Comparison of various summarization tool for average F-score using ROUGE-1, ROUGE-2, ROUGE-SU4 and ROUGE-L at 95% interval confidence

	ROUGE -1	ROUGE -2	ROUGE-SU4	ROUGE-L
PROPOSED	0.4	0.13	0.21	0.26
LSA	0.37	0.11	0.19	0.31
MSWORD	0.14	0.03	0.05	0.08

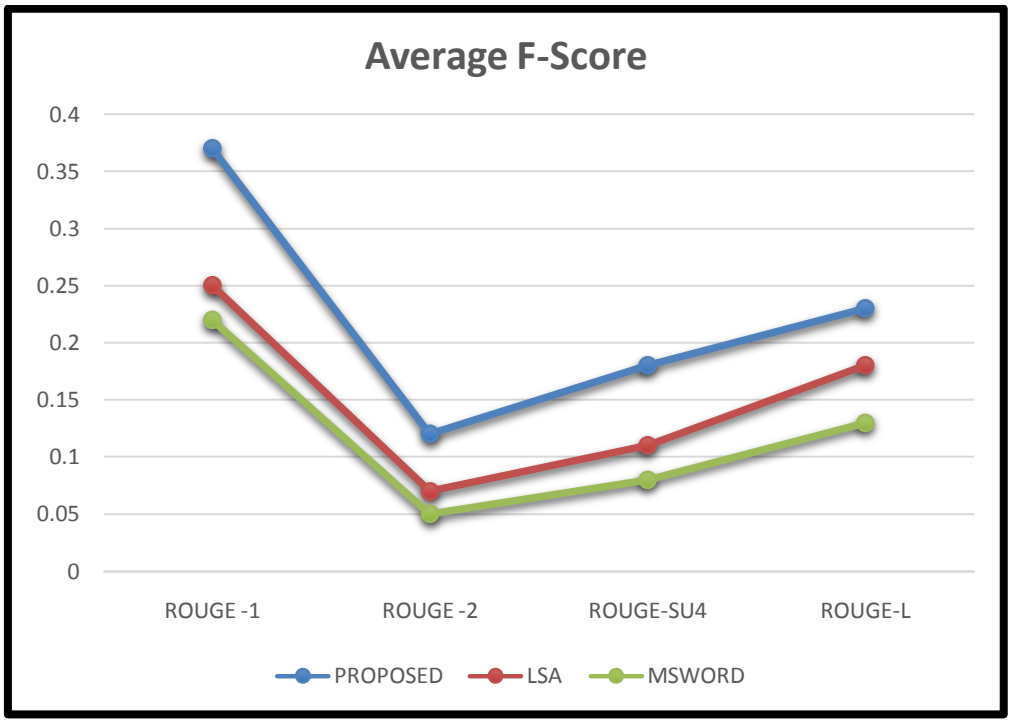


Figure 6-15 Comparison chart of average F-score obtained from different summarization tools

Table 6-6 List of all various summarization tool in descending order of average F-score for ROUGE-1, ROUGE-2, ROUGE-SU4 and ROUGE-L at 95% confidence interval

ROUGE -1	ROUGE -2	ROUGE-SU4	ROUGE-L
PROPOSED	PROPOSED	PROPOSED	PROPOSED
LSA	LSA	LSA	LSA
MSWORD	MSWORD	MSWORD	MSWORD

Table 6.1 and 6.2 display the result obtained for average recall for ROUGE-1, ROUGE-2, ROUGE-SU4 and ROUGE-L. It is evident that the proposed model for Text Summarization performs better as compared to other methods for all except ROUGE-L. ROUGE-L shows better result for LSA summarizer as compared to MSWORD and the proposed one. The reason being the length of the summary generated by each method is different and the human summary provided in the corpus, and the summary which is obtained from MSWord Summarizer may not purely be based on extraction.

In addition, while preparing the summary of the original document, it is possible that a sentence has an overlapped sentence and repeated sentences also. Our proposed method Summarizer includes the selected original sentence from the input document without repetition.

Table 6.3 and 6.4 display the results that have been obtained for average precision. Over here the proposed model is below the MSWord Summarizer. As discussed earlier, it may because the MSWord summary is not based on pure extraction. That may reason to perform better for average precision.

The values of the harmonic mean i.e. the F-score which is the perfect comparison metric, is shown in Table 6.5 and Table 6.6. It is clearly observed

that the proposed model gives far better score as compared to MSWORD and LSA.

The summary obtained from our proposed method, MSWord Summarizer and LSA Summarizer have been compared to human summaries for evaluation. The dataset corpus has a wide range of subjects such as hotel, electronics etc. For all the different types of documents the proposed model has shown a better generated summary. This model works well with small (approx. 50 words) as well as large (approx. 500 words at least) documents.

DUC 2004 Dataset Results:

ROUGE – N (N=1 & 2), ROUGE-SU4 and ROUGE-L scores obtained after applying on various summarization tools to the DUC 2004 dataset for average recall, average precision, and average f-score results are presented below.

Table 6-7 Comparison of various summarization tools for average recall using ROUGE-1, ROUGE-2, ROUGE-SU4 and ROUGE-L at 95% interval confidence

	ROUGE -1	ROUGE -2	ROUGE-SU4	ROUGE-L
PROPOSED	0.23	0.051	0.07	0.19
FreqSum	0.17	0.02	0.05	0.13
LexRank	0.24	0.06	0.09	0.18

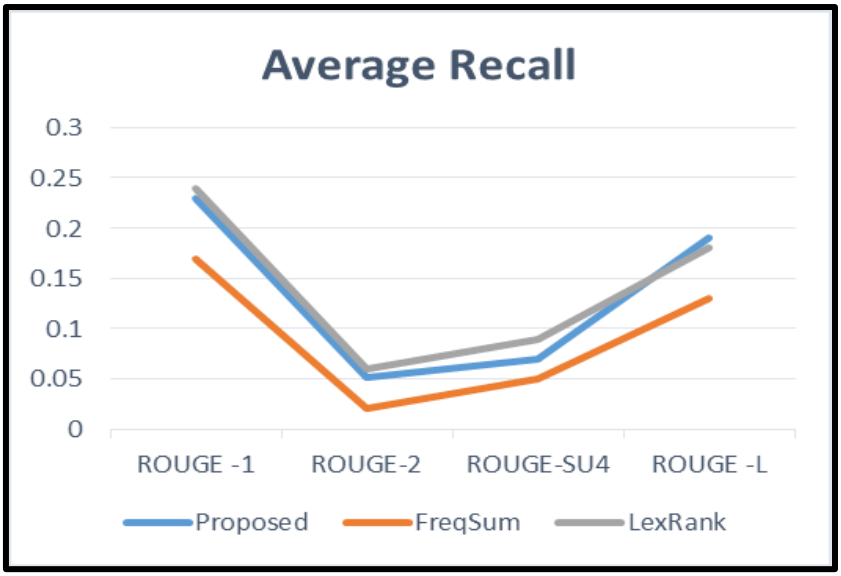


Figure 6-16 Comparison chart for average recall obtained from different summarization tools

Table 6-8 Comparison of various summarization tools for average precision using ROUGE-1, ROUGE-2, ROUGE-SU4 and ROUGE-L at 95% interval confidence

	ROUGE -1	ROUGE -2	ROUGE-SU4	ROUGE-L
PROPOSED	0.32	0.2	0.17	0.22
FreqSum	0.25	0.06	0.09	0.18
LexRank	0.28	0.07	0.09	0.2

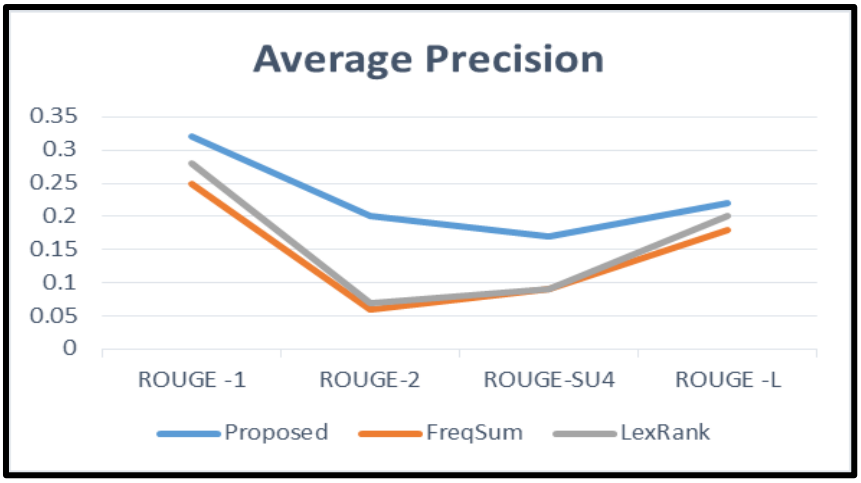


Figure 6-17 Comparison chart for average precision obtained from different summarization tools

Table 6-9 Comparison of various summarization tool for average F-score using

	ROUGE -1	ROUGE -2	ROUGE-SU4	ROUGE-L
PROPOSED	0.22	0.06	0.1	0.19
FreqSum	0.18	0.02	0.05	0.14
LexRank	0.24	0.06	0.08	0.18

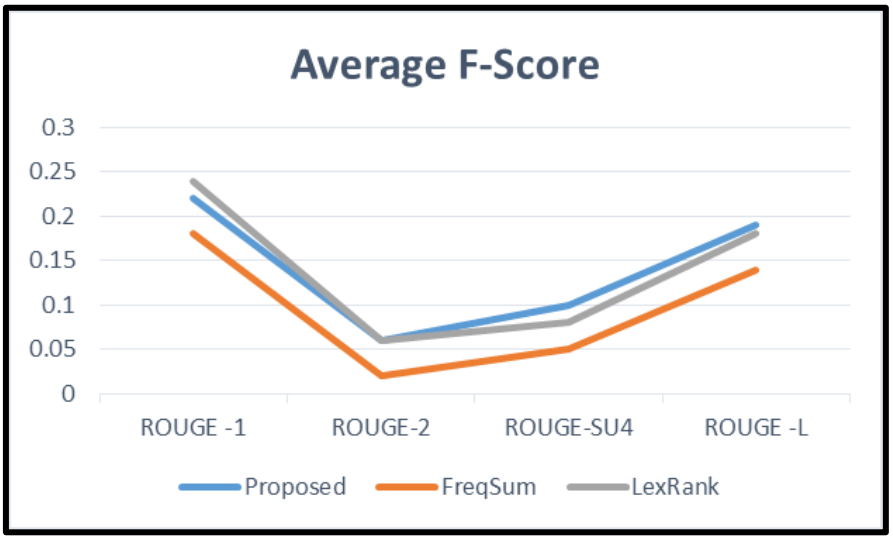


Figure 6-18 Comparison chart for average F-score obtained from different summarization tools

As can be seen by the details given in Table 6.7 to 6.9, the proposed model works well for the DUC 2004 datasets as well. Average recall is almost as good as the LexRank one, whereas the average precision is much better that the other two. The F-score vales show a very good output for the proposed model.

Comparison of Results of Datasets:

Table 6-10 Comparison of datasets

	DUC - 2004			Opinosis		
	Proposed	**FreqSum**	**LexRank**	**Proposed**	**LSA**	**MSWord**
ROUGE – 1	0.22	0.18	0.24	0.3703	0.2598	0.2225
ROUGE – 2	0.06	0.02	0.06	0.123	0.0706	0.0572
ROUGE - SU4	0.1	0.05	0.08	0.1889	0.1109	0.0872
ROUGE - L	0.19	0.14	0.18	0.2345	0.181	0.137

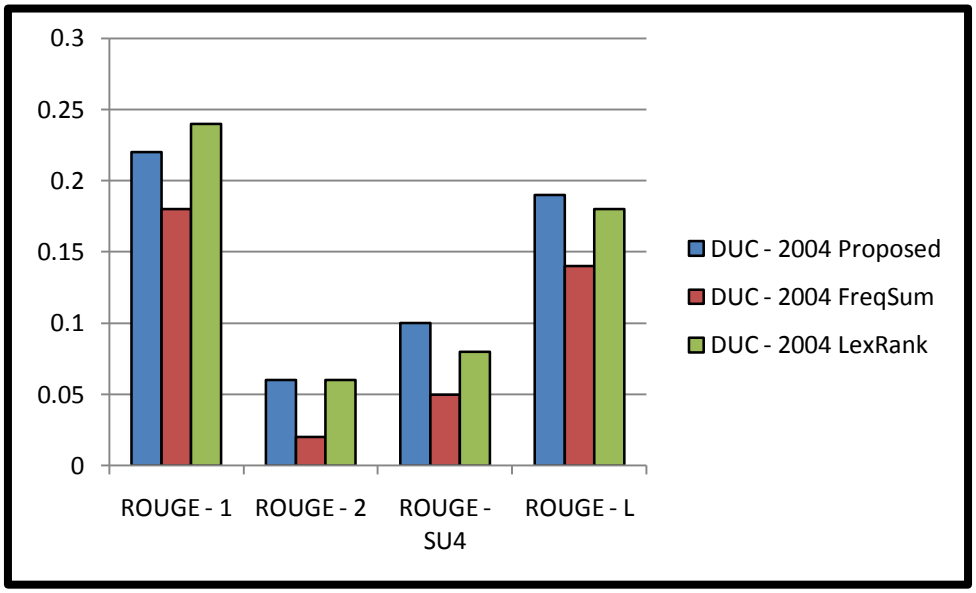


Figure 6-19 DUC 2004 Dataset Comparison

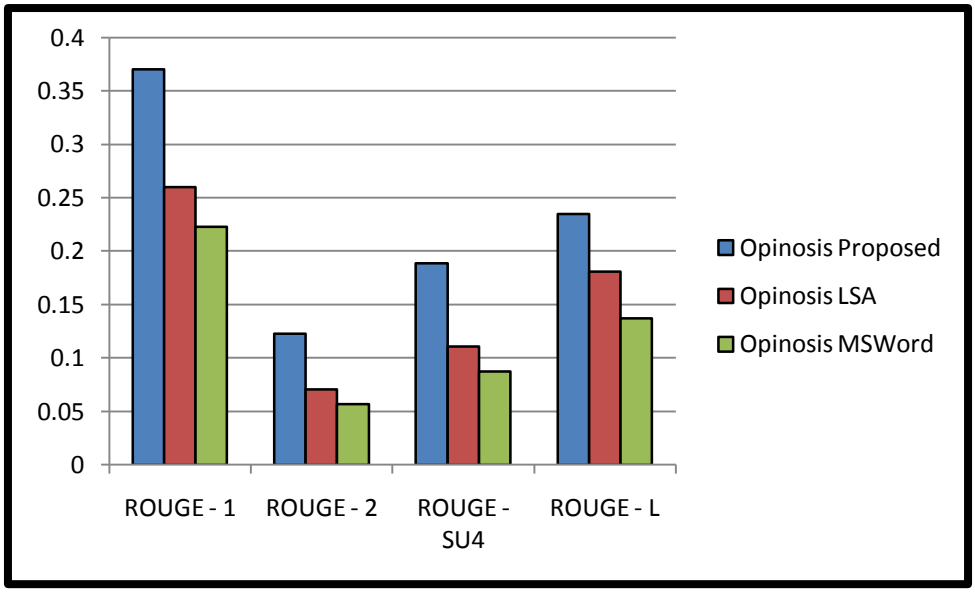


Figure 6-20 Opinosis Dataset Comparison

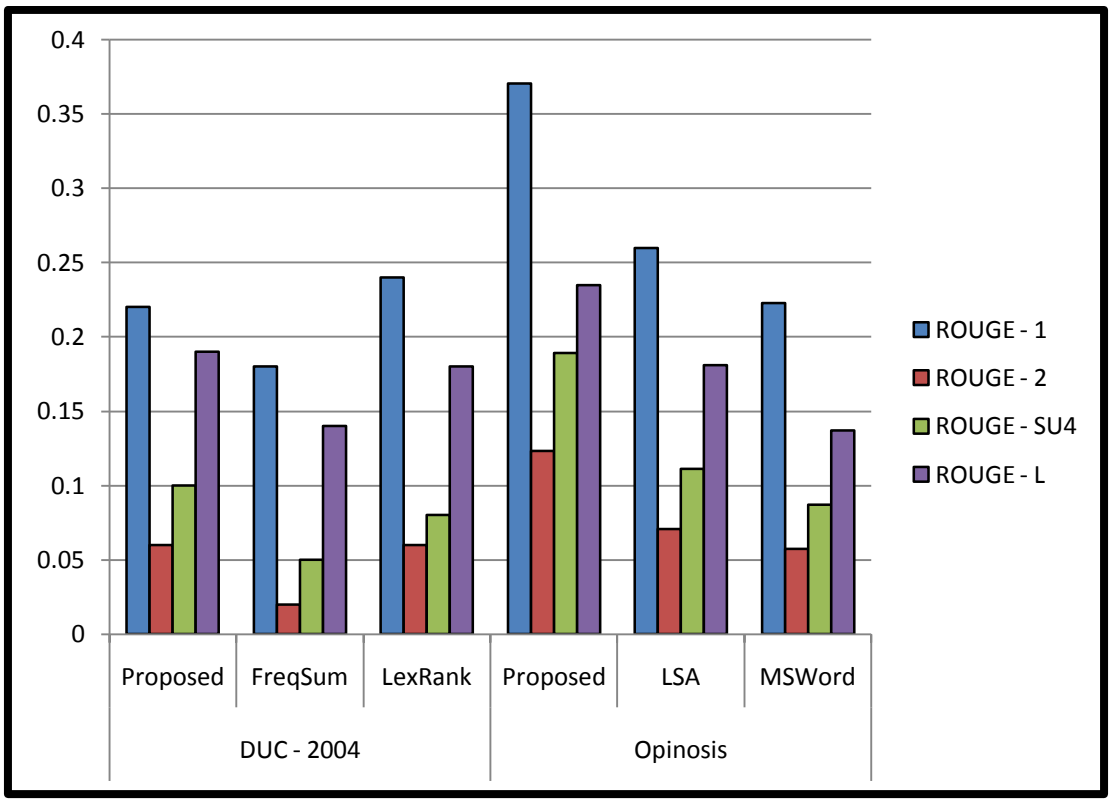


Figure 6-21 DUC 2004 and Opinosis Datasets combined comparison

The Tables 6.10 and the Figure 6.19 to 6.21 show the comparative results of the proposed model on both the Opinosis and the DUC 2004 datasets. It is quite evident that the proposed model works extremely well as compared to the inbuilt tools.

SUMMARY

In this chapter the Hybrid model was discussed and implemented. This model generated a better summary for the datasets as compared to the existing models wherein the comparison was made using the Rouge toolkit. This model works well with large documents i.e. at least 50 sentences having approximately 500 words. The work done in this chapter has been published in the Springer series.