Personalized E-news Recommendation System

Group name: Astraeas

Index Number	Name
134006J	Alwis E.V.K.
134012A	Asanga M.W.L.
134028D	Dandeniya D.
134040G	Dissanayaka L.G.A.N.

Faculty of Information Technology

University of Moratuwa

July 2018

Personalized E-news Recommendation System

Group name: Astraeas

Index Number	Name
134006J	Alwis E.V.K.
134012A	Asanga M.W.L.
134028D	Dandeniya D.
134040G	Dissanayaka L.G.A.N.

Dissertation submitted to the Faculty of Information Technology, University of Moratuwa, Sri Lanka for the partial fulfillment of the requirements of the Honours Degree of Bachelor of Science in Information Technology.

July 2018

Declaration

We declare that this thesis is our own work and has not been submitted in any form for another degree or diploma at any university or other institution of tertiary education. Information derived from the published or unpublished work of others has been acknowledged in the text and a list of references is given.

Name of Student — Signature of Student

Alwis E.V.K.

Asanga M.W.L.

Dandeniya D.

Dissanayaka L.G.A.N.

Date: ……

Name of the Supervisor — Signature of the Supervisor

Dr. (Mrs.) A.T.P. Silva ……..

Date: ……

Acknowledgement

We would like to acknowledge and extend our sincere gratitude to the following people who have made the completion of this project possible. We are thankful to Mr. P. M. Karunarathne, dean of the Faculty of Information Technology for his enormous encouragement. We are thankful to our supervisor Dr. (Mrs.) A.T.P. Silva for her encouragement, much needed motivation, guidance and coordination as our project supervisor. We are also grateful to Dr. Lochandaka Ranathunga, the head of the Information Technology Department, Dr. Subha Fernando, the head of the Computational Mathematics Department and Mrs. Sumudu Wijetunge, the head of the Interdisciplinary Studies Department. Finally a warm thank you extends towards all other academic staff for the help and inspiration they extended, all non-academic staff members who helped us throughout this project, our faculty members, batch mates who motivated and supported us and to our families and friends.

Abstract

Information overloading is one of the major problems exists in the e-news domain with the vast amount of content added by thousands of e-news web portals daily. So it's very difficult to read all the news articles and find the relevant information since it takes lot of time as well as human effort. Hence, the project aims to design and develop a "Personalized E-news Recommendation System". There are e-news recommendations systems already developed by Google and Yahoo. But the major limitations of these existing systems include that the e-news recommendations by these systems are not personalized and they do not a summarized view of the e-news content. Our system addresses these research gaps prevailing in the existing systems and provide personalized news recommendations and a summarized view on the e-news content. The system consists of four main modules namely e-news extraction and classification, aggregation, summarization and recommendation. The system extracts e-news articles from a pre-defined set of e-news sites and these articles are pre-processed using several techniques. Then the extracted news items are classified in to different categories like political, business, sports, technology, entertainment and other using an ensemble classifier. Then those classified e-news articles from multiple sources are aggregated in to different clusters where a cluster contains e-news articles about the same topic. Then the summarization module generates a single summary representing the key information of the news articles within a cluster. The system generates extractive summaries for each cluster by extracting significant information from original documents in the clusters themselves based on a hybrid model. Then the lexical, syntactical and semantic redundancies are removed and the final summary is created after arranging the final summary sentences in the proper coherent order. The recommendation module gives personalized e-news recommendations for different users of the system. The recommendation module also uses a hybrid recommendation model using content based filtering which checks the user's past click events, collaborative filtering which recommends e-news articles by identifying similar type of user groups based on the user profiles and Location aware personalization which recommends e-news articles based on the user's current location. We present the personalized e-news recommendation system which reduces the user's reading time and effort to a great extent.

Contents

List of Figures

List of Tables

Chapter 1

1. Introduction

1.0 Introduction

With the advent of the information age, people are more towards on browsing online news sites rather than traditional ways of news consumption via printed media like newspapers. Today online news reading has become very popular since web provides access to news articles from millions of sources all around the world. These e-news web sites are generating thousands of news per day. Therefore, a critical problem with news service websites is that the volume of articles can be overwhelming to the users. This problem refers to the "information overloading". So, managing this kind of vast number of news articles has become a challenging task. Therefore, providing a news articles categorization engine is a timely requirement [1]. This may classify diverse news articles in to different classes like political news, financial news, sports news, entertainment news, technological news etc. Then it will be very convenient for the users to access similar kind of news in a single place [2].

But simply classifying the news articles is not sufficient because there is a huge number of news articles describing about the same news in diverse sources. So, bringing all the news articles which describe about the same news into one place is also really important. If the news articles about the same news from different sources can be integrated together then they can be accessed from a single place which is very convenient for the users. Aggregating of these news articles based on similarity of the news content will give a better user experience [3]. Even today there are so many news aggregation web sites have been developed like google news, yahoo news to collect news from various sources and to provide an aggregate view of news from around the world [4]. Although there are available news aggregation systems this has become a hot research topic because researchers are looking towards most accurate news aggregation systems. News aggregators capably handle the large amount of news that is published nowadays. By using aggregators consumers can reduce search costs and terminate their search for content that they would seek.

But still people need to go through at least a certain number of news articles one by one to get to know more details about a particular news item. That is because different news sites describe about the same news in different ways, different sites reveal different perspectives on the same topic and some news sites provide some additional information about particular news items [5]. People may have no much time to read everything and it's difficult to make critical decisions based on whatever information is available. So, if these news web sites can provide a summarized view of the aggregated news items with most important details of a particular news item then the users can save lot of time. Therefore, it has gained much more attention towards news summarization. The summarization involves distilling the most important information from a source (or sources) to produce an abridged version for a particular task [6]. These systems focus on identifying and presenting important, common information in news. Then it helps users gain a broad and diverse understanding about the news items by presenting various perspectives on the same news topic.

Another key challenge of news websites is to help users find the articles that they are interesting to read. So, the requirement for a news recommendation engine has gained a big boom. Lot of theories also have been identified and introduced in this field acquiring the attention of potential researches to research more to provide the users with news articles that are interesting for them to read. People hope to obtain their interested news fast and get pleasant reading experience. Under these circumstances the need of a personalized news recommendation system appears which better meets the user's individual needs of news. For users who are logged in and have explicitly enabled web history, the recommendation system builds profiles of users' news interests based on their past click behavior [7]. Through the analysis of user's interests, we can analyze the personalized news recommendation for different people. The challenge here is we need to analyze the user's interests in a time tagged manner as the user's interests may also change over the time [8]. So, in order to address all the above-mentioned issues, we are going to develop an automated and intelligent system of classified, aggregated, summarized and personalized news recommendations.

1.1 Background & Motivation

With the popularity of the internet, the information content available and produced daily on different e-news portals has increased at an enormous rate. These news sites are generating thousands of news from political news, financial news, sports news etc. even in a single second [2]. But most of the people are not interested at reading all these types of news where some people may interested at political news, business people may want to read financial news and young generation may want to read sports news or entertainment news likewise. But if all of the news items are jumbled together then it will be very difficult to find the news items they want to read. It will consume lot of time as well as a lot of human effort. Instead if the news items can be classified into different groups of political, financial, sports, and entertainment separately then people will feel it very easy to find the type of news categories they want to read. So it was the main intuition behind having a classified e-news system.

Although the news items are categorized into classes still there will be large number of news articles about the same topic generated by diverse sources. So in order to have a proper understanding about the news people need to visit those e-news sites one by one and search for the particular news which also consumes lot of time [4]. So instead if the system can aggregate all the news articles which are describing about the same news from different sources into a single place the problem can be solved which was the insight for an e-news clustering system.

Within a single cluster of news articles a large number of news articles about a particular topic will be available. So the readers have to go through each and every news article to get a clear and detailed understanding about that particular news. So the problem will be the same which is the wastage of time and man power. With this main intuition we are going to propose a system which can summarize all the news articles describing about a same news and present the most important details about that news to the reader [5]. They the readers can get a proper understanding about the news items by reading those summaries which does not take that much of time and effort.

People are also interested in obtaining their preferred types of news items fast and thereby having a pleasant reading experience. Therefore, if the system can provide personalized news recommendations for the users then it will enhance the user

friendliness of the system as well. So the main intuition and motivation behind a news recommendation system was to enhance the user experience.

1.3 Problem in Brief

The information overloading is one of the serious problems nowadays since information is generating in a rapid rate with the advent of internet. When it comes to the e-news context this problem remains the same. There is enormous volume of news articles from numerous portals on the web [2]. These contain gigantic amount of news articles from all around the world added daily at a rate of hundreds, even thousands or more per hour. This prohibits a difficulty for the access to the right information and users must spend a lot of time manually sifting out useful or relevant information.

With the busy life styles of people there's a trend where people are mostly referring to e-news sites to know day to day incidents happening around the world. With its convenience more and more people prefer to read news online instead of reading the paper-format press releases. Although more and more information from around the world is available online and often at no cost, many news readers only consult a small subset of news sources. Reasons include the overwhelming number of sources where there is an enormous amount of news items from varied e-news web sites, language barriers, or simply habit. It's time consuming to refer all those news items one by one in these numerous e-news web sites. In addition, when people are interested in a certain news item and wants to know important details of that news item they might need to refer number of e-news web sites which is also time consuming. Therefore rather reading the entire detailed news article if a summary of the related news item can be displayed it will be convenient for the users [4]. With the vast domain of information available in numerous sources, people hope to obtain their interested news fast and have pleasant reading experience. The classical solution usually used to solve the information overloading is a recommendation, especially personalized recommendation [7]. A challenging problem is how to efficiently select specific news articles from a large corpus of newly-published press releases to recommend to individual readers, where the selected news items should match the reader's reading preference as much as possible. Therefore it's a timely requirement to develop a system

of classified, aggregated, summarized and personalized news items gathered from varied e-news portals.

1.4 Aim & Objectives

1.4.1 Aim

To develop a system to classify, aggregate e-news articles gathered from varied e-news portals and generate a separate summary for each e-news cluster from the aggregation phase and further provide personalized e-news recommendations.

1.4.2 Objectives

- Review the literature on e-news classification, aggregation, summarization and recommendation systems.
- Design and develop a module to extract text information from web news pages and classify the extracted news items as sports news, political news, financial news etc...
- Design and develop a module to cluster the similar news articles gathered from different e-news web sites together.
- Design and develop a module to generate a separate summary for each e-news cluster.
- Design and develop a module to give personalized news recommendations by tracking user behavior patterns.
- Evaluate the proposed system.

1.5 Conclusion

The problem of information overloading in the e-news context has lead people to consume lot of time and effort in finding what they exactly want. When presenting a solution to this problem with an e-news classification, aggregation, summarization and recommendation system there exists a number of approaches used by various researchers. Through this research project our main concern was to identify the most efficient, accurate and suitable approaches to address this problem.

Chapter 2

2. Literature Review

2.1 Introduction

In this chapter we mainly focus on analysing the similar approaches to solve the above mentioned problem. This chapter compares the other's work along with our solution and brings out the importance and uniqueness of our project. This chapter analyses the features and pitfalls of the existing approaches in detail. We identify the gaps in currently used approaches and design our system to address the identified pitfalls of the existing systems.

2.2 Automatic e-News Extraction

There are two stages in the extraction process. First, the e-news websites are crawled to gather e-news pages. Then e-news article contents are extracted from e-news web pages. The e-news websites consist of different types of web pages such as advertisement pages, blog pages, shopping pages, etc. Therefore extracting only e-news item is a challenging task. Most of the time e-news sites are not static, they change their layout dynamically over the time.

2.2.1 DOM Tree Exploration

Gupta et al. [9] and Mukherjee et al. [10] discover the idea of using a DOM tree-based method. Gupta [9] explain Crunch, it is a content extraction tool which delivers a set of customizable filters to reduce the clutter from the web page. They optimize the link to text ratio in link list remover which removes nodes with a high link to text ratio. Using only the text-to-link ratio yields particularly low recall, and this approach is unsatisfactory to extract the e-news articles from e-news webpages. Crunch differs from the CoreEx approach in two key aspects. First, it is intended as a common tool for pages from diverse domains, and is not only news focused. Second, it uses to interact with a human and is not a fully automated system. An extension of their work [11] attempts to automatically classify a Web site and use earlier adjusted settings for extraction. Mukherjee et al. [10] have developed a system to automatically annotate the content of the e-news Web sites considering semantic analysis structural of the DOM

nodes. They partition the HTML by structural analysis. The partitions are assigned semantic labels by a prefixed ontology and lexical associations with the support of WordNet. They achieve 100% recall and precision for 35 e-news article pages from 8 e-news websites their precision and recall are over concept instances, and not actual content blocks as in our system. A news article page has only one instance of the concept of detailed e-news, which their system extracts perfectly.

2.2.2 Scoring Blocks

Lin and Ho [12], Yin and Lee [13], and Tseng and Kao [14] introduce approaches that split an e-news web page into blocks and score them to identify their importance. Lin and Ho [12] recognize informative content blocks by calculating the entropy values of terms in a block consider on their occurrence in an earlier seen set of web pages from the same e-news web site. But there are restricted to pages that use an HTML <table> layout. Since the entropy value is calculated on a per-page cluster, their system cannot process single web pages from unseen e-news websites, unlike CoreEx. Yin and Lee [13] construct a graph model of a Web page and then apply link analysis on this graph to compute a PageRank-like importance value for each basic element. Unlike our system, they give a continuous measure of importance for every element. Their system yields a recall of around 85% for 788 news articles. Tseng and Kao et al.[14] propose a technique for recognizing primary informative blocks by weighting the blocks using features that capture the ”regularity, density and diversity” of each block. We cannot compare CoreEx with their work as they do not report results on news sites.

2.2.3 Machine Learning

Preceding work has also applied machine learning techniques to address this problem. The Columbia News blaster Project [15] uses an article extraction module that extracts 34 text-based features used by the Ripper machine learning program. Song et al. [16] train models to absorb the importance of blocks in web pages using neural networks and SVMs. The authors found that a feature based on the number of links proved to be the most discriminating in their set of 42 features. Lee et al. [16] proposed PARCELS, a system that uses a co-training approach with stylistic and lexical features

to categorize the blocks inside a web page. Gibson [17] use a Conditional Random Field sequence labeling model to label parts of the page as content or not content.

2.2.4 Wrappers

Laender et al. [18] deliver an analysis of several web page data extraction systems which use wrappers for their approaches. Muslea et al. [19] and Kushmerick et al. [20] discuss automatic learning of wrappers. Metanews et al. [20] is an information gathering agent for e-news articles that employ wrappers. It eliminates redundant HTML tags and uses pattern matching with site-specific manually defined patterns on the reduced page to extract e-news articles. Though wrappers can provide an excellent text extraction, they work only on precise web pages or sets of web pages that share the same layout. Once the layout changes, the wrappers essential to be updated as well. This instability unfortunately requirements, continuous supervision of wrapper-based approaches.

2.2.5. RSS Feeds

Hao et al. [17] proposed the approach to create automatic e-news article contents extraction based on RSS feeds. This method is appropriate to collect data from frequently updating web pages. This method is layout independent and it does not require to consider about news site before the extraction process.

2.3 E-news classification approach

In our approach, providing categorized e-news articles is an essential requirement for e-news aggregation, summarization, and recommendation models. This process may classify e-news articles into predefined news categories like political news, financial news, sports news, entertainment news, technology news, environment news etc.

There are different types of classifiers are used in different research papers. Basically, classification techniques are classified into five different categories such as supervised machine learning algorithms, unsupervised machine learning algorithms, semi-supervised machine learning algorithms, content-based learning algorithms, and statistical learning algorithms [21], [22]. The learning algorithm in supervised machine

learning is provided with input values, and output labels do not easily identify a function that approximates this behavior in a generalized manner. Examples of supervised learning techniques are SVM, decision trees, genetic algorithm, artificial neural network, Naive Bayes, Bayesian network, and random forest.

2.3.1 Support Vector Machines

SVM is a supervised learning algorithm which works better with smaller datasets too. It is a very powerful algorithm. It could be used for both classification and regression approaches. However, classification is the most widely used approach of SVMs. In SVM s each data sample is plotted in a high dimensional space with the attributes of the data sample as dimensions [23]. Through finding the hyperplane, which separates the two classes very clearly classification can be done. Attributes represented by binary classifiers are known as binary attributes. The presence or absence of the attribute is detected by the binary classifier. The most popular attribute learning model is this classifier.

In Inoshika et al.[24], the training process was developed in order to recognize whether the selected message belongs to the group ‘A’, messages will be classified as "Group A" or "other". Then messages in “other” as “Group B” or “other”. That classification is done until all classed are classified. In this approach, the process needs to do repetitively. It’s time and cost consuming. So there should be an optimal mechanism to use the SVM machine learning algorithm in the more efficient way.

2.3.2 Naïve Bayes Classifier

Naïve Bayes algorithm classifies a dataset into two or many classes. Ramon et al. [25] proposed method using Naïve Bayes to classify newspaper advertisements. Naïve Bayes is statistical classification technique. Let $w_1^n = w_1 \ldots w_n$ denote the n words representing the textual content of the advertisement. The classification score is

$$P(C).\prod_{i=1}^{n} P(w_i|C)$$

Some researchers comparing the performance of two or more individual classifier on the same data set to show that which classifier perform well on what kind of data set. In the context of combining multiple classifiers for text classification, a number of

researchers have shown that combining multiple classifiers can improve classification accuracy.

2.3.3 K- nearest- neighbour algorithm

Jiang et al. proposed a text classification approach based on a modified K- nearest-neighbor algorithm. It is combined with a constrained one pass clustering algorithm [26]. Uguz et al. The proposed algorithm for reducing the number of features using information gain feature selection approach. For feature selection the genetic algorithm and principle component analysis are applied. Then for the classification k-nearest neighbor algorithm and decision tree algorithm will be used. This approach is efficient but could be improved by the introduction of few more powerful classifiers or an ensemble of classifiers [22].

2.4 E-news Aggregation

The basic idea of e-news aggregation is identifying similar e-news articles from different sources and putting them into a single location for easy viewing. Mainly there are three subtasks in the e-news aggregation process namely preprocessing, feature extraction and clustering.

2.4.1 Feature extraction

Various methods and techniques can be applied for the extraction of features and they are described in detail in the following sections.

2.4.1.1 Latent Dirichlet Allocation Model (LDA)

Latent Dirichlet Allocation Model is a topic modeling technique which represents a text document as a mixture of pre-extracted set of topics. The model takes a set of text documents as the input and the number of topics which need to extract as a parameter. The specified number of topics is equal to the number of features. The model extracts the features using following steps. Suppose the specified number of topics is k;

- Read each text document one by one and assign each word in each document into randomly selected topic. This random assignment of words into topics gives topics representation for entire document set and gives words representation for every k topics but these assignments are not correct.
- To improve the quality of the topics, we need to consider each word in each document and calculate two things for each topic.
    - V1 = The number of currently assignments of word *w* from document *d* into topic *t* / The number of occurrences of word *w* in document *d*
    - V2 = The number of currently assignments of word *w* from document *d* into topic *t* / The number of currently assignments of word w from all the documents into topic t
- Then re-assigning word *w* of document *d* into topic *t* which has maximum V1*V2 for word *w* and document *d.*
- More accurate and stable assignments of words can be obtained by repeating the previous step a large number of times.
- After getting more accurate and stable topics, feature value of each document for each topic is calculated as follows
    - The number of all the assignment into topic t from document d / The total number of words in document d

The problem with LDA model is the specification of the number topics is needed before extracting the features. But in the application of news article clustering, the number of topics is not known before running the algorithm [3].

2.4.2.2 Doc2Vec Model

Doc2Vec is an unsupervised feature extraction technique for text documents and it is heavily based on Word2Vec algorithm which represent words as vectors by considering semantic relationship between words. Word2Vec is a neural network model and Doc2vec is an extension of Word2vec.

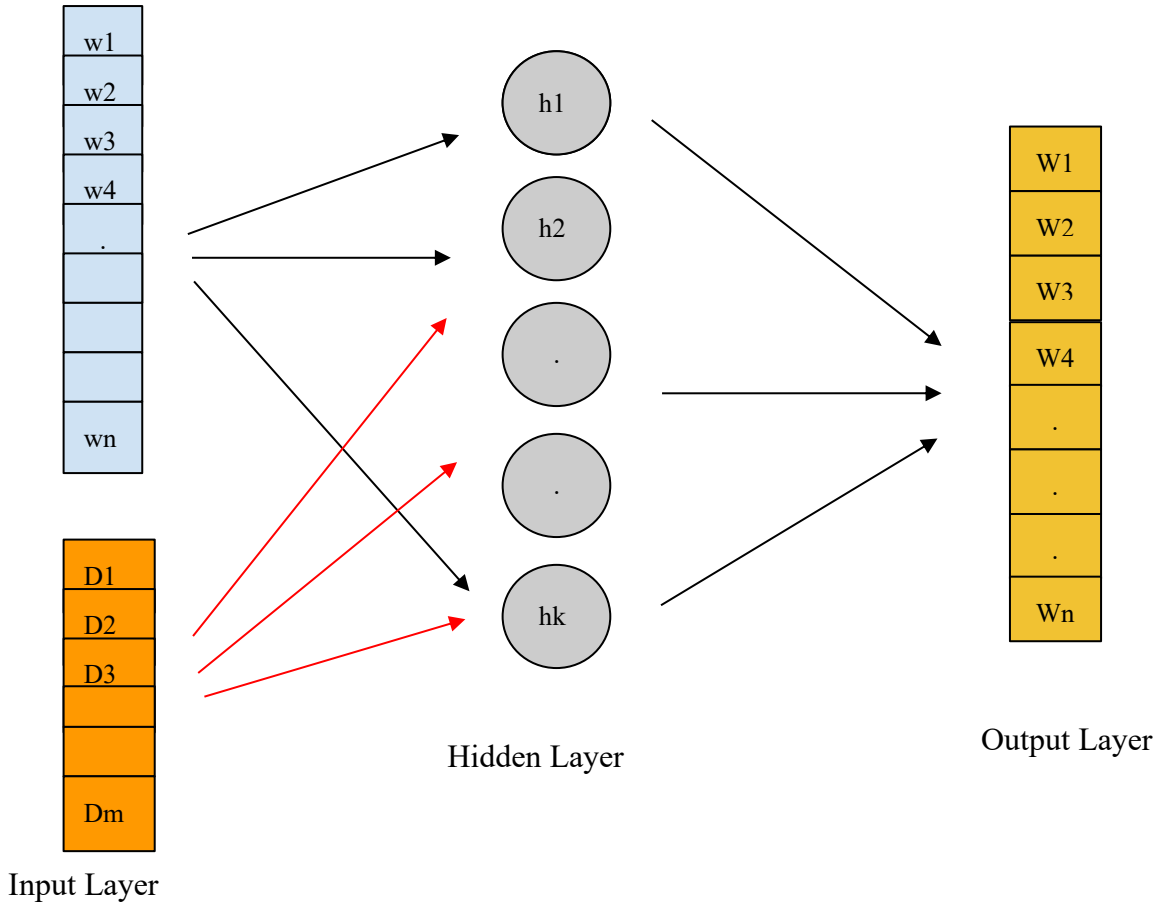

Figure 2.1: Neural Network Model of Doc2vec Model

There is one input node per each word in the corpus and there is one input node per each document in the data set. The only one difference between Doc2vec model and the word2vec model is the document input vector. The network is trained as normal word2vec model and the only difference is that we update the weights between document input vector and hidden layer too. Each document input node has connected to each hidden node and number of nodes in the hidden layer is equal to number of features. The weights between particular document input node and hidden nodes be the feature values for that document. The main problem with doc2vec model is that the algorithm gives poor results when the length of a document is small [4] [28] [29].

2.4.2.3 K-means clustering

K-means algorithm is a very popular clustering algorithm and it is widely used because of its simplicity. Before running the algorithm, the number of clusters should be specified as a parameter.

First, the algorithm randomly initialize a midpoint for each cluster and those midpoints are called as centroids. The number of initialized centroids should be equal to the specified number of clusters. After initializing the centroids, consider each data point one by one and find the nearest centroid for each data point by calculating euclidean distance. The data points which belong to the same centroid are considered as one cluster. After that, calculate midpoint coordinate again by getting average euclidean distance of data points which belong to the same cluster. Then the previous centroids are replaced by the newly calculated midpoints. We need to repeat this process until we get stable centroids.

Mainly, there are two major problems with k-means clustering algorithm. First one is the output is always depend on the initialization of centroids and because of that the algorithm gives different results for different runs. The second problem is, we need to specify the number of clusters before running the algorithm. But in the application of news article clustering, the number of clusters is not known before running the algorithm [30].

2.4.2.4 Affinity propagation

Affinity propagation is a newly introduced clustering algorithm which is based on the concept of message passing between the dataset. The main problem of k-means algorithm and other similar clustering algorithms is that they require to estimate the number of clusters and selecting initial centroids. Instead of that affinity propagation finds the clusters by taking input measures of similarity between data points, and simultaneously consider all the data points as potential exemplars.

Let's consider $X = \{x_1, x_2, \ldots, x_N\}$ is the data points in the dataset. The algorithm runs recursively by updating two matrices. Those are responsibility matrix and availability matrix. Responsibility *r(i,k)* represents, how well-suited point k is to consider as the exemplar for point i by relatively considering other potential exemplars for point i. The responsibility matrix is updated by using following function.

$$r(i,k) = s(i,k) - \max\left(a(i,k') + s(i,k')\right) \text{ Where k'}\neq\text{k}$$

Here *s(i,k)* represent the similarity value between *i* and *k* data points. According to the above function, for the calculation of *r(i,k)* the algorithm required the similarity value*s(i,k)* and availability value *a(i,k)* calculated by the previous iteration. At the initial step, all availability values are set to zero. Availability *a(i,k)* represents how appropriate it would be for point i to choose point k as its exemplar, taking into account the support from other points that point k should be an exemplar. The availability matrix is updated by using following function.

$$a(i,k) = \min\left(0, r(k,k) + \sum_{i' \neq j,k} \max\left(0, r(i',k)\right)\right)$$

Where $\boldsymbol{i \neq k}$ and

$$a(k,k) = \sum_{j' \neq k} \max\left(0, r(i',k)\right)$$

The responsibility matrix and availability matrix is updated until the cluster boundaries remain unchanged over a number of iterations, or after some predetermined number of iterations. At any point in process, summing Responsibility (r) and Availability (a) matrices gives the clustering information. The exemplars are selected from the final matrices if $r(i, i) + a(i, i) > 0$.

The problem with affinity propagation clustering algorithm is that the algorithm cannot identify the outliers. In the application of news article clustering, an outlier is a news article which has not any similar news articles in the dataset [31].

2.5 Intelligent e-news summarization

Automatic text summarization can be defined as “the process of automatically creating a shorter version of text as its essential part”[34]. An automatic summarization approach will have mainly three steps to follow. They include topic identification, interpretation and summary generation. In the topic identification step the most prominent parts from the original text need to be identified and there are various techniques available in the context for topic identification. Interpretation is used to remove the redundancies and merge different subjects to form one general content. In the summary generation step the system uses text generation method and the sentences are put into the summary in the order of the position in the original document [35]. There are two main types of automatic text summarization namely abstractive text summarization and extractive text summarization.

2.5.1 Extractive text summarization

Extractive text summarization provides a syntactic level of representation which extracts the salient parts from the original text and then concatenate them into a shorter form to generate the summary [36]. Extractive summarization mainly involves the major steps namely pre-processing, processing and the post processing steps. A structured representation of the original text can be taken after pre-processing. Sentence boundary identification is a pre-processing technique which identifies the sentence boundaries with the presence of a dot at the end of a sentence. There are many NLP tools available for sentence tokenization like OpenNLP, NLTK and TextBlob. Word tokenization breaks down the extracted sentences into meaningful units called tokens. A token can be an individual word, number or a punctuation mark. The tools for word tokenization in the NLTK include TreebankWordTokenizer, WordPunctTokenizer, PunktWordTokenizer, WhitespaceTokenizer etc. Stop word elimination removes the common words with no semantic like ‘a’, ‘and’, ‘the’ which do not have any emphasis on the summary generation. Stemming derives the stem or radix of each word which emphasize its semantics. Porter stemming algorithm is one of the most popular stemming algorithms available in the context.

In the processing step we have to define the algorithm of the text summarizer. So the features which influencing the relevance of sentences are decided and weights are assigned to each feature and thereby assign scores for each sentence. Then the top ranked sentences are used to generate the summary. There is lot of work that has been carried out in the extractive summarization. There are various extractive text summarization approaches used by the researchers and some of them are described below. Finally the post processing involves tasks like redundant sentence removal and sentence ordering.

2.5.1.1 Term Frequency-Inverse Document Frequency (TF-IDF) method

The term frequency-inverse document frequency is a very primitive method of text summarization. Hans Christian, Mikhael Pramodana Agus, Derwin Suhartono et al. [37] have discussed a Term Frequency-Inverse Document Frequency (TF-IDF) based text summarization approach. TF-IDF is used as a measure to score the sentences which is based on the word frequencies which reflects how important a word is to a document in the corpus. They had only used nouns and verbs of the text considering that they bring the important details about the text. Then a score is assigned to each sentence by taking the sum of the TF-IDF values of every noun and verb in the sentence. Then the sentences are arranged in the descending order of their scores and the final summary is generated using only the top ranked sentences. The amount of sentences extracted for the summary depends on the compression rate.

The major limitation of this approach is that the accuracy of the results was low since they have used only one measure, the TF-IDF to extract the important sentences from the text. But if number of features like cue words, relative length of sentences, identification of title words could be integrated together then the accuracy will be higher.

2.5.1.2 Text summarization with Artificial Neural Networks

Dharmendra Hingu, Deep Shah, and Sandeep S. Udmale et al. [38] have presented an Artificial Neural Network based approach for extractive text summarization. They have

used features like relative position of sentences, named entities, cue-phrases, title relevance, relative length of the sentences, frequencies of words and numerical data to train the neural network. The neural network consists of input layer neurons for the above features, a hidden layer and one output layer neuron which outputs the score of each sentence. Here synonym checking is also performed to assign same weights for words with the same meaning. So the weights for these extracted features for each sentence are fed into the neural network as inputs. Supervised learning is used to train the network and the output is the score for each sentence based on the weights of features fed into the system. That score is directly proportional to the importance of the sentence. These scores are then used to generate the summary.

This approach is better than the TF-IDF method since it concerns number of features when extracting the most important sentences. But the limitation here in this approach is it takes lot of time for the training process because a huge text corpus is required in order to have accurate results.

2.4.1.3 K-means clustering based text summarization

Sumya Akter, Aysa Siddika Asa, Md. Palash Uddin, Md. Delowar Hossain, Shikhor Kumer Roy, and Masud Ibn Afjal et al. [39] have designed a multi document extractive summarization system using the k-means clustering algorithm. In this approach the word scores are given based on the TF-IDF measure and then the sentence scores are assigned by summing the term frequencies of words in the sentence with its position. If any cue word is present in the sentence the sentence score is incremented by one. Then the sentences are ranked in the descending order of their scores and two clusters are initialized taking the maximum sentence score and the minimum sentence score as initial centroids. Then the Euclidean distance from each sentence to the two centroids are calculated and the sentences are assigned to the cluster which has the minimum distance. Then new centroid values for each cluster are reassigned and the same process is repeated until the centroid values won't change. Finally the top sentences from each cluster are taken to form the final summary.

The limitation in this approach is that it consumes lot of time and effort since defining the k value at the beginning is tricky and the most optimal k value can be gained only by the trial and error process.

2.5.2 Abstractive text summarization

Abstractive text summarization assumes a semantic level of representation of the original text and involve some linguistic processing [40]. Therefore abstractive text summarization involves understanding the main concepts of the original text and express them in a clear natural language. But the researches on abstractive text summarization are not evolved yet and therefore they do not provide acceptable level of accurate results.

2.5.2.1 Rich semantic graph reduction technique

Ibrahim F. Moawad and Mostafa Aref et al. [41] have presented an abstractive summarization approach using the rich semantic graph reduction technique. A Rich Semantic Graph is an ontology based representation developed to be used as an intermediate representation for natural language processing applications. This approach consists of three phases creating a Rich Semantic Graph for the source document, reducing the generated Rich Semantic Graph to more abstracted graph and finally generate the abstractive summary from the abstracted Rich Semantic Graph. The input document can be represented semantically by creating a Rich Semantic Graph. In the Rich Semantic Graph the nodes represent the verbs and nouns of the original text along with edges corresponding to semantic and topological relations between them. Named entity recognition, morphological and syntactic analysis, cross-reference resolution are considered in creating the Rich semantic Graph to reduce the syntactic ambiguity and then retrieve the typed dependency relationships between words. In this phase Rich Semantic sub graphs are created for all the sentences in the input text individually. Then the sentences rich semantic sub graphs are merged together to represent the whole document semantically by creating the final rich semantic graph. In the Rich Semantic Graph reduction phase a set of heuristic rules is applied to reduce the graph by replacing, deleting or consolidating the graph nodes using the WordNet relations. Finally an abstractive summary is generated from the reduced Rich Semantic Graph.

The limitation in this approach is low accurate results are gained since the abstractive summarization is not grown up to the standard yet.

2.6 E-news Recommendation

With the development of the technology, recommendation systems are most important for the online web based applications and mobile applications. It's like tracking user's behavior, interests and day to day works without user's intention. Platforms like Facebook, LinkedIn, YouTube, Netflix and Amazon use recommendation engine for providing better service for their customers [42]. Social media applications like Facebook suggests friends for their users to communicate, LinkedIn provide job recommendations for the users, YouTube suggests recommended videos for the users, Netflix suggests recommended movies for their users, e-commerce sites like eBay, Aliexpress and Amazon recommend goods for the users to buy. They use this kind of a recommender system to increase product sales and user satisfaction, by providing correct and most relevant information.

After the web provides access to the online news articles, online news reading became more popular with millions of sources available to the users to read. A key challenge is to provide user interest and news articles the users want [43]. As a result of this, news recommendation has become a new way to introduce news for the users. Yahoo and Google first introduced this kind of a system to the world and after that many other news providers also identified the value of this and they also changed their news portals for more user friendliness [44] [45]. Because of the internet, everyone can easily access to the web contents which provide enormous number of news articles are updated in every mass media within every minute. When providing news recommendation for the user, we need to consider user's long term and short-term interests and social relationship of the user with the time manner. Many surveys notify that people are not selected news based on titles, and read only 3 or 4 lines. Hot news (popular news) change frequently and it's needed to recommend those in sensitive manner for the users.

Recommendation systems can be categorized into two sections. Those are personalized and non-personalized (Popularity based Recommender System). Non-personalized recommendation means without considering individual user's preference, system

provides general recommendation to the users. When we login to the news portal it provides most popular news items around the world. These popular items are based on age, geography, sex, count of purchases, feedback etc. [45]. Based on these parameters, system calculates the mean of the news rating of all the users and lists down the news articles according to their mean value. This is called as a “stereotyped recommender system”. But there is a problem with this system, when there are less number of ratings available mean value will be less accurate. So, this kind of a system provide less confidence.

Personalized recommendation system means that it provides recommendations based on individual user’s behavior and interests. User’s interests and behaviors are difference from user to user. So, it’s necessary to provide recommendation based on individual user’s preferences. Based on above information user profiles are created, where the user profiles help to provide recommendations for the users. Personalized recommender system can be divided into 3 main categories based on the recommendations made [45].

- Content-based recommendation system
- Collaborative recommendation system
- Hybrid recommendation system

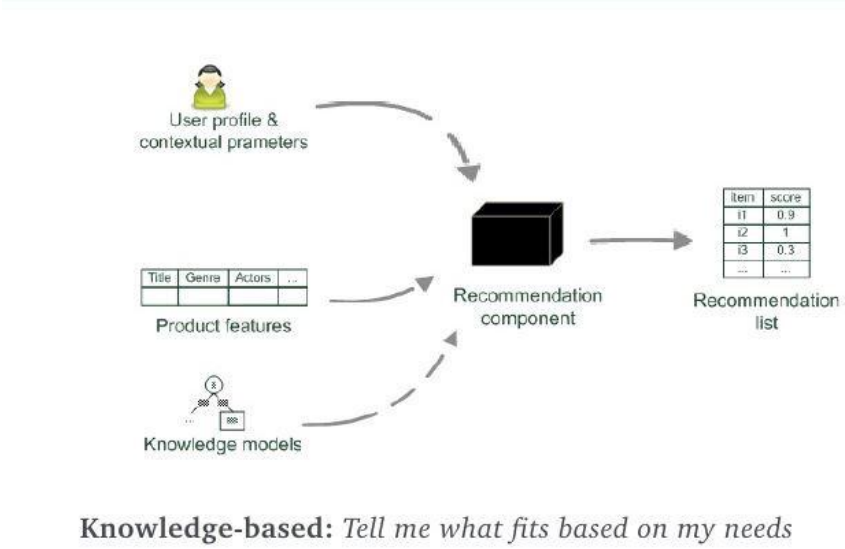

Figure 2.2: Content-based Recommendation System

Content-based recommender systems provide recommendations based on the user's past behaviors and interests. Google and Wikipedia are examples for these kinds of systems. The basic idea of this is keeping keywords of the past articles and provide recommendations based on them. But the biggest problem here is large pie of information provides some difficulties to provide recommendations. As an example, if someone searches for "The University Culture", there will be large number of documents containing the key words "The" and same as for the "Culture" as well, but collaborative filtering algorithm is most widely used algorithm for the news recommendation. News portal like Digg uses this kind of a technique for news recommendation [46].

News content is often represented using vector space model. A well-known method is TF-IDF. Before calculating the TF-IDF values, a series of preprocessing steps are executed, including removing stop words, tokenizing, stemming and so on. Then a news article is represented by a keyword vector, where each entry is the TF-IDF value of the corresponding keyword. Based on the history of user's behaviors, the user profile can be created. For a newly-published news article, we can compute the similarity between the user profile and the news article by similarity functions (Jaccard similarity or cosine similarity).

$$\cos(p_u, q_i) = \frac{p_u \cdot q_i}{\| p_u \| 2 \| q_i \| 2} = \frac{\sum_t q_{ut} p_{it}}{\sqrt{\sum_t q_{ut}^2} \sqrt{\sum_t p_{it}^2}}$$

2.6.1 News Ranking

Created user profile can be used for the news recommendation. For each user, every news article set is evaluated to find the similarities between them. As the first step, it's needed to crawl an upcoming news and then filter noisy words and sentences to extract relevant information from the article. Secondly, the TF-IDF scores are calculated and then stored in "News Profile". Last, these similarity scores which are in User's profile and News Profile are computed using cosine similarity function. Following figure shows the brief understanding about the above scenario according to the rank of the news recommended to the user.

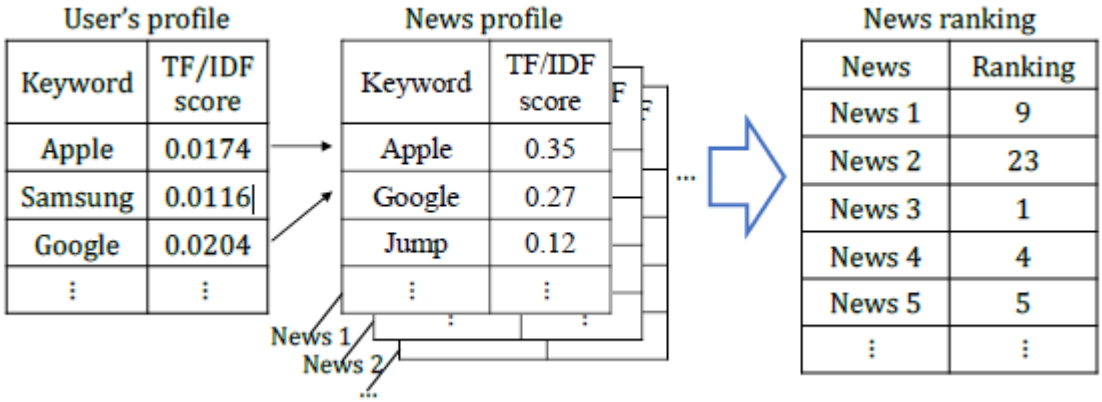

Figure 2.3 : News ranking with user profiles

Collaborative recommender systems are based on the nearest neighbor concept and it consists of two major filtering concepts [42] [45]. Those are User-based filtering and Item-based filtering. User-based filtering looks at the similarities between users and Item-based filtering looks at the similarities between items (News articles). To calculate how similar two users, Karl Pearson's correlation formula is used [42] [47].

2.6.2 User-based Collaborative Filtering Algorithm

2.6.2.1 Improved Pearson Correlation Coefficient Formula

Because the news has the characteristic of strong timeliness, lot of users tend to click on top news and comment in a specific period. When the recommendation system analyzes the user's interests and calculates the similarity of users [7]. Two users reach a consensus over controversial news items is more valuable than the hot news. Visibly, the hot factor will seriously affect the recommendation system on mining interests of users, thereby affecting the personalized service provided to the users.

Therefore, Optimization of Pearson correlation coefficient formula by introducing the parameter of hot, can reduce the importance of the popular news to finding similar users, improve the recommendation accuracy rate and enhance the user experience. The hot (hj) of news j is calculated for analyzing is as follows:

$$h_j = \frac{\sum_{j=1}^{N} r_{i,j}}{N} \, (\mathrm{h_j} > 0)$$

Here, N represents the total number of users (including users who did not score on the news), rij represents the ratings of the user i to the news j. In calculating the sum of ratings, if the user i has no ratings record to news j, skip the user thus it can be seen that more people score on the news j and higher the score, the more popular the news. The hot value range is 0 <hj<Max (rij) [43]. Each user's ratings on the news they visited is a vector, which is expressed as follows:

$$\vec{u} = (r_{1,1}, r_{1,2}, \ldots, r_{1,n})$$

The average score for the user u to all news items is (r_u r, the ratings news set of user x signs as Jx, and the ratings news set of user y signs as Jy, Union news set commented by Users x and y signs as Jxy, Using the traditional Pearson correlation coefficient formula to calculate similarity between the user x and y is as follows [46]:

$$sim(x, y) = \frac{\sum_{j \in J\,xy}(r_{x,j} - \overline{r_x})(r_{y,j} - \overline{r_y})}{\sqrt{\sum_{j \in J\,xy}(r_{x,j} - \overline{r_x})^2}\sqrt{\sum_{j \in J\,xy}(r_{y,j} - \overline{r_y})^2}}$$

The improved Pearson correlation coefficient formula used to calculate the similarity is:

$$sim^*(x, y) = \frac{\sum_{j \in J\,xy}\frac{1}{h_j}(r_{x,j} - \overline{r_x})(r_{y,j} - \overline{r_y})}{\sqrt{\sum_{j \in J\,xy}(r_{x,j} - \overline{r_x})^2}\sqrt{\sum_{j \in J\,xy}(r_{y,j} - \overline{r_y})^2}}$$

As it can be seen from the formula improved, the more popular the news, for the calculation of the similarity between user x and y smaller role.

2.6.2.2 User-Based Collaborative Filtering Algorithm

1. To preprocess the ratings data of user u

Create a user-rating matrix for the target user, and obtains the average ratings of the user u on all news items r_u:

$$r_u = \frac{\sum_{j=1}^{N} r_{u,j}}{N}$$

Here, N refers to the total number of news, Jui represents the ratings of user u on news j.

2. Similarity calculation between users to select the neighbor set U of target user u calculating the similarity of target users and others by improved Pearson Correlation Coefficient formula.

$$sim\ (u,i) = \frac{\sum_{j \in J_{ui}} \frac{i}{h_j} (r_{u,j} - \overline{r_u})(r_{i,j} - \overline{r_i})}{\sqrt{\sum_{j \in J_{ui}} (r_{u,j} - \overline{r_u})^2} \sqrt{\sum_{j \in J_{ui}} (r_{i,j} - \overline{r_i})^2}}$$

Among them, Jui represents the union news set of user u and i commented.

3. Predicting the ratings of user u on candidate news items to obtain the results of recommendation.

The predictive scoring formula of user u for news j is:

$$r_{u,j} = \overline{r_u} + z \sum_{u\acute{} \in U} sim\ (u, u\acute{})\ (r_{u\acute{},\imath} - \overline{r_{u\acute{}}})$$

Wherein, U is the set of neighbors of target user u, z is a normalization factor:

$$Z = \frac{1}{\sum_{u' \in U} sim(u, u')}$$

In addition, during the process of predicting ratings, before the user u's neighbors have an impact on u, the first step is cutting their respective average. It does take full account of impact of previous rating habit of users that may always scores high or low to accuracy and objectivity of forecasting results.

2.6.3 Item-based Collaborative filtering algorithm

Item-based CF looks for items (News articles) that are similar to the articles that the user has already rated and recommended. But what does that mean and when we say item-item similarity? In this case it doesn't mean whether two items are the same by

same attribute, what similarity means is how people treat two items the same in terms of like and dislike. This method is quite stable as compared to user based CF [8], because the average item has a lot more ratings than average user. So, an individual rating doesn't impact as much.

Although user-based CF approaches have been applied successfully in different domains, some serious challenges remain when it comes to large data manipulations systems, which need to handle millions of users and millions of data. When it comes to the news items, vast number of sources provide thousands of news articles daily, so it's impossible to compute and make predictions in real time. Similarity between two news articles (**j1** and **j2**) is calculated by taking the ratings of the users who have rated both the news items using the cosine similarity function.

$$sim\ (j_1, j_2) = \frac{\sum_u (r_{u,j1} - \bar{r_u}\ )(r_{u,j2} - \bar{r_u})}{\sqrt{\sum_u (\ r_{u,j1} - \bar{r_u})^2}\ \sqrt{\sum_u (r_{u,j2} - \bar{r_u})^2}}$$

Once we have the similarity between the items (News articles), the prediction is then computed by taking a weighted average of the target user's ratings on these similar news articles. The formula to calculate rating is very much like the user based collaborative filtering except the weights are between news articles instead of between users. So, predictive scoring formula of user u for news j is [46]:

$$r_{u,j} = \frac{\sum_{j\ \in J}^{t} sim\ (j, j^t) r_{u,j}^t}{\sum_{j \in J}^{t} sim\ (j, j^t)}$$

Most collaborative filtering algorithms are based on neighborhood formation concept. The Neighborhood formation concept is based on Pearson correlation or Cosine similarity algorithms, but the problem is neighborhood algorithms may not be able to produce many news recommendations for the user.

2.6 Summary

This chapter summarizes the recent approaches used to solve the identified problem. Through this chapter the recent approaches and the pitfalls and gaps that exists of them are identified. The way we are going to address the identified issues will be discussed in the next chapters.

Chapter 3

3. Technology Adopted

3.1 Introduction

Through this chapter we are going to focus in brief about the technologies we used in implementing the system. We analyzed many technologies which are most appropriate to be used in implementing our system. These technologies contain certain algorithms, libraries and different technological approaches we have used. After studying about the problem we are going to address in detail, we chose the most appropriate technologies to be used in our approach. This chapter also reveal the suitability of the adopted technologies over the other technologies available.

3.2 Programming Languages

The major programming language used for developing this project is Python. The reason which led us to choose python as the main programming language for our project was that python exhibited high performance for developing machine learning projects against other programming languages. There were also lot of resources which support machine learning tasks for python language. Python is a high level programming language which supports object oriented programming and functional programming. Python is a robust language which provides a variety of useful libraries which makes python a powerful language in order to incorporate with number of functionalities. Pycharm IDE was used as the development environment for python. Python also has the support for GUI designing and Tkinter is a popular GUI toolkit provided by python. It also supports frameworks like Django for developing web applications.

3.3 Development Tools

Scrapy and newspaper libraries were used for extracting the e-news content from different e-news web portals. Variety of other libraries like numpy, scipy, pandas, Natural Language Toolkit (NLTK), scikit-learn, networkx, difflib were also used to implement the system. NumPy provides support for large, multi-dimensional arrays and matrices, along with a large collection of high-level mathematical functions to operate

on these arrays. SciPy is used for scientific computing and pandas is a useful library for data manipulation and analysis. The NLTK provides tools for processing natural language for English language. Scikit-learn is a machine learning library featured with various classification, regression and clustering algorithms. Networkx is a used for studying graphs and networks. The difflib module contains tools for computing and working with different sequences, especially for sequence matching of text.

3.4 Natural Language Processing Toolkit (NLTK)

The system uses NLP tools to convert the human language into the machine language which can be easily understand by the machine. So NLP tools provided by NLTK (Natural Language Tool kit) based on python is also used in the system for algorithms in natural language processing. It supports various operations for processing human language like tokenization, stemming, lemmatization, tagging and parsing. It also provides various resources like lexical dictionaries such as wordNet. It's a free and open source project which supports various functionalities for the processing of English language.

3.5 Application Lifecycle Management (ALM)

Bugzilla server software was used as the ALM tool for this project. It was used for project management tasks like tracking requirements, Project and Sprint backlog items and their progress, planning team capacity and for development related activities such as source control, build management, continuous integration etc.

3.2.5 Version controlling systems

When developing individual modules separately, integrating all the modules together to build the final system is a quite difficult task. Therefore, git was used as the version controlling system and the code repository throughout the development of the system.

3.3 Summary

The technologies we used in general is discussed. These technologies contain certain algorithms, libraries and different technological approaches. There are more tools and libraries that we could use in the implementation stage.

Chapter 4

4. Our Approach

4.1 Introduction

This chapter summarizes the approach used by us to solve the identified problem. The conceptual approach is explained in terms of input, output and process for the modules. The flow of inputs and outputs between the individual modules is identified through this chapter.

4.2 Approach

The major components of the system identified are as follows,

- Data extraction phase
- Classification phase
- Aggregation phase
- Summarization phase
- Recommendation phase

4.3 E-news extraction phase

E-news extraction phase extracts e-news items using web crawling and scarping. We use state-of-art tools as well. Our approach combines two advanced methodologies. The URLs or RSS feeds of the e-news websites are given as inputs and the extracted e-news items from the specified URLs or RSS feeds are outputted by the e-news extraction phase. When extracting data we need to remove irrelevant data such as advertisements, user comments etc. and extract only the e-news content from e-news web sites.

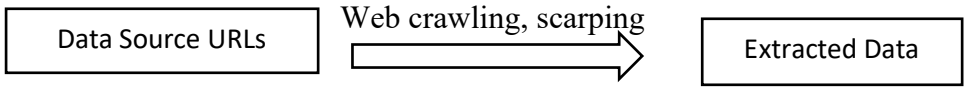

Figure 4.1: E-News Extraction

4.3.1 Inputs into the e-news extraction phase

At the beginning of the process, the user provides root URLs or RSS feeds into the system. Web crawling and extraction are the next stages of the process. Newspaper python library is used for scraping and feedparser library is used to read RSS-feeds. These days most of the e-news websites provide RSS feeds. If there is RSS-feed available, the system uses it for the scraping purpose. Newspaper automatic e-news article scraper is used for other URLs. We scrape from the RSS feed first, because the data is much more reliable when gathering through the RSS feed. JSON file is used to feed system from RSS feeds and root URLs which makes it convenient to add or remove new e-news websites. Feedparser is used to load the RSS feed of the e-news web site. It builds the structure for the data by constructing dictionary Newspaper. An article dictionary is created to store records for each e-news article. Then newspaper library is used to scrape the content of the links which we got from the RSS-feed. Finally, the data object is saved to file as JSON.

4.3.2 Output by the e-news extraction phase

Data object in the JSON format is the output of the e-news extraction phase. It contains e-news titles and content in text format.

4.4 Classification module

Classification module classifies news items into predefined categories such as political, entertainment, sport, business and technology.

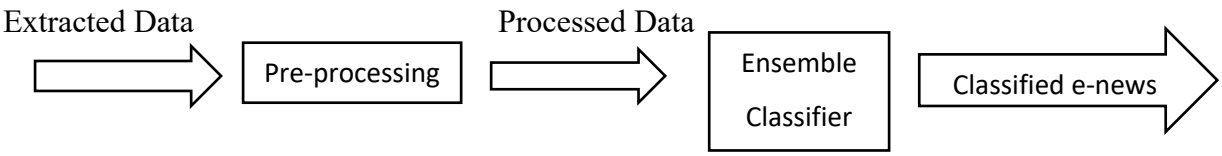

Figure 4.2: E-news Classification

4.4.1 Inputs into the classification module

Extracted raw data is provided as the input to the classification module. Pre-processing techniques are applied to these data to improve the quality of the data. Then learning level gets that information and extracts features from the processed data. It selects most optimal features and then constructs the classifier. In the classification, we use a novel approach by using an ensemble classifier by assembling several different classification models.

4.4.2 Process of the classification module

The whole functioning of the classification module can be subdivided into different tasks namely;

- Pre-processing
- Feature extraction
- Training each classifier
- Building the ensemble classifier

4.4.2.1 Pre-processing

The pre-processing is significant in text processing as well as text classification, because it supports, to sum up, an article effectively by removing redundant words and confirming each word to its root form, helping the classifier to recognize the article more easily and efficiently. In pre-processing, we remove noise from data, which are irrelevant to our process such as advertisements, user comments, etc. and do transform cases, text tokenization, stop word removal, word stemming, lemmatization and POS (Part Of Speech) tagging.

4.4.2.2 Feature extraction

Feature extraction is the next stage. We use “term frequency–inverse document frequency” (TD-IDF) method to extract features from the e-news items. Using this method we can measure how frequently a word is occurring in an article. Each

document has different lengths. So long article could have a higher frequency than the shorter article. Therefore as a normalization method, the term is divided by the article length. BBC data set is used to extract features.

4.4.4.3 Training each classifier

BBC news dataset contains both news and related categories of each news item. These news and the labels are used to train the classifiers.

4.4.4.4 Building the ensemble classifier

Ensemble classifier by assembling several different classification models together is used for our classification. It gives a low error rate and it lowers over fitting when compared with standalone classifiers. The theory of combining classifiers is suggested as a novel direction for the improvement of the performance of individual classifiers. Support Vector Machine (SVM), Neural Networks, Naive Bayes classifier, Decision Trees, Discriminant Analysis, Nearest Neighbours and Random Forest algorithms are considered as the most powerful classification algorithms. Therefore, the classification algorithms namely the SVM, Random Forest, and Multi-normial Naïve Bayes were used to build the ensemble classifier after evaluating each of them

The soft voting method is used for the ensemble classifier and it gives more accurate results than the hard voting method which doesn't concern about the percentage of each voting values.

4.4.3 Output by the classification module

E-news article category name i.e. political, business, entertainment, technology, sports or other is given as the output by the classification module.

4.5 Aggregation module

The goal of news aggregation phase is to identify the news articles which are related to the same topic or incident and cluster them. In previous phase the classifier categorizes

the news articles into different classes such as political, business, sports etc. The aggregation module reads this classes one by one and finds clusters based on the contents of the articles.

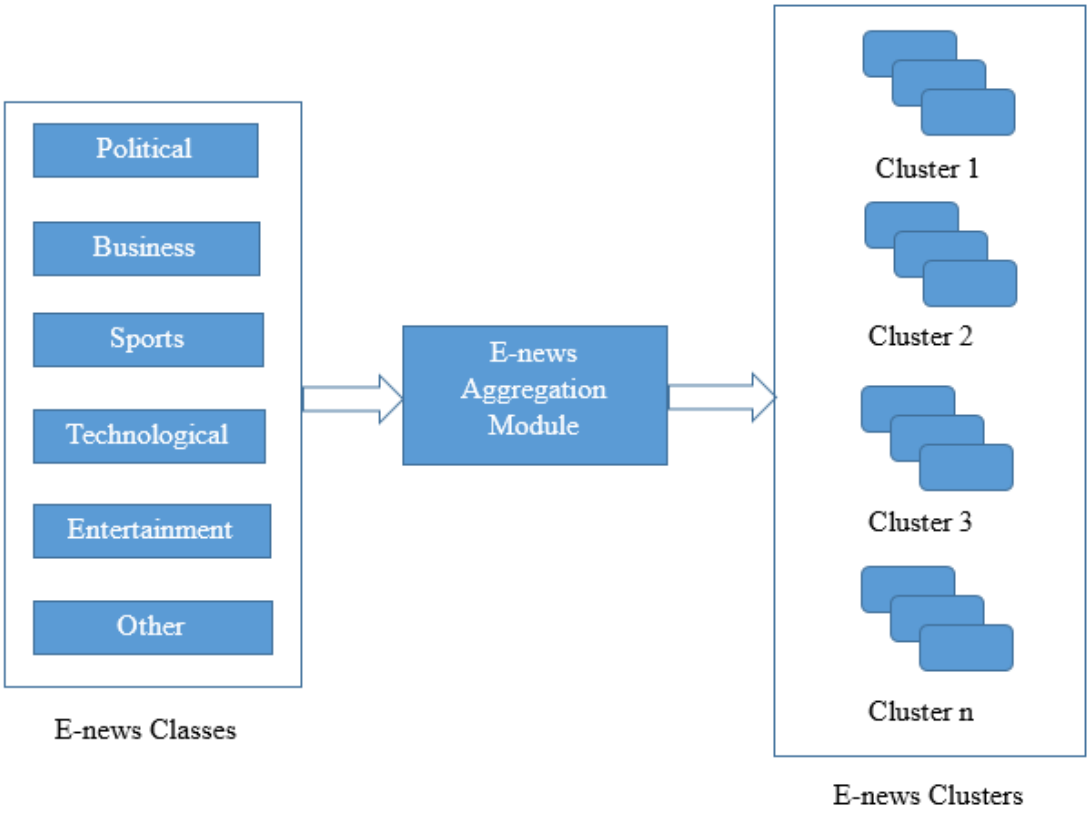

Figure 4.3: Abstract view of the e-news aggregation component

4.5.1 Inputs into the Aggregation module

Each news classes found by the classification phase such as politics, business, sport etc. are given as inputs into the aggregation module individually.

4.5.2 Processing of Aggregation module

The basic process of the news aggregation module can be divided into following steps.

- Preprocessing
- Features Extraction
- Clustering

4.5.2.1 Preprocessing

In the preprocessing phase, the text documents are tokenized into words. Then stop words and punctuations are removed. Stemming is applied as the final step of preprocessing.

4.5.2.2 Feature Extraction

A news article is just a group of words. These group of words should be represented as a group of numerical values to perform a clustering algorithm. Vector Space Model is such a text representation technique where it represents a set of text documents as a group of vectors based on term frequencies. In this phase Tf-Idf vector is calculated as the feature vector for each of the news article.

4.5.2.3 Clustering

In the clustering phase, news articles are grouped as clusters based on the similarity of feature vector which is obtained by the feature extraction phase. The articles which are within the same cluster share similar topic or incident and the articles within different clusters share dissimilar topic or incident. The clustering is performed by using Density Based Spatial Clustering of Application with Noise (DBSCAN) algorithm.

4.5.3 Output of Aggregation Module

The output of news aggregation module is a set of group of news articles about the same topic. The articles belongs to same group are sharing similar meaning and the articles belongs different groups are sharing different meaning.

4.6 Summarization module

The ultimate goal of this module is to generate individual summaries for the clustered e-news articles generated as outputs by the aggregation module. This generates extractive summaries which extract the most salient information from the original source documents themselves. The summarization module identifies the most

prestigious sentences from the original set of documents in the cluster by assigning an importance score for each sentence in the original documents. Then it generates individual summaries for the e-news articles in the cluster. Then those individual summaries are compiled to form an aggregated intermediate level of summary and then the redundancies are removed from the aggregated summary. The final summary sentences are arranged in the coherent order and separate summary for each cluster of e-news articles is presented to the users.

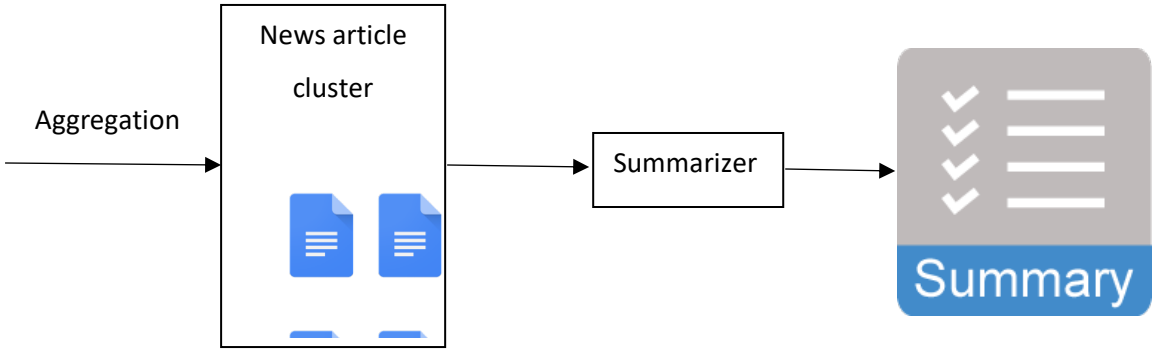

Figure 4.4: Abstract view of the e-news summarizer component

4.6.1 Inputs into the summarization module

The summarization module accepts e-news clusters outputted by the aggregation phase separately. A cluster contains e-news articles which discuss about the same topic. Within a particular news category there is a set of e-news clusters generated and that set of clusters is given as inputs to the summarization module. Summarization module accepts clusters from each e-news category too.

4.6.2 Processing of the summarization module

The processing of the summarizer consists of several phases namely;

- Pre-processing phase
- Document processing phase
- Sentence scoring phase
- Post-processing phase

4.6.2.1 Pre-processing phase

Each e-news article from the cluster were subjected to different types of pre-processing techniques. Therefore the pre-processing phase involves techniques like sentence tokenization, word tokenization, stop word removal, stemming, lemmatization and POS (Part Of Speech) tagging.

4.6.2.2 Document processing phase

The hardest part in this research part is identifying the most significant sentences from the original documents in the cluster to be extracted as the summary sentences. Therefore we need a mechanism of ranking the sentences based on some measures of importance to extract the important sentences. So a combination of few approaches were used to assign each sentence an importance score and thereby identifying the important sentences from the source documents. The approaches include:

- Graph based approach
- Feature extraction based approach

4.6.2.2.1 Graph based approach

A sentence similarity graph is generated for the original documents based on the similarities between the sentences. Various similarity measures are available in the context to find the similarities between the sentences. Cosine similarity, BOW (Bag of Words) measure, Euclidean distance, Jaccard similarity are such measures used to find the sentence similarities [48]. Then based on the similarities measured between sentences the sentence similarity graphs are generated for original documents.

4.6.2.2.2 Feature extraction based approach

In this method the importance of the sentences are detected by considering the presence of a set of pre-defined features in the sentences. Here we need to identify the type of the feature set to be considered when determining the most important information from the source documents. They include features like the sentence length, sentence position,

title word feature, named entity count, verbs count, nouns count, presence of numerical values, key word frequencies etc. [38].

4.6.2.3 Sentence scoring phase

Based on the results of the previous phase a sentence score is given to each and every sentence in the source documents. Sentences in the sentence similarity graphs are scored by using the graph based ranking algorithm namely the PageRank algorithm. In the feature extraction based approach, weights were assigned to the feature set based on the importance of each feature when generating the summary. Then each sentence in the source documents is given a score as a weighted average score. The final sentence scores are assigned as an average of the scores by PageRank and weighted average scores [49].

Then the sentences are arranged in the descending order of sentence scores and the sentences with high ranks which cover a compression rate of 30% of the original documents are extracted to generate individual summaries for the e-news articles in the cluster [6]. Then all these individual summaries are summed up to form an intermediate level of summary.

4.6.2.4 Post processing phase

Since the sentences for the summary are extracted from multiple e-news sites there can be some redundancies of sentences and those redundancies need to be removed. The similarities between sentences are examined in three perspectives namely syntactic similarity which identifies sentences which have the similar syntactic relationship, lexical similarity which identifies similar sentences based on the total overlap between vocabularies and semantic similarity which identifies sentences with the same meaning. After identifying these similarities between the sentences, the redundant sentences are removed from the aggregated intermediate level of summary.

Sentence ordering is another post processing task identified in order to arrange the final summary sentences to follow the coherent order. If there is no flow between the final summary sentences, then it won't be readable which displays characteristics of a poor

summary. So the sentences are arranged in the coherent order by sequence matching and form the final summary.

4.6.3 Output by the summarization module

The output of the summarization module is an extractive summary which contains the sentences with key information from the set of original documents in the cluster themselves. So here all the individual summaries generated for each news article within a cluster are processed and integrated together to form a one final summary. The summarization module provides summaries for each e-news cluster in each e-news category.

4.7 Recommendation phase

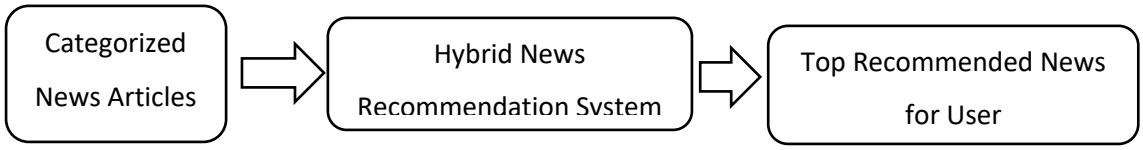

Figure 4.5: Abstract view of the recommender component

Categorized e-news articles are stored inside the article database and those articles are used by the Hybrid News Recommendation System to recommend news articles for the user. Hybrid News Recommendation System consists of Content based filtering module, Collaborative filtering module and Location aware personalization module. The recommendation module provides the top N recommendations for the user. User wise article scores (ranking) are stored inside article database that help to recommend news articles based on the similar users which recommends news articles preferred by one user to another user who has similar kind of interests. It helps to find similarities between users and similarities between News articles (Collaborative Filtering).

4.8 Summary

This chapter identifies the overall high level architecture of the proposed system and summarizes the contribution of the individual modules in terms of flow of inputs and outputs and process to achieve the goal of the system, Conceptual structure and function of the individual modules is summarized through this chapter.

Chapter 5

5. Analysis and Design

5.1 Introduction

This chapter focusses on the design of our approach through which we are going to address the identified problem. The top level architectural diagram of the main system elements and connections between them are described. Diagrams and designs are included in this chapter with the description.

5.2 High level design and architecture

The system extracts e-news articles from a set of pre-defined e-news sites as data sources namely BBC, CNN, Ada derana, News first, Daily news, the Guardian and Fox news. Here given the URLs or the RSS of the e-news web portals the system automatically connects with them and extracts news items from them. This extracts only the text information from varied e-news web sites by excluding other irrelevant content like advertisements and user comments. This process needs to be performed at each specified time interval because these e-news sites are updating very often by adding fresh news items more frequently. Then we need to pre-process these data and then purified data is used for classifying the news articles into different pre-defined categories like political, business, entertainment, sports, technology news etc. The classified news articles are further processed by the aggregation module and the similar news articles which are describing about the same news item from different e-news web sites are aggregated together and displayed in a single place. The aggregated news items are used by the summarization module to generate a separate summary for each e-news cluster. Here an extraction based summary is generated by the system by extracting the most salient information from original documents themselves. The recommendation module uses the previously classified original e-news items by the classification phase and gives personalized news recommendations by tacking user behaviors and by identifying similar user groups. The user behavior is tracked by considering the past browsing history of the users. The recommendation system builds user profiles which contain user's news interests. Here a hybrid approach with collaborative filtering, content-based filtering and popularity model is used by the recommendation module.

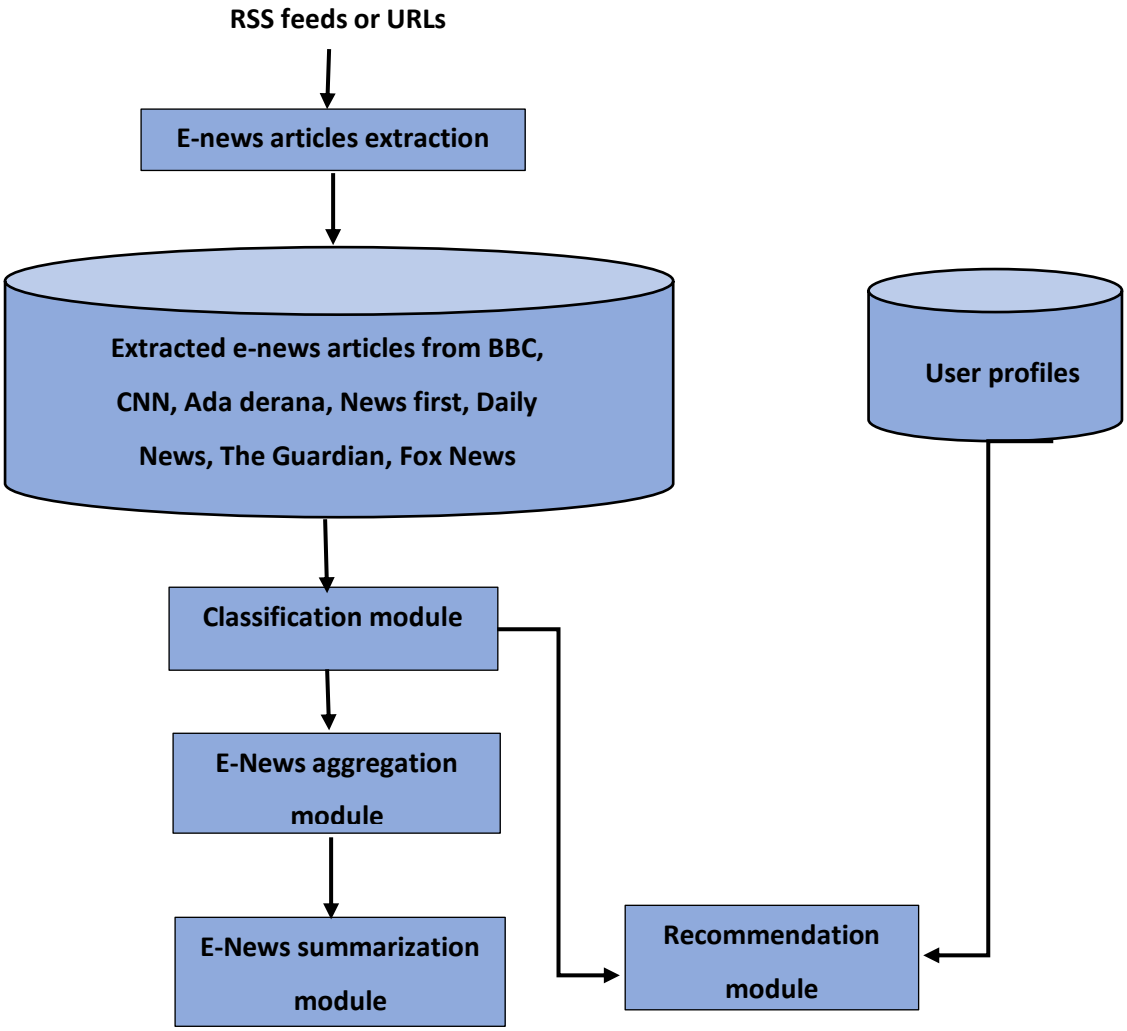

Figure 5.1: High level system architecture

5.4 e-New Extraction and Classification Design

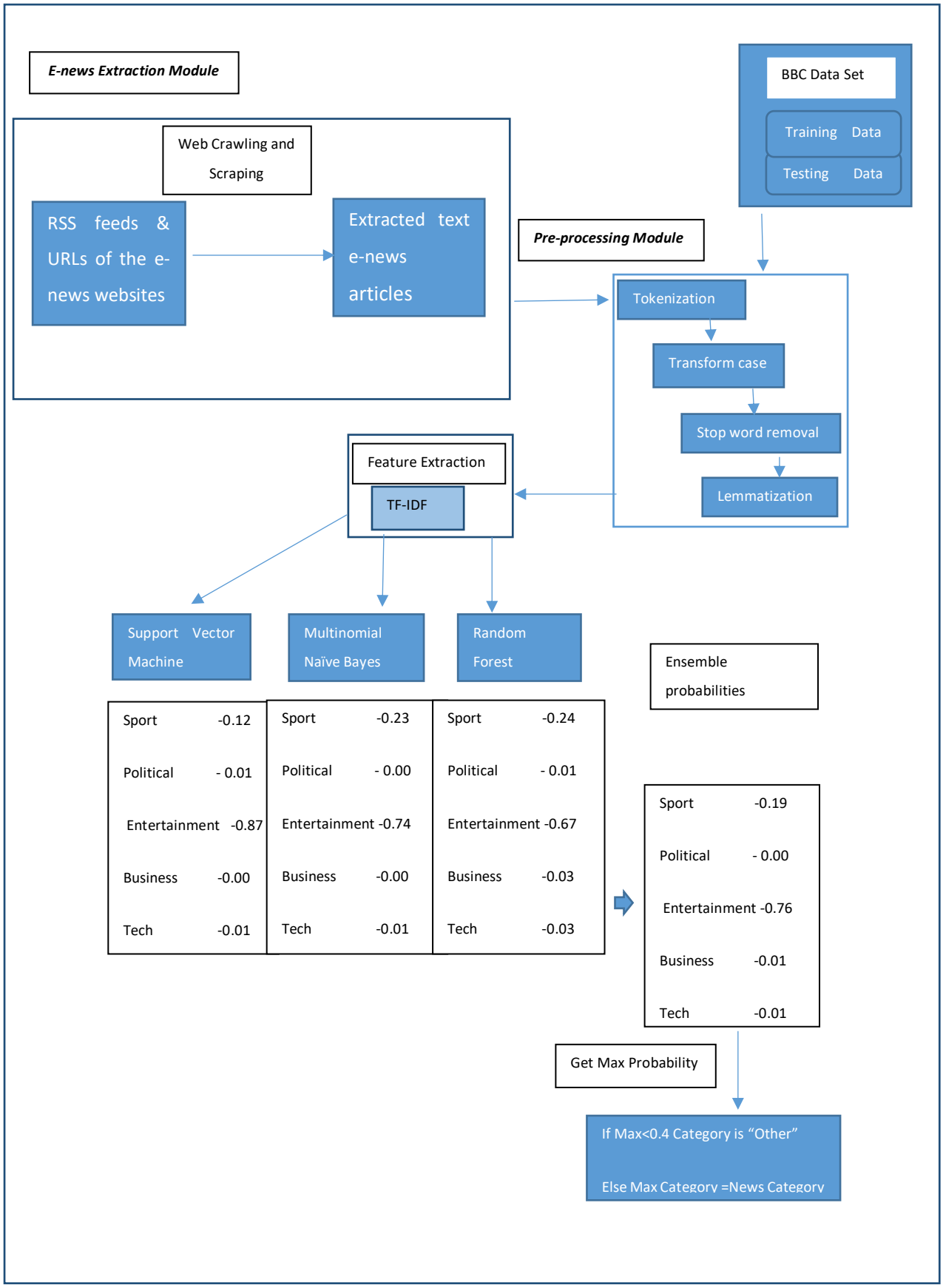

Figure 5.2: E-news Extraction and Classification Module Design

5.4.1 E-news Extraction Level

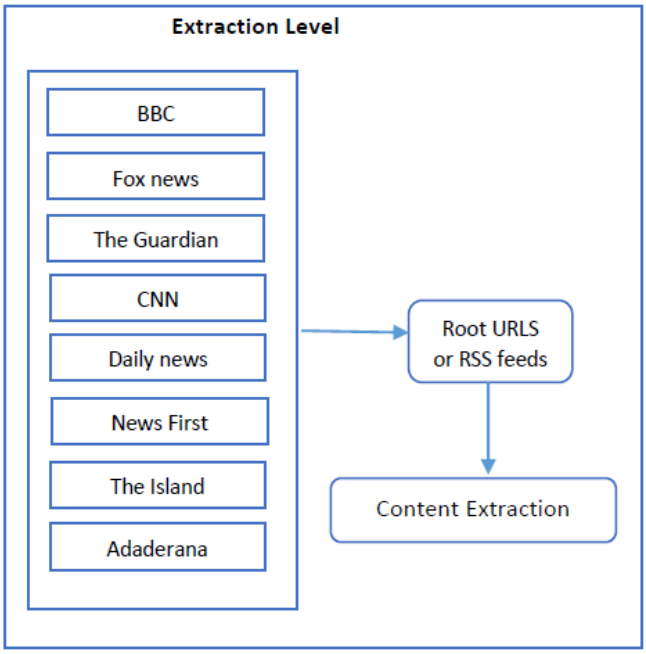

Figure 5.3: e-news Articles Extraction

E-news extraction phase uses state-of-the-art tools, which we could extend with functionality to meet the defined requirements. In extraction, we use e-News websites as data sources such as BBC, CNN, Daily News, Daily Mirror etc.

Our approach combines two advanced methodologies. At the beginning of the process, the user provides root URLs or RSS feeds into the system. Web crawling and Extraction are the next stages of the process. Newspaper a python library is used for scraping and feedparser is used to read RSS-feeds. These days most of the e-news websites provide RSS feeds. If there is RSS-feed available, we use them for scraping e-news articles. Newspaper, an automatic e-news article scraper is used for other sites. First it scrapes from the RSS feed, since the data is much more reliable when gathering through the RSS feed. To feed system from RSS feeds and root URLs JSON file is used. Therefore, it is convenient to add or remove e-news websites. Feedparser is used to load the RSS feed of the e-news web site. Build the structure for the data by constructing dictionary Newspaper. An article dictionary is created to store records for each e-news article. Then Newspaper library is used to scrape the content of the links which we got from the RSS-feed. Finally, the data object is saved to file as JSON [14].

Figure 5.4: RSS feed of The Guardian website.

5.4.2 Pre-processing Level

The pre-processing procedure is significant in text processing as well as text classification, because it supports, to sum up, an article effectively by removing redundant words and confirming each word to its root form, helping the classifier to recognize the article more easily and efficiently. In pre-processing, we remove noise from data, which are irrelevant for our process such as advertisements, user comments, etc. and do transform cases, text tokenization, stop word removal, word stemming and lemmatization [21].

5.4.2.1 Transform Cases

Transform case use to convert all the news items into lowercase letters. It helps to get an effective outcome from other pre-processing stages.

5.4.2.2 Text Tokenization

The tokenize technique breaks raw e-news strings into sentences, then breaks those news sentences into words and punctuation, and after applies a part of speech tag. This approach eliminates white spaces, tab, newline, etc. The token is then normalized. NLTK toolkit is used for text tokenization [21].

5.4.2.3 Stop Word Removal

E-News becomes a rapidly growing communication medium where different type of users are involved in. As a result, irrelevant textual data, abbreviations, irregular expressions and infrequent words can be created. To reduce that noise of textual data is essential for the accuracy of the sentiment analysis. Hence, to produce quality data set we are removing such stop words from the web comments by using pre-compiled stop word lists or stop word identification in NLTK package. Stop word removal is done by default the set of English language stop words from NLTK is used.

5.4.2.4 Word Stemming

Stemming is used to find out the root or stem of a word. There are sets of rules to apply. This is one of the most important steps in the pre-processing process. Stemming reduces the time consuming and space required and that increase efficiency of the classifier [21]. In our approach S-Stemmer, Lovins, Porter, and Husk Stemmer are used.

5.4.2.5 Lemmatization

Lemmatization is the procedure of looking up a single word from the range of morphologic affixes that could be applied to specify tense, gender, etc. First need to recognize the WordNet tag form based on the Penn Treebank tag, which is returned from NLTK's standard pos_tag function. If the tag with 'N', 'V', 'J' or 'R' then can properly identify if it's a noun, verb, adjective or adverb. We then use the new tag to look up the lemma in the lexicon [25]. The WordNetLemmatizer looks up data from the WordNet lexicon and does Lemmatization on the news items.

5.4.2 e-News Classification Level

The classification module consist of feature extraction and ensemble classification

5.4.2.1 Feature Extraction

TF-IDF model is used to extract features. As the dataset, we used BBS news dataset which contains 2225 news articles with class labels. The TF-IDF contains Term Frequency and Inverse Document Frequency. Term frequency summarizes how often a given word appears in a document. Inverse document Frequency considers no of documents as well. Therefore, it downscales words that appear a lot across documents. We extracted 14788 features using TF-IDF [23].

$$log \frac{\#docs}{1 + \#docs\ using\ word}$$

TF-IDF output a numerical binary array.

5.4.2.2 SVM

This linear classification method can be used for multiclass classification other than the binary classification. SVM which is doing the classification using linear decision boundaries is called as linear SMV and as well as with the little enhancement of the algorithm SVM can be modified for nonlinear classification which uses the non-linear decision boundaries. SVM is a supervised learning algorithm and for a given set of training data, this algorithm generates an optimal hyper plane which can use to categorize new data items. SVM is commonly recognized to be a more accurate algorithm [24].

5.4.2.3 Random Forest Algorithm

Random forest algorithm is inbuilt ensemble classifier which consists of two main stages. There are random forest creation and make prediction from the created random forest. First randomly select “k” number of features from total “m” number of features. (k<<m) Than determine the node “d” using the best split point among the “K” features. Thereafter divide the node into descendant nodes using best split. After that repeat above steps until “I” number of nodes has been created. Then create the forest by repeating all above steps “n” number of times [24].

5.4.2.4 Multinomial Naïve Bayes

This classifier is suitable to classify discrete features. It is a probabilistic classifier based on text features. Naïve Bayes classifier can be trained very efficiently by requiring a relatively trivial quantity of trained data [24].

5.4.2.5 Ensemble Classification

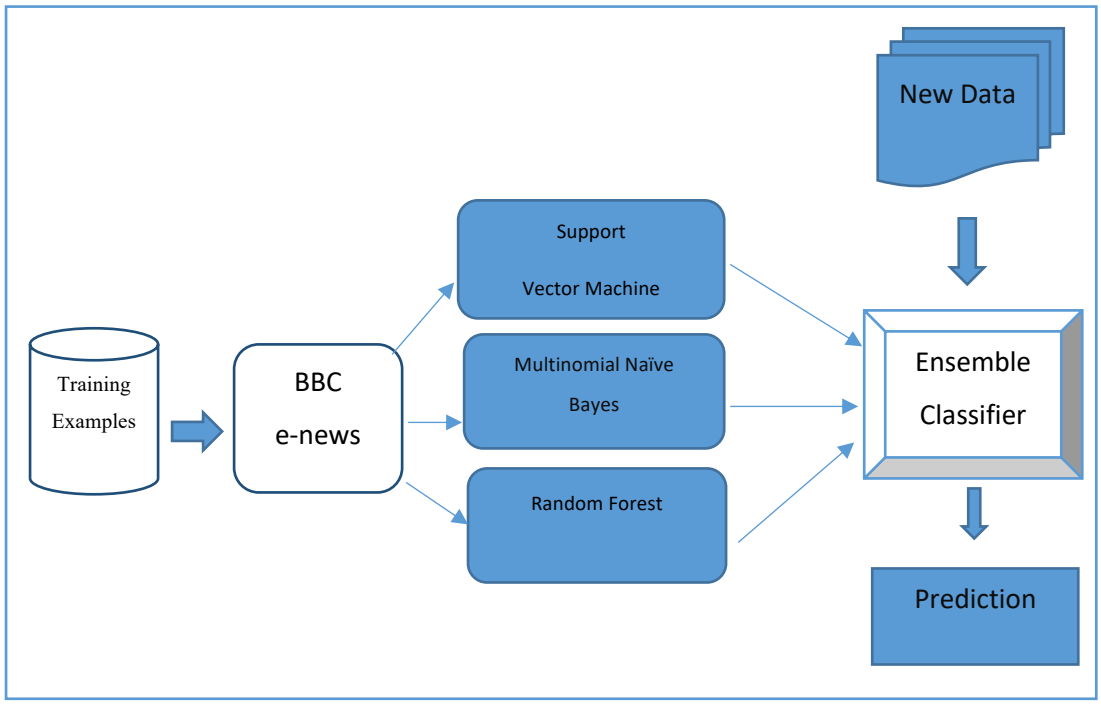

Figure 5.5: Ensemble Classifier

Ensemble classifier is used for our classification. It assembles several different algorithms or several different models together to create an ensemble learner. It gives a low error and lows over fitting than standalone classifiers. The theory of combining classifiers is suggested as a novel direction for the improvement of the performance of individual classifiers. Support Vector Machine (SVM), Neural Networks, Naive Bayes classifier, Decision trees, Discriminant Analysis, Nearest Neighbours and Random Forest algorithms are the most powerful classification algorithms. Therefore we evaluate these techniques and construct ensemble classifier using SVM, Random Forest algorithms, and Multinomial Naive Bayes Used individual trained models for

classification. First extract features from news article using TF-IDF, then feed those features into saved model vectors. Next get probabilities from each model. Those probabilities are used to generate weighted average probability for each class. Than aggregate probabilities from different classes and get the maximum value class as predicted class for particular e-news item. Below shows probabilities, for example e-news article.

Below e-news item gives following probabilistic results
RF- Radom Forest Algorithm
MNB- Multinomial Naïve Bayes Algorithm
SVM-Support Vector Machine Algorithm

Russia wins opening World Cup match against Saudi Arabia

The 2018 World Cup opened in spectacular fashion as Russia defied their recent poor form to score five past Saudi Arabia and record the biggest win by the host nation in the opening game of a World Cup since 1934.

Stanislav Cherchesov's team had not won in their past seven matches and had been criticised from all sides, including a series of barbed comments from Russian president Vladimir Putin.

But in front of a largely partisan crowd of 78,011 at the Luzhniki Stadium they never looked in danger against a naïve Green Falcons' side that seemed only too willing to gift possession to their opponents.

And there was one sour note for the home side when Alan Dzagoev limped off with what looked like a hamstring injury midway through the first half.

But on a night when they were under serious pressure to deliver, Russia got their campaign up and running in emphatic fashion.

Figure 5.6: an E-news item

$$Ensemble\ probability = \frac{P\_RF + P\_MNB + P\_SVM}{No\ of\ classifiers}$$

P_RF=Probability value from Random Forest Algorithm
P_MNB= Probability value from Random Forest Algorithm
P_SVM= Probability value from Support Vector Machine

Ensemble those individual classifiers using probabilities.

Maximum probability value is 0.91112514 and this news is categorized as sport news.

If the maximum probability value is less than 0.4 it should categorized as “Other” news.

Since there are no prominent features for any category.

Steps:

1. Calculate individual probabilities for each classifier and for each type.
2. Calculate ensemble probabilities
3. If maximum ensemble probability>0.4 select maximum probability category as a type of the e-news
4. Else select “Other” as a type of the e-news

5.5 E- news Aggregation Module

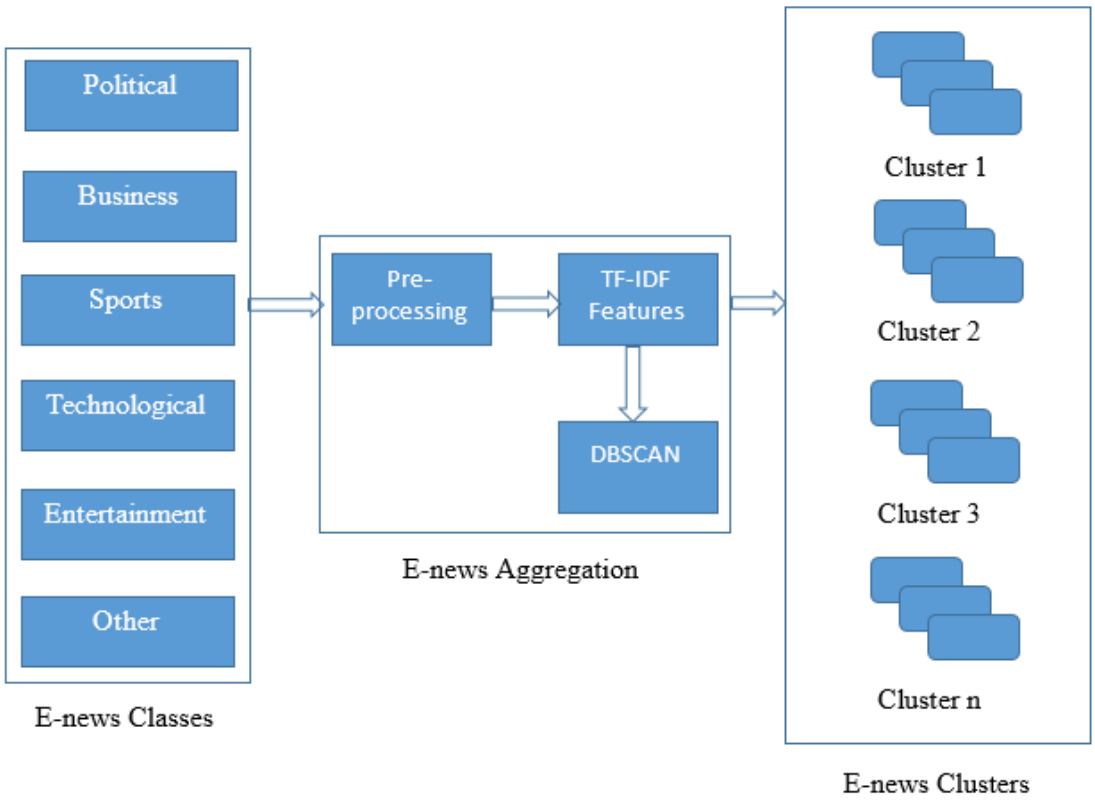

Figure 5.7: Design of the e-news aggregation component

The diagram above illustrates the design of the e-news aggregation module and each phase of the design of aggregation module has described in detail below.

5.5.1 Preprocessing

In preprocessing phase, the text documents are tokenized into words. Then stop words and punctuations are removed. Stemming is applied as the final step of preprocessing. The main problem of Vector Space Model is high dimensionality of data. One of the main purpose of document preprocessing is reducing the high dimensionality of data.

5.5.2 Tf-Idf Features extraction

A news article is just a group of words. These group of words should be represented as a group of numerical values to perform a clustering algorithm. Vector Space Model is such a text representation technique that it represents set of text documents as a group of vectors based on term frequencies.

Given a set of news article documents $D = \{d_1, d_2, \ldots, d_i, \ldots, d_N\}$. Where d_i is i^{th} article and N is the number of news article in the data set. Every document in the dataset is represent as a term weight vector, $d_i = \{w_{i1}, w_{i2}, w_{i3}, \ldots, w_{it}\}$. Where w_{ij} is j^{th} term weight of i^{th} document and t is the number of unique words in the entire dataset. TF-IDF score is calculated using the following equation and that value is taken as the term weight.

$$tf - idf(i,j) = tf(i.j) * idf(j)$$

Where *tf(i,j)* is the term frequency of j^{th} term in the i^{th} document. *idf(i)* value is calculated as,

$$idf(i) = \log\left(\frac{N}{df(j)}\right)$$

Where *N* is the number of articles in the dataset and *df(j)* is number of articles which are contained j^{th} term [33].

If the term frequency of a particular term in a particular article is high, the tf-idf value is high. That means if a term is appearing in an article very frequently that article has high weight for that dimension. On the other hand if a term is appearing in most of the documents in the data set, *idf* gives low value and tf-idf also gives low value. If a word

is occurred very frequently in a document but it contains in a less number of documents in the data set tf-idf gives high score [32].

5.5.3 Density Based Spatial Clustering of Application with Noise (DBCAN)

E-news clustering is the process of grouping news articles in such a way that articles in the same group are more similar to each other than those in other groups. DBSCAN algorithm partition data points into dense regions separated by lower dense regions.

There are two global parameters in DBSCAN algorithm [33].

- *EPS* - The maximum distance between two data points which belong to same cluster
- *MinPts* - The minimum number of data points should be there in one cluster.

In DBSCAN algorithm, all the data points can be categorized into three categories.

- *Core point* - If a point has number of neighbor points more than *MinPts* value within its *EPS* range, that point is called as a core point.
- *Border point* - If a point has number of neighbor points fewer than *MinPts* value within its *EPS* range but it is a neighbor point of a core, that point is called as a border point.
- *Noise point* - If a point does not belong to any of above categories, that point is called as a noise point.

There are two concept called, Density-Reachability and Density Connectivity in this algorithm.

- Density - Reachability
    - Directly density-reachable - A point q is directly density-reachable from a point p: if p is a core point and q is in p's EPS range.
    - Density reachable - A point p is density-reachable from a point q: if there is a chain of points P1,...,Pn with P1=q, Pn=p such that P(i+1) is directly density-reachable from Pi.
- Density Connectivity
    - Density connected - A pair of points p and density connected: if they are density-reachable from a common point O.

The formal definition of a cluster can be described as follows.

For given data set D, and parameter EPS and MinPts, A cluster C is a subset of D by satisfying two criteria,

- Maximality
    - ∀ p, q if p ∈ C and if q is density-reachable from p, then also q ∈ C
- Connectivity
    - ∀ p, q ∈ C, p and q are density-connected [33]

5.6 E-news summarization module

The design of the e-news summarization module which depicts how each phase of the module interacts with other phases is shown below.

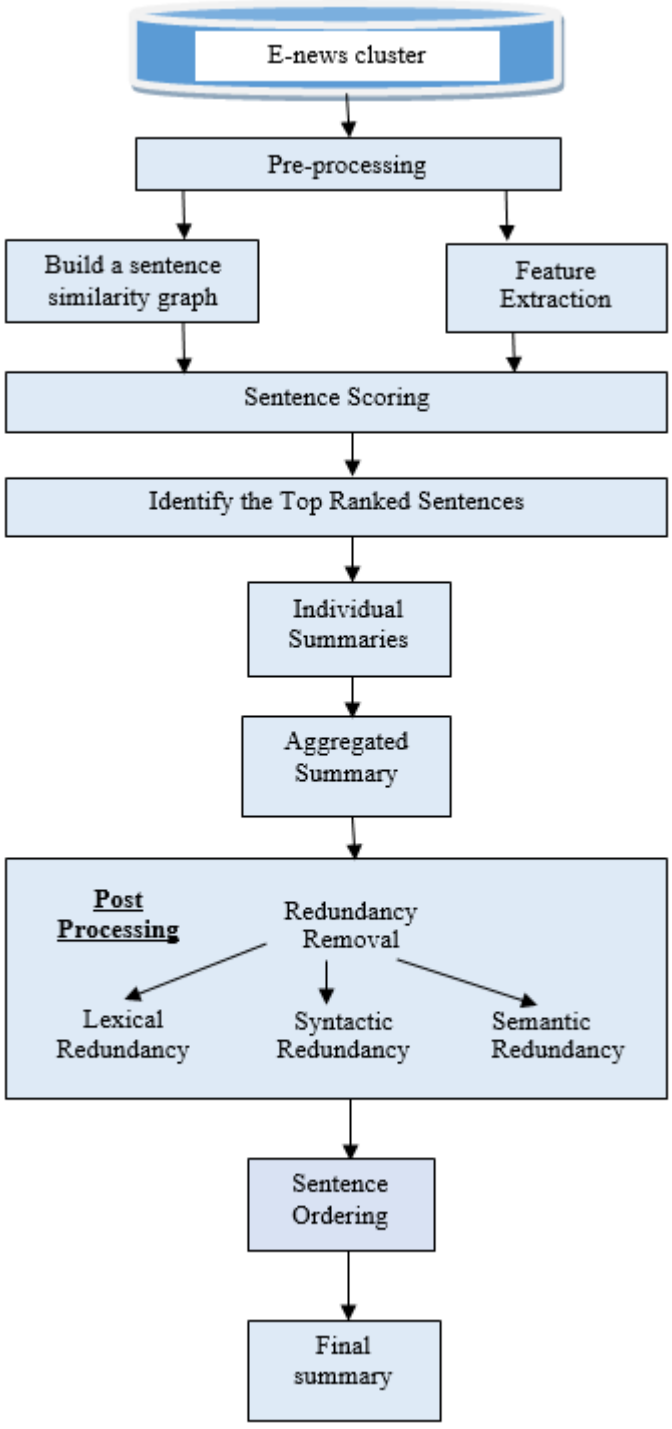

Figure 5.8: Design of the e-news summarization component

The design of the summarization module consists of several phases as illustrated in the diagram above. The design details of each phase are described below.

5.6.1 Pre-processing

Each e-news article in the cluster needs to be pre-processed before performing any type of further processing. Pre-processing involves certain techniques like sentence boundary identification, word tokenization, stop word elimination and stemming/ lemmatization, POS tagging etc. First the sentence boundaries from the text need to be identified and the sentence tokenizer named "sent_tokenize" tool provided by NLTK was used here. Then the sentences need to be tokenized into words since processing smaller units of word tokens is much more efficient than processing whole sentences. The word tokenizer named "WhitespaceTokenizer" from NLTK was used for that purpose. Then the stop words need to be eliminated from the extracted tokens because these stop words do not possess any semantic meaning about the text and do not make any sense in the further processing. So keeping these stop words in the memory is an extra burden which reduces the performance too and hence they need to be eliminated. There is an in-built stop words list for English language in NLTK and we can also define our own stop words in the list. After the removal of stop words, stemming needs to be performed in order to extract the main stem of the word. For e.g.: if we consider the words "computerize", " computerization", "compute", " computed" all of them have been derived from the word "computer". So the ultimate goal of stemming is to extract this main stem word "computer". Lemmatization refers to extracting the stem word with the use of a lexical dictionary like WordNet and morphological analysis of words which cannot be easily performed by just removing the affixes from words like in stemming [6]. For e.g. if we want to extract the stem word of "sung" we cannot remove affixes and find the stem word. So here we need to refer to a lexical vocabulary and take the stem word of "sung" as "sing". So that it is easy to handle only the main stem of a word list derived from one particular stem rather than handling all of them together. For this purpose one of the most popular stemming algorithms, the Porter stemming algorithm is used. Then POS tagging also needs to be performed which assigns the part of speech tags for each word. Then only the words which are tagged as nouns, verbs and proper nouns are used by the summarization module considering those terms are containing the key information.

5.6.2 Sentence scoring

It's very important to identify the most salient sentences from the original documents to be included as summary sentences. Therefore, a sentence score is given to each sentence based on a hybrid model. The hybrid model was developed by using the following methods combined together.

- A graph based approach – TextRank algorithm
- A feature based approach

5.6.2.1 TextRank algorithm

The graph based approach used for sentence scoring is the TextRank algorithm. The TextRank algorithm first builds a sentence similarity graph for the original document based on the similarities between the sentences. The sentence similarities are measured based on some similarity measure. After evaluating various similarity measures available in the context, cosine similarity measure was found to be more effective than others and therefore cosine similarity measure was used to measure the sentence similarities. Since stop words like 'a', 'the', 'an' are more frequently occurring in a document, those stop words get a high importance value which add some noise. Therefore, in order to remove that noise each word needs to be normalized with TF-IDF [37]. Then a similarity matrix between sentences is constructed and finally each sentence is assigned a sentence score based on the pageRank algorithm [49].

5.6.2.1.1 Sentence similarity graph

A sentence similarity graph illustrates how the sentences in the text are inter-related to each other based on the similarities between the sentences. The nodes in the graph represent individual sentences in the text and the edges represent the similarity between the sentences. Here it's important to consider how the sentence similarities are defined when building a sentence similarity graph. There are number of similarity measures available in the context to find the similarities between the sentences like cosine similarity, bag-of-words model, jaccard similarity, Euclidean distance etc. [48]. A sentence similarity graph indicates how much of common information each sentence in

the document has with other sentences. Different similarity measures evaluated to find the best similarity measure when building the sentence similarity graph are described below.

5.6.2.1.2 Bag-of-words model

The bag-of-words model first generates a vocabulary of unigrams for the entire document which contains only the unique words of the document. Then based on the presence of these individual words in the sentences, sentence vectors are created for each and every sentence in the document. Usually these sentence vectors are sparse vectors which contain lot of zero entries [48]. To overcome this problem they need to be normalized by using TF-IDF count or any other measure.

5.6.2.1.3 Cosine similarity

The news articles in a cluster may be represented by a cosine similarity matrix where each entry in the matrix is the similarity between the corresponding sentence pair. The cosine similarity measure between sentences calculates the cosine of the angle between the two sentence vectors. It can range from -1 to +1 where the value is +1 when the two sentence vectors are exactly the same and -1 when the two sentence vectors are exactly opposite of each other [37]. It's required to build the sentence vectors before calculating the cosine similarity between sentences. The sentence vectors are built by applying word counts appearing in the sentence for each unique word in the vocabulary. Hence the sentences are represented as a weighted vector of TF-IDF. The cosine similarity is calculated by using the formula;

$$\text{Cosine similarity}(A, B) = \frac{A.B}{|A||B|}$$

Where A and B represent the sentence vectors.

5.6.2.1.4 Jaccard similarity

The jaccard similarity finds the similarity between two sentences based on the overlaps of words or the words in common between the two sentences and the jaccard similarity between sentence A and B is calculated by:

$$Jaccard\ similarity(A,B) = \frac{A \cap B}{A \cup B}$$

After evaluating each of these similarity measures to find the sentence similarities, it was proven that the cosine similarity gives the highest accuracy rate compared with the others.

5.6.2.1.5 PageRank algorithm

PageRank which is the underlying technology behind the Google search engine can be used to assign a prestige score to each sentence in the sentence similarity graph. Its base concept is "The linked sentence is good, much more if it from many linked sentences" [49]. In PageRank, the score of a sentence is determined depending on the number of other sentences that link to that sentence as well as the individual scores of the linking sentences. A high PageRank score is gained if the sentence has more linking sentences and if the sentence is linked to sentences with high PageRank score. The PageRank score of a sentence 'A' can be computed by using:

$$\mathrm{PR(A)} = (1 - \mathrm{d}) + \mathrm{d}\frac{\mathrm{PR(T_1)}}{\mathrm{C(T_1)}} + \cdots + \frac{\mathrm{PR(T_n)}}{\mathrm{C(T_n)}}$$

where $T_1, T_2, \ldots, T_n$ are sentences that link to the sentence 'A', C (T_i) is the number of linking sentences to the sentence T_i and d is the damping factor which can be set between 0 and 1 and usually valued as 0.85.

5.6.2.2 Feature extraction based approach

A sentence can be represented as a set of features to give it an importance score. The features used here are the sentence length, sentence position, presence of title words, numerical literals count, nouns count, verbs count, named entities count, keyword frequencies (Thematic word Feature) etc. Based on the presence of these features we can predict whether the sentence is important to be included in the summary or not.

Sentence length feature: Sentences which are too short are considered as not important to be included in the summary since they do not contain significant information about the document whereas too long sentences are also not considered as important to be included in the summary since they are very descriptive and therefore not suited as summary sentences. Therefore a score between 0 and 1 is assigned based on how close sentences' length is to the ideal length.

$$\text{Sentence length score} = \frac{\text{No. of words in the sentence}}{\text{No. of words in the longest sentence}}$$

Sentence position feature: The first sentence and the last sentence of a document, i.e. the introductory sentence and the concluding sentence are considered as important to be included in the summary.

$$\text{Sentence position score} = \frac{\text{Index position of the sentence in the document}}{\text{Total no. of sentences in the document}}$$

Title words feature: If the sentences contain words from the headline, then those sentences are said to be important to be included in the summary. So a score between 0 and 1 is assigned based on the percentage of words common to the news headline.

$$\text{Title word score} = \frac{\text{No. of title words in the sentence}}{\text{No. of words in the sentence}}$$

Numerical literals count, Nouns count, Verbs count, and Named entities count: The importance of the sentences are given as a percentage of the presence of the numerical values, nouns, verbs, and named entities.

$$\text{Numerical literal, NN, VB, NE score} = \frac{\text{Total no. of numerical values, NN, VB, NE in the sentence}}{\text{Total no. of words in the sentence}}$$

Thematic word Feature: If the sentences contain frequent words then they are said to be highly relevant to the document. TF-IDF is such a widely used measure where the term frequency implies how often a certain word is appeared in a sentence where the inverse document frequency implies how often that word appears in a given set of documents. Based on the frequencies certain words may appear in a text, the most important sentences from the text can be decided [38].

$$\text{TF} = \frac{\text{Total appearance of word in the document}}{\text{Total no. of words in the document}}$$

$$\text{IDF} = \log\left(\frac{\text{Total no. of documents}}{\text{No. of documents containing the word}}\right)$$

$$\text{TF} - \text{IDF} = \text{TF} * \text{IDF}$$

5.6.2.2.1 Normalization scheme

The requirement of a normalization scheme is highly applicable here in the feature based approach since the individual feature scores of the sentences range from very low values to very high values and hence the values are not evenly distributed. Therefore, normalization by sigmoid generates feature scores of the sentences ranging from 0 to 1 which preserves an evenly distributed distribution. A very significant observation here is that it's very effective when the feature values are normalized when compared to the no normalization. A normalization scheme was used for the generated feature values; some of them based on a sigmoid function and others based on the number of sentences in the document [38]. When taking the sigmoid function as the normalization scheme rather than just taking the sentence length, the accuracy rate was high.

- Title word feature, numerical literals count, named entity count, nouns count, verbs count and thematic word feature: These features were normalized by the sigmoid function. There may be a chance of neglecting some small length sentences which are important in summary generation if there is no normalization scheme.
- Sentence Position: This feature is normalized by the number of sentences in the document. This gives us the % position of the sentence in the document.

5.6.2.2.2 Weighted average score

Initial weights need to be assigned to the feature vector based on the importance of each feature when generating the summary. Then based on the presence of the features in the feature set in each sentence a weighted average score is assigned to each sentence by accumulating all the feature scores. The sentences are scored as a weighted average score computed as [38]:

$$\text{Weighted average score of the sentence} = \sum \text{wf}$$

where w is the weight assigned for each feature and f is the feature score of that sentence. Therefore not all text features are treated with same level of importance as some of the features have more importance or weight and some have less. Then the final sentence scores are computed as an average score from the scores taken from the two methods graph based method and the feature based method.

5.6.3 Summary sentences selection for summary generation

After computing the final sentence scores they are arranged in the descending order in the sentence scores and the heap queue algorithm is used for that where the top ranked sentences are popped out of the heap to generate the summary. The compression rate decides how many sentences need to be extracted to generate the final summary. The compression rate is usually 30% of the original document and summary sentences are selected which covers that 30%. Here individual summaries are generated for each news article in a cluster and then those individual summaries are aggregated together to form

an intermediate level of summary. That is an extractive summary which extracts key information from the original set of documents themselves.

5.6.4 Post processing

After generating the intermediate level of summary by compiling all the individual summaries in the previous step, there are some post processing tasks need to be performed. Redundancy removal and sentence ordering are such most non-trivial post processing tasks need to be performed in order to make the final summary more readable and coherent.

5.6.4.1 Redundancy removal

Since sentences are extracted from multiple documents redundant sentences may include in the summary. These redundant sentences need to be removed by identifying similar sentences. The similarity between sentences are identified from three major perspectives namely the syntactic similarity, lexical similarity and semantic similarity. The syntactic similarity identifies whether the sentences follow the same structure or not. The lexical similarity measures the similarity between sentences as a measure of overlap of words between sentences. Some sentences may describe about the same thing even though they use different words and phrases which needs to be identified by finding the semantic similarity.

5.6.6.1.1 Lexical similarity

Lexical similarity between sentences are identified as overlap of words between the sentences. In order to identify the overlap of words between sentences the jaccard similarity measure is used. Then it is judged that the sentences are lexically dissimilar by the condition that the jaccard similarity between the two sentences is less than the threshold value. The jaccard similarity measures the similarity between two sentences as the amount of word overlap normalized by the union of the sets of words present in the two sentences.

5.6.6.1.2 Syntactic similarity

In order to find the syntactical similarities between sentences, the n-gram models are used. First 2-gram models are constructed for the two sentences and the dice coefficient between them is computed by using the formula;

$$Dice\ coefficient = \frac{2|X \cap Y|}{|X| + |Y|}$$

Where X and Y are separate sets of bi-grams modelled for each sentence in the sentence pair. If a non-zero value is obtained as the dice coefficient, it's said that there is some kind of a syntactical relationship between the two sentences. Although the sentences may contain some syntactical similarity, the meaning of the sentences may be not the same. So we can't just remove the syntactically similar sentences without finding the semantic similarities between them.

5.6.6.1.3 Semantic similarity

Semantic similarities between sentences are identified based on the two methods namely:

- WordNet based semantic similarity
- Word2vec based semantic similarity

5.6.6.1.4 WordNet based semantic similarity

Wordnet lexical dictionary based semantic similarity identifies the semantic similarities between the sentences based on the synsets of each word in the sentences. First the sentences are tokenized into words and then wordNet part of speech tags are assigned for each token which establish the connection between four part of speech tags namely noun, verb, adjective or adverb. Then sysnsets for each word in the sentence pair are found which represent a specific meaning of a word. It includes the word, its explanation, and its synonyms [50]. Finally the semantic similarity between sentences are computed based on the semantic relatedness of the pairs of synsets by using an edge

counting method like path distance. The path distance is a score denoting the number of edges in the shortest path.

5.6.6.1.5 Word2vec based semantic similarity

Word2vec model is based on the concept of word embedding where a word embedding is a type of word representation where the words with the similar meaning to have the same representation [51]. The word2vec model is trained by using an artificial neural network based on the skip-gram model and the Continuous Bag of Words (CBOW) model. The skip-gram model predicts the context of a given word and the CBOW model predicts a number of words based on the parameter of window size when the context is given. The pre-trained neural network by Google News was used here which generates a word embedding of 300 features represented as a vector of real values. Therefore, the word embeddings with approximately similar near real values for the vectors are considered as semantically similar words. Then the cosine distances between the word embeddings are computed to find the semantic similarity between the two sentences. The figure 5.17 depicts that the words with approximately similar values for the word embeddings which carry the same semantic meaning are grouped together.

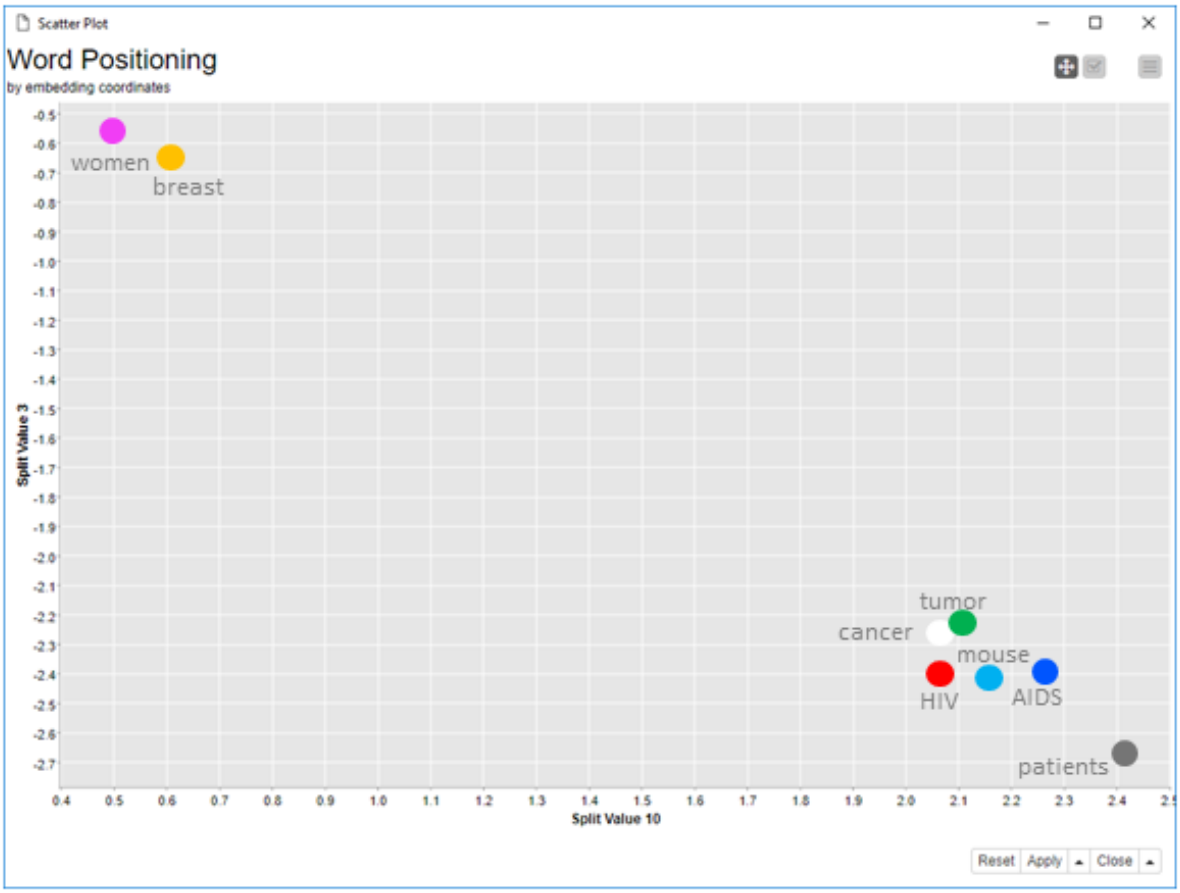

Figure 5.9: Semantic similarities between words based on word2vec model

Finally the semantic similarity score between sentences is computed as an average of the two methods, the wordNet based method and the word2vec based method. If that score is greater than the threshold value, then it's considered that the two sentences are semantically similar and thereby remove those redundant sentences.

5.6.4.2 Sentence ordering

A proper sentence ordering algorithm needs to be applied to make the final summary more coherent. Therefore, the final summary sentences selected after the redundancy removal process are arranged in the proper coherent order by using sequence matching. The sequence ratio or the coherent score which denotes the number of primitive operations namely the insertion, substitution and deletion need to be performed in order to make one sentence exactly similar with the other sentence is computed in terms of the Levenshtein edit distance. The sequence ratios are computed in both the directions

where the sequence ratio to make sentence 1 similar with sentence 2 and the sequence ratio to make the sentence 2 similar with sentence 1. Then the sentences are ordered in the manner that preserves the high sequence ratio out of the two.

5.7 Recommendation module

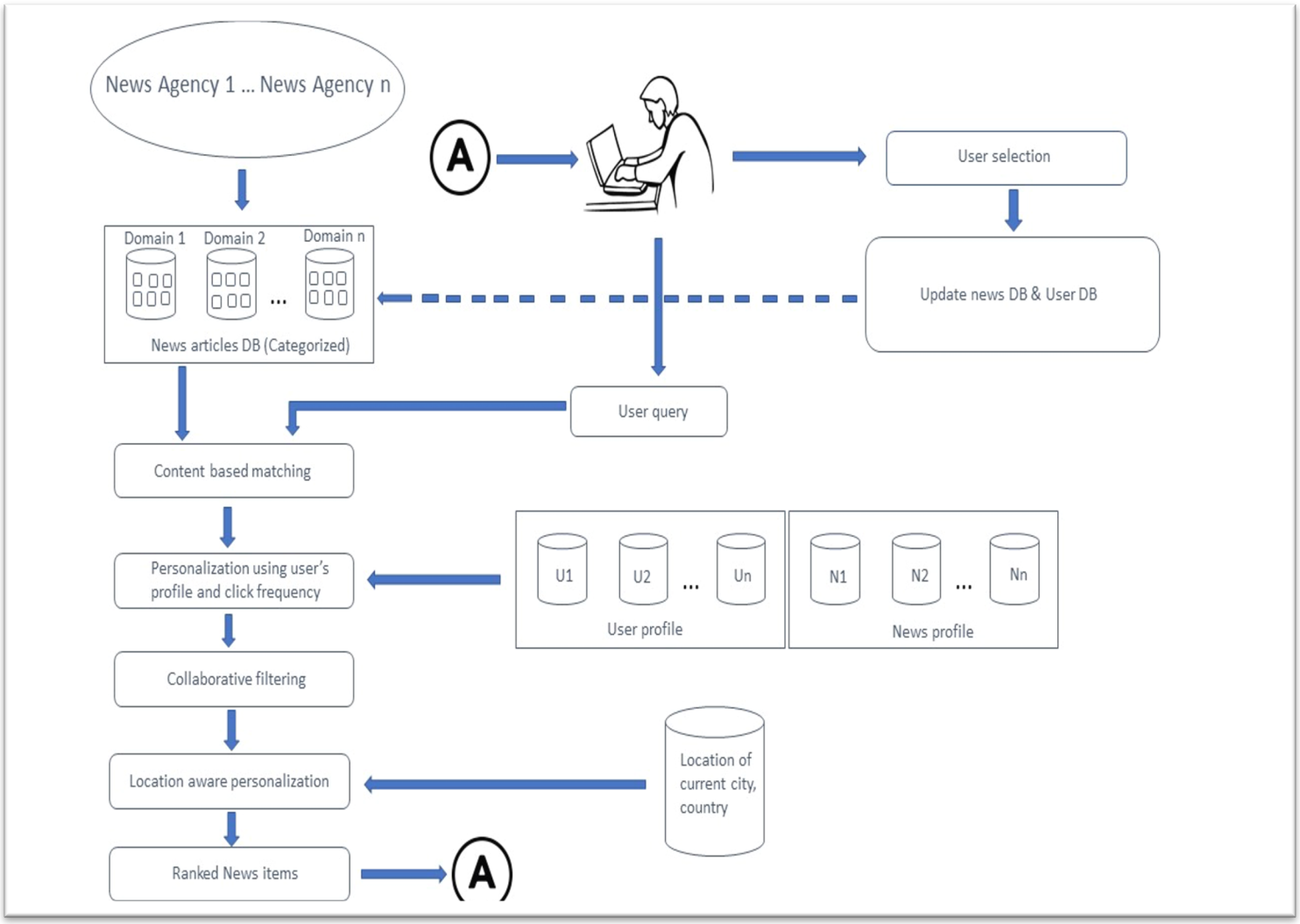

Figure 5.10 Design of the recommendation module

Traditional recommendation systems consist only Content based filtering or Collaborative filtering or both methods for recommending news for the users. That's not reliable and not as much as accurate for a recommendation engine. So, we propose a Hybrid Recommendation System, consist with content-based filtering, Collaborative filtering and Location aware personalization with user preferences. User profile is used for tracking the user's long-term interest and short-term interest. Individual user profile can be divided into two parts, those are static user profile and dynamic user profile. Static user profiles are used for storing users' long-term interests and dynamic user profiles are used to store users' short-term interests. Static profiles are constructed

during user signup process and dynamic profiles are created when they are using the system.

Due to vast content of information provided by the online mass media, it is necessary to have a powerful database such as HBase. HBase is a NoSql database work as a fast and real-time data provider. It's important to have accurate and real-time news update for user to read information. Click frequency of the article i and category j relationship can be shown as follows.

$clicks_i$ = Total number of clicks made by one user on article *i*:

$$f_{click}(i,j) = \frac{\sum_{users} clicks_i}{\sum_{users} \sum_{i=1}^{n} clicks_i}$$

Users input query is matched with the snipped stored inside of the article database. When user searches on some news, and try to get the article, it helps to provide the best content-based recommendation for the user. Related users are identified using User-based collaborative filtering and similarities between articles are calculated using Item-based filtering algorithm [43].

5.7.1 Location aware personalization

Current location of the user can be used to recommend interesting news to the user. The traditional method for imparting location awareness is by using the city name as the key term to rank the news. But, mobile users are always interested in the happening of the neighboring cities. So, the entire location parameter (state/country) is split into hexagonal regions. Details of hexagonal regions pertaining to a latitude, longitude, city, state and country are maintained in region database. Based on the percentage coverage in region database, the city/state/country news can be reported to the user. Care is also taken to maintain information on whether the news is location specific or not in snippet

$$f_{loc}(i,j,loc) = cov_{loc} * f_{click}(i,j)$$

database. Location awareness factor is got by including the coverage criteria in click frequency factor as shown below:

Where $\mathbf{f_{loc}(i,j,loc)}$ is the location awareness factor for article 'i' in category 'j'.

$\mathbf{COV_{loc}}$ is the area coverage of the city in the hexagonal cell. It is given by the formula;

$$\frac{area_{loc\ i}}{\sum_{i=1(hexagon)}^{n} area_{loc\ i}}$$

Where $\mathbf{area_{loci}}$ is the area of *loci* (city *i*) in the hexagon. The denominator gives the area of other cities in the hexagon. All the location details of the users are stored inside of the location database. So, system can provide recommendation according to the user's current location [46].

5.8 Summary

This chapter discusses the basic components of the project and how each components interact. It includes the design methodology of our system and the functions we designed for each module.

Chapter 6

6. Implementation

6.1 Introduction

In this chapter we have described the implementations which we have done according to the analysis and design described in previous chapters. How technology is used to implement the solution is described further in this chapter. This chapter provides details of implementation of each module that is stated in the design section step wise.

6.2 Implementation of Extraction module

The implementation is done using python programming language. In extraction e-news articles should be extracted from different e-news web portals. At the beginning the system needs URLs or RSS feeds of predefined web sites that. As a first step, seed URLs and RSS feeds will be given to the system as a JSON file. The format of the JSON file shows below. This format helps to add or remove new website easily.

```
{
  "Yahoo": {
    "rss": "https://www.yahoo.com/news/rss/",
    "link": "https://www.yahoo.com/news/"
  },
  "cnn": {
    "link": "http://edition.cnn.com/"
  },
  "bbc": {
    "rss": "http://feeds.bbci.co.uk/news/rss.xml",
    "link": "http://www.bbc.com/"
  },
  "theguardian": {
    "rss": "https://www.theguardian.com/uk/rss",
    "link": "https://www.theguardian.com/international"
  },
  "adaderana": {
    "rss": "http://www.adaderana.lk/rss.php",
    "link": "http://www.adaderana.lk/"

  },
  "newsfirst": {
    "rss": "https://www.newsfirst.lk/feed/",
    "link": "https://www.newsfirst.lk/"
  },
  "dailynews": {
    "rss": "http://www.dailynews.lk/rss.xml",
    "link": "http://dailynews.lk/"
  }
}
```


Figure 6.1: Arrangement of the JSON Object

Initialize a data object which stores extracted news items.

```
data = {} # store our scraped data ine
data['newspapers'] = {}
```

Figure 6.2: Creation of the data object

Newswebsites.json file contains the URLs and RSS feed of predefined e-news websites. So then need to open the JSON file using a python script

```
# Loads the JSON files with news sites
with open('newswebsites.json') as data_file:
    e_news_websites = json.load(data_file)
print(e_news_websites)
```

Figure 6.3: Open JSON file with News Sites

Iterate through JSON file and checking weather rss feed is provides or not. If it is available, use FeedPaser to load RSS feeds. Then build the structure for the data by constructing a dictionary *newsPaper*.

if 'rss' in value:

```
# Iterate through each news sites
for e_news_site, value in e_news_websites.items():
    # If a RSS link is provided in the JSON file, this will be the first choice.
    # Reason for this is that, RSS feeds often give more consistent and correct data
    # If you do not want to scrape from the RSS-feed,
    # just leave the RSS attr empty in the JSON file.
    if 'rss' in value:
        d = fp.parse(value['rss'])
        print("Downloading articles from ", e_news_site)
        newsPaper = {
            "rss": value['rss'],
            "link": value['link'],
            "articles": []
        }
```

Figure 6.4: If RSS Feed is available, use FeedPaser

List of links to e-news articles taken from the RSS-feed is the variable d. It will loop through for each entry. Check publish date field to get consistent data. If publish date field is not available the entry will be discarded. An article dictionary is created to store data for every e-news item.

```
for entry in d.entries:
    # Check if publish date is provided, if no the article is skipped.
    # This is done to keep consistency in the data and to keep
    # the script from crashing.
    if hasattr(entry, 'published'):
        if count > LIMIT:
            break
        article = {}
        article['link'] = entry.link
        date = entry.published_parsed
        article['published'] = datetime.fromtimestamp(mktime(date)).isoformat()
        try:
            content = Article(entry.link)
            content.download()
            content.parse()
```

Figure 6.5: Check Published Date

Newspaper library is used for scrape the content of the links. Try block will be used to avoid failures.

```
try:
    content = Article(entry.link)
    content.download()
    content.parse()
except Exception as e:
    # If the download for some reason fails (ex. 404)
    # the script will continue downloading
    # the next article.
    print(e)
    print("continuing...")
    continue
```

Figure 6.6: Article Downloading and Parsing

Title, text and URL of the articles will be stored in article object than it added into the dictionary.

```
#Article title
article['title'] = content.title
#Article text
article['text'] = content.text
#Article url
article['link'] = content.url
newsPaper['articles'].append(article)
print(article)

cur_Date = datetime.now().strftime("%Y_%m_%d_%H_%M_%S.%f")
save_path = 'C:/Users/jayanid/PycharmProjects/FYP_G/e-News'
completeName = os.path.join(save_path,cur_Date + '.txt')
with open(completeName, 'w',encoding="utf-8") as the_file:
    the_file.write(article['link'])
    the_file.write("\n")
    the_file.write(article['title'])
    the_file.write("\n")
    the_file.write(article['text'])

print(count, "articles downloaded from", e_news_site, ", url: ", entry.link)
count = count + 1
```

Figure 6.7: Get Data using RSS Feeds

If there isn't rss feed than else block will be executed. In here the articles will be scraped directly from the e-news web site.

```
paper = newspaper.build(value['link'], memoize_articles=False)
```

Figure 6.8: Use URLs to scrape data

It builds the list of e-news articles found on the first page of the website. And download the news items using newspaper library than parse those downloaded web pages to extract the content.

```
try:
    content.download()
    content.parse()
except Exception as e:
    print(e)
    print("continuing...")
    continue
```

Figure 6.9: Scrape Data using URLs

If publish date is none article will be skipped. It will maintain the consistence of the data. If it found 10 articles without having published date, the web page will be skipped.

```
print(count, " Article has date of type None...")
noneTypeCount = noneTypeCount + 1
if noneTypeCount > 10:
    print("Too many noneType dates, aborting...")
    noneTypeCount = 0
    break
count = count + 1
continue
```

Figure 6.10: Use URLs to scrape data

Then download and parse the web pages. Then extract data using newspaper library and store them.

```
        article = {}
        article['title'] = content.title
        print(article['title'])
        article['text'] = content.text
        article['link'] = content.url
        #article['published'] = content.publish_date.isoformat()
        newsPaper['articles'].append(article)

        cur_Date = datetime.now().strftime("%Y_%m_%d_%H_%M_%S.%f")
        save_path = 'C:/Users/jayanid/PycharmProjects/FYP_G/e-News'
        completeName = os.path.join(save_path,cur_Date + '.txt')
        with open(completeName, 'w', encoding="utf-8") as the_file:
            the_file.write(article['link'])
            the_file.write("\n")
            the_file.write(article['title'])
            the_file.write("\n")
            the_file.write(article['text'])


        print(count, "articles downloaded from",
              e_news_site, " using newspaper, url: ", content.url)
        count = count + 1
        noneTypeCount = 0
count = 1
data['newspapers'][e_news_site] = newsPaper
```

Figure 6.11: Store data gathered from URLs to scrape data

Then save all scarped e-new article and Meta data into JSON file.

```
# Finally it saves the articles as a JSON-file.
try:
    with open('scraped_articles.json', 'w') as outfile:
        json.dump(data, outfile)
except Exception as e: print(e)
```

Figure 6.12: Save Data into JSON file

6.3 Implementation of Classification module

Before classification, preprocessing techniques will be applied to the data. Such as removing punctuations, stemming, transfer cases.

```
import re

def clean_str(string):

    string = re.sub(r"\'s", "", string)
    string = re.sub(r"\'ve", "", string)
    string = re.sub(r"n\'t", "", string)
    string = re.sub(r"\'re", "", string)
    string = re.sub(r"\'d", "", string)
    string = re.sub(r"\'ll", "", string)
    string = re.sub(r",", "", string)
    string = re.sub(r"!", " ! ", string)
    string = re.sub(r"\(", "", string)
    string = re.sub(r"\)", "", string)
    string = re.sub(r"\?", "", string)
    string = re.sub(r"'", "", string)
    string = re.sub(r"[^A-Za-z0-9(),!?\'\`]", " ", string)
    string = re.sub(r"[0-9]\w+|[0-9]","", string)
    string = re.sub(r"\s{2,}", " ", string)
    return string.strip().lower()
```

Figure 6.13: e-news Pre-processing

```
for index,value in enumerate(x):
    print("processing data:",index)
    x[index] = ' '.join([Word(word).lemmatize()
                         for word in clean_str(value).split()])
```

Figure 6.14: e-news articles lemmatization

Data will be read by the .csv file. Then, news will be added to the x array and type of the news will be added to the Y array. To extract features we use BBC news dataset and use TF-IDF feature extraction method.

```
data = pd.read_csv('C:/Users/jayanid/PycharmProjects/FYP_G/dataset.csv')
x = data['news'].tolist()
y = data['type'].tolist()
```

Figure 6.15: Read BBC data set

Then create TF_IDF vector from dataset to feed classifiers. Defined min dif as 2. It will ignore terms that appear in less than 2 documents. Fit_transform learn the vocabulary dictionary and return term-document matrix.

```
vect = TfidfVectorizer(stop_words='english',min_df=2)
X = vect.fit_transform(x)
Y = np.array(y)
print("no of features extracted:",X.shape[1])
```

Figure 6.16: TF-IDF Vector Creation

Dataset divides into two as train data and testing data. 20% of data as testing data.

```
X_train, X_test, y_train, y_test = train_test_split\
    (X, Y, test_size=0.20, random_state=42)
```

Figure 6.17: Split Data set Into Training and Testing Datasets

Then use Random Forest Classifier to classify e-news items.

```
model_RF = RandomForestClassifier(n_estimators=300, max_depth=150,n_jobs=1)\
    .fit(X_train, y_train)
y_pred_RF = model_RF.predict(X_test)
c_mat_RF = confusion_matrix(y_test,y_pred_RF)
kappa_RF = cohen_kappa_score(y_test,y_pred_RF)
acc_RF = accuracy_score(y_test,y_pred_RF)
print("Confusion Matrix RF:\n", c_mat_RF)
print("\nAccuracy: ",acc_RF)
```

Figure 6.18: Random Forest Classifier

Then save trained model as pickle model

```
# Save to file in the current working directory
pkl_filename = "pickle_model_RF.pkl"
with open(pkl_filename, 'wb') as file:
    pickle.dump(model_RF, file)
```

Figure 6.19: Save Trained Random Forest Classifier

Multinomial Naive Bayes classifier.

```
model_MNB= MultinomialNB().fit(X_train, y_train)
y_pred_MNB = model_MNB.predict(X_test)
c_mat_MNB = confusion_matrix(y_test,y_pred_MNB)
acc_MNB = accuracy_score(y_test,y_pred_MNB)
print("Confusion Matrix MNB:\n", c_mat_MNB)
print("\nAccuracy: ",acc_MNB)
```

Figure 6.20: Multinomial Naïve Bayes Classifier

Support Vector Machine classifier.

```
model_SVM = SVC(kernel='linear',probability=True).fit(X_train, y_train)
y_pred_SVM = model_SVM.predict(X_test)
c_mat_SVM = confusion_matrix(y_test,y_pred_SVM)
kappa_SVM = cohen_kappa_score(y_test,y_pred_SVM)
acc_SVM = accuracy_score(y_test,y_pred_SVM)
```

Figure 6.21: Support Vector Machine Classifier

Ensemble classifier is created using individual classifiers such as Naive Bayes, Support Vector Machine and Random Forest classifiers.

In here hard voting method is used to ensemble classifiers. Majority rule voting will be used in hard Voting Classifier.

```
#If 'hard', uses predicted class labels for majority rule voting.
eclf1 = VotingClassifier(estimators=[('mnb', model_MNB), ('rf', model_RF),
                                     ('svm', model_SVM)], voting='hard')
eclf1 = eclf1.fit(X_train, y_train)
y_pred_eclf1 = eclf1.predict(X_test)


c_mat_eclf1 = confusion_matrix(y_test,y_pred_eclf1)
kappa_eclf1 = cohen_kappa_score(y_test,y_pred_eclf1)
acc_eclf1 = accuracy_score(y_test,y_pred_eclf1)
print("Confusion Matrix eclf1:\n", c_mat_eclf1)
print("\nAccuracy: ",acc_eclf1)
```

Figure 6.22: Hard Voting Method

The soft voting method predicts the e-news class label based on the sum of the predicted probabilities of individual classifies.

```
estimators = [pickle_model_MNB,pickle_model_RF,pickle_model_SVM]


def _collect_probas(X):
    return np.asarray([clf.predict_proba(X_test_MNB) for clf in estimators])


def _predict_proba(X):
    avg = np.average(_collect_probas(X_test_MNB), axis=0)
    return avg

prob=_predict_proba(X_test_MNB)
print(prob)
```

Figure 6.23: Weighted Average Method

```
for z in prob:
    p=max(z)
    print(p)
    if p<0.4:
        print("other")
        y.append("other")
    else:
        q=np.argmax(z)
        if q == 0:
            print("business")
            y.append("business")

        elif q == 1:
            print("entertainment")
            y.append("entertainment")

        elif q == 2:
            print("political")
            y.append("political")

        elif q == 3:
            print("sport")
            y.append("sport")

        else:
            print("tech")
            y.append("tech")

    print(np.argmax(z))
print(y)
```

Figure 6.24: Predict Class Labels

Could use weights parameter to adjust the contribution of the individual classifiers.

```
eclf3 = VotingClassifier(estimators=[('mnb', model_MNB),
                                     ('rf', model_RF),
                                     ('svm', model_SVM)],
                         voting='soft', weights=[2,1,1], flatten_transform=True)
eclf3 = eclf3.fit(X_train, y_train)
y_pred_eclf3 = eclf3.predict(X_test)
c_mat_eclf3 = confusion_matrix(y_test,y_pred_eclf3)
kappa_eclf3 = cohen_kappa_score(y_test,y_pred_eclf3)
acc_eclf3 = accuracy_score(y_test,y_pred_eclf3)
print("\nAccuracy: ",acc_eclf3)
```

Figure 6.25: Ensemble classifier considering weight parameter

6.4 Implementation of the e-news Aggregation Module

The implementation of e-news aggregation can be divided into three parts. Those are preprocessing, features extraction and clustering. Python nltk, sklearn, gensim packages have used for implementing the e-news aggregation module.

6.4.1 Preprocessing

As the first step of preprocessing stop words and punctuations are removed.

```
stopWordsAndPunctuatoin = set(stopwords.words('english')).union(set('\n'
    '\r''\t'"\'"'\u2013''\u2014''\u00A9''!''"' '#' '$' '%' '&' '•' '‘' '’'
     '“'    '”' '(' ')' '*' '+' ',' '-' '.' '/' ':' ';''<' '=' '>' '?' '@'
    '\[' '\\' '\]' '^' '_' '`' '{' '|' '}' '~')).union(set(["--","--", "''", "``" ,"--"]))
```

Figure 6.26: Stop words elimination

After that stemming is applied.

```
ps = PorterStemmer()
if w not in stopWordsAndPunctuatoin:
    wordsFiltered.append(ps.stem(w))
```

Figure 6.27: Porter's stemming algorithm

6.4.2 Features Extraction

For features extraction, three different feature models were implemented. Those are LDA model, Doc2vec model and Tf-idf model. But Tf-idf feature model gave higher accuracy than other two models. LDA model was implemented using gensim python library.

```
dictionary = corpora.Dictionary(preprocessedDocs.values())
lda = models.LdaModel(bow_corpus.values(), num_topics=no_topics,
                      id2word=dictionary, #   mapping word IDs to words
                      update_every=1,     #   update topics after every (update_every*chunksize) docu
                      chunksize=50,       #   number of documents load to the memory at a time
                      passes=10)          #   number of iterations
```

Figure 6.28: LDA model

Doc2vec model was implemented using gensim python library.

```
class LabeledLineSentence(object):
    def __init__(self, documents):
        self.documents = documents
    def __iter__(self):
        for key, value in self.documents.items():
            yield LabeledSentence(value, [str(key)])

def documentEmbedding(preprocessedDocs):
    it = LabeledLineSentence(preprocessedDocs)

    model = Doc2Vec(size=300, window=10, min_count=1, alpha=0.025, min_alpha=0.025)
    model.build_vocab(it)
    for epoch in range(10):
        model.train(it,total_examples=model.corpus_count,epochs=model.iter)
        model.alpha -= 0.002
        model.min_alpha = model.alpha
        model.train(it, total_examples=model.corpus_count, epochs=model.iter)
```

Figure 6.29: Doc2vec model

Tf-idf model was implemented using sklearn python library.

```
vectorizer = TfidfVectorizer()
vector = vectorizer.fit_transform(preProcessedDocs.values())
```

Figure 6.30: Tf-idf model

6.4.3 Clustering

For clustering, three different clustering algorithms were implemented. Those are K-means algorithm, Affinity propagation algorithm and DBSCAN algorithm. But DBSCAN algorithm gave higher accuracy than other two algorithms.

K-means algorithm, Affinity propagation algorithm and DBSCAN algorithm were implemented using sklearn python library.

```
db = KMeans(n_clusters=no_ofClusters, init='k-means++', max_iter=100, n_init=1)
db.fit_predict(vector)
```

Figure 6.31: k-means clustering algorithm

```
db = AffinityPropagation(affinity='euclidean', damping=0.5)
db.fit_predict(vector)
```

Figure 6.32: Affinity propagation algorithm

```
db=DBSCAN(eps=1.15, min_samples=2, metric='euclidean',algorithm='auto', n_jobs=1)
db.fit_predict(vector)
```

Figure 6.33: DBSCAN algorithm

6.5 Implementation of the summarization module

In the summary generation process the most salient sentences are extracted by assigning a sentence score to each and every sentence of the input documents. A hybrid approach is applied to assign sentence scores; a feature based approach and a graph based approach combined together as described in the previous chapters.

6.5.1 Implementation of the graph based approach for sentence scoring

The TextRank algorithm which is one of the famous graph based approaches was implemented here. It first constructs a sentence similarity graph by considering the sentence similarities. After evaluating various similarity measures described in earlier chapters, the cosine similarity measure was used for the implementation purposes since it showed higher accuracy rate. Before computing the cosine similarities between sentences we need to generate sentence vectors. Sentence vectors are generated by computing feature weights (e.g.: Term frequency).

```
def sentence_similarity(sent1, sent2, stopwords=None):
    if stopwords is None:
        stopwords = []

    sent1 = [w.lower() for w in sent1]
    sent2 = [w.lower() for w in sent2]

    all_words = list(set(sent1 + sent2))

    vector1 = [0] * len(all_words)
    vector2 = [0] * len(all_words)

    # build the vector for the first sentence
    for w in sent1:
        if w in stopwords:
            continue
        vector1[all_words.index(w)] += 1

    # build the vector for the second sentence
    for w in sent2:
        if w in stopwords:
            continue
        vector2[all_words.index(w)] += 1

    return cosine_similarity(vector1, vector2)
```

Figure 6.34: Generation of sentence vectors

Then the cosine similarity is calculated for the pairs of sentence vectors which represent the sentences. This gives a measure to predict how much each sentence in the document is similar to other sentences in the document. It calculates the cosine similarity as '1' if the sentences are identically similar to each other and as '-1' when the sentences are exactly opposite of each other or else any value between -1 and +1 in all the other cases based on the similarities between them. In other terms the sentences are said to be similar if the cosine distance is '0'. Therefore when the cosine distance is less, it implies that the similarity between the sentences is high. I.e. Cosine similarity is equal to (1-cosine distance).

```
def cosine_similarity(a, b):
    dot_product = np.dot(a, b)
    norm_a = np.linalg.norm(a)
    norm_b = np.linalg.norm(b)
    return dot_product / (norm_a * norm_b)
```

Figure 6.35: Cosine similarity calculation

Then a sentence similarity matrix is generated based on the similarities between the sentences and it needs to be converted to a graph to apply the PageRank algorithm to assign sentence scores. When building the sentence similarity matrix the values need to be normalized with TF-IDF to remove noise from the stop words.

```
def build_similarity_matrix(sentences, stopwords=None):
    # Create an empty similarity matrix
    S = np.zeros((len(sentences), len(sentences)))

    for idx1 in range(len(sentences)):
        for idx2 in range(len(sentences)):
            if idx1 == idx2:
                continue

            S[idx1][idx2] = sentence_similarity(sentences[idx1], sentences[idx2], stop_words)

    # normalize the matrix row-wise
    for idx in range(len(S)):
        S[idx] /= S[idx].sum()

    return S
```

Figure 6.36: Generation of sentence similarity matrix

The ultimate result of this approach is returning the PageRank scores for each sentence in the original set of documents. The inbuilt pagerank function from the networkx library was used to compute pageRank scores for each sentence.

```
def textrank(document):
    sentences = sent_tokenize(document)
    S = build_similarity_matrix(sentences, stop_words)
    nx_graph = nx.from_numpy_matrix(S)
    sentence_scores = nx.pagerank(nx_graph)
    return sentence_scores
```

Figure 6.37: Application of the pageRank algorithm for the sentence similarity graph

6.5.2 Implementation of the feature based approach for sentence scoring

The feature based method assigns a weighted average score for each sentence based on the availability of a set of pre-defined features in the sentences. These features include the sentence position, sentence length, availability of title words, named entities count, nouns count, verbs count, numerical literals count, key word frequencies etc.

The sentence position feature assigns each sentence a score based on the position of each sentence in the original document. So it assigns high scores for the sentences which are at the first place of the document or the introductory sentences and the sentences which are at the end of the document or the concluding sentences.

```
def sentencePositionFeature(i, size):
    # size=total number of sentences in the text
    relative_position = i / size

    if 0 < relative_position <= 0.1:
        return 0.17
    elif 0.1 < relative_position <= 0.2:
        return 0.23
    elif 0.2 < relative_position <= 0.3:
        return 0.14
    elif 0.3 < relative_position <= 0.4:
        return 0.08
    elif 0.4 < relative_position <= 0.5:
        return 0.05
    elif 0.5 < relative_position <= 0.6:
        return 0.04
    elif 0.6 < relative_position <= 0.7:
        return 0.06
    elif 0.7 < relative_position <= 0.8:
        return 0.04
    elif 0.8 < relative_position <= 0.9:
        return 0.04
    elif 0.9 < relative_position <= 1.0:
        return 0.15
    else:
        return 0
```

Figure 6.38: Calculation of the sentence position score

The sentence length feature assigns each sentence a score based on the relative length of the sentence compared relatively to an ideal length of a sentence. So the sentences which are too short and sentences which are too long are excluded from the summary.

```
ideal = 20.0

def lengthFeature(sentence):
    len_diff = math.fabs(ideal - len(sentence))
    return len_diff / ideal
```

Figure 6.39: Calculation of the sentence length score

The title word feature checks for the availability of words in the news headline in each sentence and assigns high scores for the sentences which contain words in the headline. It assigns a score between 0 and 1 based on the percentage of words that are common in the sentences with the headline.

```
def getTitleScore(title, sentence):
    titleWords = removeStopWords(title)
    sentenceWords = removeStopWords(sentence)
    matchedWords = [word for word in sentenceWords if word in titleWords]
    if (len(title) != 0):
        titleScore = len(matchedWords) / (len(title) * 1.0)
    else:
        titleScore = 0.0

    return titleScore
```

Figure 6.40: Calculation of the title score

Based on the number of numerical literals, nouns, verbs, named entities in each sentence a score is assigned to them. These feature scores are normalized by sigmoid function in order to get feature scores ranging from 0 to 1 and thereby having an evenly distributed contribution.

```
def nounFeature(sentence):
    tagged = pos_tag(sentence)  # penn tree bank pos tags
    counts = Counter(tag for word, tag in tagged)
    nouns = [word for word, pos in tagged \
             if (pos == 'NN' or pos == 'NNP' or pos == 'NNS' or pos == 'NNPS')]
    if ((len(sentence) - 1) != 0):
        nn = len(nouns) / (len(sentence) - 1)
        noun_score = sigmoid(nn)
    else:
        noun_score = 0.0

    return noun_score
```

Figure 6.41: Calculation of the noun score

```
def numericalFeature(sentence):
    b = []
    for element in sentence:
        b.append(len(element))
    numbers = [int(s) for s in sentence if s.isdigit()]
    if ((len(sentence) - 1) != 0):
        n = len(numbers) / (len(sentence) - 1)
        numeric_score = sigmoid(n)
    else:
        numeric_score = 0.0

    return numeric_score
```

Figure 6.42: Calculation of the numerical literal score

```
def verbFeature(sentence):
    tagged = pos_tag(sentence)
    counts = Counter(tag for word, tag in tagged)
    verbs = [word for word, pos in tagged \
             if (pos == 'VBD' or pos == 'VBG' or pos == 'VBN' or pos == 'VBP')]
    if ((len(sentence) - 1) != 0):
        v = len(verbs) / (len(sentence) - 1)
        verb_score = sigmoid(v)
    else:
        verb_score = 0.0

    return verb_score
```

Figure 6.43: Calculation of the verb score

```
def properNounFeature(sentence):
    tagged = pos_tag(sentence)
    properNouns = [word for word, pos in tagged \
                   if (pos == 'NNP' or pos == 'NNPS')]
    if ((len(sentence) - 1) != 0):
        p = len(properNouns) / (len(sentence) - 1)
        properNoun_score = sigmoid(p)
    else:
        properNoun_score = 0.0
    return properNoun_score
```

Figure 6.44: Calculation of the proper noun score

The thematic word feature assign sentence scores based on the term frequencies which are the word counts of each word in the sentence and thereby identifying the most

frequent words which are referred to as keywords of the text. Before identifying the keywords of the document, the stop words which do not have any semantic meaning need to be removed. Because these stop words are more frequently appearing in the documents and they should not be identified as key words. So prior to the identification of key words these stop words need to be eliminated.

```
def getKeywords(text):
    text = removePunctations(text)
    words = splitWords(text)
    words = removeStopWords(words)
    uniqueWords = list(set(words))

    keywords = [{'word': word, 'count': words.count(word)} for word in uniqueWords]
    keywords = sorted(keywords, key=lambda x: -x['count'])

    return (keywords, len(words))


def getTopKeywords(keywords, wordCount):
    for keyword in keywords:
        articleScore = 1.0 * keyword['count'] / wordCount
        keyword['totalScore'] = articleScore * 1.5

    return keywords
```

Figure 6.45: Calculation of the key word frequencies

Then based on the individual scores obtained for each feature the final aggregated score is calculated for each of the sentences. The weights are assigned for the features in the feature vector based on their relative importance when generating the summary. It has proven that the title score and the key word frequencies have a higher weightage or importance in the generation of summary and those feature are assigned higher weights. Then final sentence scores are calculated as a weighted average score of sum of the products of individual feature scores and their weights. Thereby the final sentence scores of each sentence in the original documents are returned by the function.

```
def score(sentences, titleWords, topKeywords):
    keywordList = [keyword['word'] for keyword in topKeywords]
    senSize = len(sentences)
    ranks = []
    for i, sentence in enumerate(sentences):
        sent = removePunctations(sentence)
        words = splitWords(sent)
        sbsFeature = sbs(words, topKeywords, keywordList)
        dbsFeature = dbs(words, topKeywords, keywordList)
        titleScore = getTitleScore(titleWords, words)
        lengthScore = lengthFeature(words)
        sentencePositionScore = sentencePositionFeature(i, senSize)
        keyWordFrequency = (sbsFeature + dbsFeature) / 2.0 * 10.0
        numericScore = numericalFeature(words)
        nounScore = nounFeature(words)
        verbScore = verbFeature(words)
        properNounScore = properNounFeature(words)
        # weighted average of scores from eight categories
        totalScore = (
                            titleScore * 1.5 + lengthScore * 0.5 + sentencePositionScore * 1.0 +
                            keyWordFrequency * 2.0 + numericScore * 1.0 + nounScore * 1.0 +
                            verbScore * 1.0 + properNounScore * 1.0
                    ) / 8.0
        ranks.append(totalScore)

    return ranks
```

Figure 6.46: Calculation of the weighted average score

Then the final sentence scores are calculated as an average value of scores taken from the above two approaches. Then the system selects the top ranked sentences to form individual summaries for the news articles in the cluster. The number of top ranked sentences depends on a compression rate which is usually selected as 30% of the original text. Finally those individual summaries are compiled together to form an intermediate level of summary.

```
def summarize(orig_text, title):
    summaries = []
    sentences = sent_tokenize(orig_text)

    textRank_scores = textRank_cosineSimilarity.textrank(orig_text).values()
    print("textRank based scores")
    print(textRank_scores)

    feature_Based_Scores = Feature_base.FeatureBasedScores(title, orig_text)
    print("feature based scores")
    print(feature_Based_Scores)

    Final_Sentence_Scores = [sum(n) / 2 for n in
                             zip(*[textRank_scores, feature_Based_Scores])]
    print("Final sentence scores")
    print(Final_Sentence_Scores)

    summary_length = round(len(sentences) * 0.3)
    sorted_sentences = sorted(((Final_Sentence_Scores[i], s) for i, s in enumerate(sentences)),
                              reverse=True)

    for sen in sorted_sentences:
        summaries.append(sen[1])

    result = summaries[:summary_length]
    print("Individual summary")
    print(result)
    return result
```

Figure 6.47: Aggregated individual level of summary generation

6.5.5 Redundancy removal

After the intermediate level of summary is created by combining the individual summaries, there are some post processing tasks need to be applied to make the final summary a more readable and coherent one. The redundancy removal is such a very important post processing task needs to be performed. Since we are taking the top ranked sentences from each e-news article in the cluster and aggregate them to form the final summary, there may be redundant sentences in the final summary. The reason for that is different e-news articles may use different words and phrases to describe about the same thing. Redundancies of the sentences are removed by identifying similar sentences in three perspectives namely the syntactic similarity, lexical similarity and semantic similarity.

6.5.5.1 Lexical similarity

By finding the lexical similarities between sentences we found the total overlaps between vocabularies of the sentences. So the jaccard similarity score between sentences is calculated which is based on the total overlaps between words. The overlaps between words were measured based on the overlaps between word tokens, word stems and word lemmas and final lexical similarity was gained as an average of these three values. If the lexical similarity is greater than a threshold value defined as 0.7, the sentences were considered as lexically similar and thereby remove the redundant sentences.

```
def token_set_match(a, b):
    tokens_a = [token.lower().strip(string.punctuation) for token in nltk.word_tokenize(a) \
                if token.lower().strip(string.punctuation) not in stopwords]
    tokens_b = [token.lower().strip(string.punctuation) for token in nltk.word_tokenize(b) \
                if token.lower().strip(string.punctuation) not in stopwords]

    ratio = len(set(tokens_a).intersection(tokens_b)) / float(len(set(tokens_a).union(tokens_b)))
    return ratio
```

Figure 6.48: Jaccard similarity measurement for word tokens

```
def stem_set_match(a, b):
    tokens_a = [token.lower().strip(string.punctuation) for token in nltk.word_tokenize(a) \
                if token.lower().strip(string.punctuation) not in stopwords]
    tokens_b = [token.lower().strip(string.punctuation) for token in nltk.word_tokenize(b) \
                if token.lower().strip(string.punctuation) not in stopwords]
    stems_a = [stemmer.stem(token) for token in tokens_a]
    stems_b = [stemmer.stem(token) for token in tokens_b]

    ratio = len(set(stems_a).intersection(stems_b)) / float(len(set(stems_a).union(stems_b)))
    return ratio
```

Figure 6.49: Jaccard similarity measurement for word stems

```
def lemma_set_match(a, b):
    pos_a = map(get_wordnet_pos, nltk.pos_tag(nltk.word_tokenize(a)))
    pos_b = map(get_wordnet_pos, nltk.pos_tag(nltk.word_tokenize(b)))
    lemmae_a = [lemmatizer.lemmatize(token.lower().strip(string.punctuation), pos) for token, pos in pos_a \
                if pos == nltk.wordnet.NOUN and token.lower().strip(string.punctuation) not in stopwords]
    lemmae_b = [lemmatizer.lemmatize(token.lower().strip(string.punctuation), pos) for token, pos in pos_b \
                if pos == nltk.wordnet.NOUN and token.lower().strip(string.punctuation) not in stopwords]

    ratio = len(set(lemmae_a).intersection(lemmae_b)) / float(len(set(lemmae_a).union(lemmae_b)))
    return ratio
```

Figure 6.50: Jaccard similarity measurement for word lemma

6.5.5.2 Syntactic similarity

Syntactic similarity between sentences identify sentences which have the same syntactic relationships. To find the syntactic relationships between the sentences, 2-gram models are modelled for the sentence pair and the dice coefficient is computed between them. If the dice coefficient is a non-zero value, it's decided that the two sentences have some syntactic relationship.

```
def dice_coefficient(a, b):
    if not len(a) or not len(b): return 0.0
    if len(a) == 1:  a = a + u'.'
    if len(b) == 1:  b = b + u'.'

    a_bigram_list = []
    for i in range(len(a) - 1):
        a_bigram_list.append(a[i:i + 2])
    b_bigram_list = []
    for i in range(len(b) - 1):
        b_bigram_list.append(b[i:i + 2])

    a_bigrams = set(a_bigram_list)
    b_bigrams = set(b_bigram_list)
    overlap = len(a_bigrams & b_bigrams)
    dice_coeff = overlap * 2.0 / (len(a_bigrams) + len(b_bigrams))

    return dice_coeff
```

Figure 6.51: Calculation of the dice coefficient

But the major concern here is although the sentences are syntactically similar the meaning of the sentences may be different. So the semantic similarity between sentences also needed to be found.

6.5.5.2 Semantic similarity

The algorithm to find the semantic similarities between sentences used two main methods; wordNet based semantic similarity and word2vec based semantic similarity. The implementation details of each of the methods are described below.

6.5.5.2.1 Implementation of the wordNet based semantic similarity

This approach computes the level of semantic similarity between sentences based on the synsets given by the wordNet lexical dictionary for each word in the sentences. It first assigns WordNet part of speech tags; i.e. either noun, verb, adjective or adverb. Then WordNet based synsets are assigned for each tagged word in the sentence pair. Then it compares pair by pair synsets and computes the path distance between the synsets. Finally all the path distances are accumulated together to find the final semantic similarity between sentences based on wordNet.

```
def sentence_similarity(sentence1, sentence2):

    sentence1 = pos_tag(word_tokenize(sentence1))
    sentence2 = pos_tag(word_tokenize(sentence2))

    synsets1 = [tagged_to_synset(*tagged_word) for tagged_word in sentence1]
    synsets2 = [tagged_to_synset(*tagged_word) for tagged_word in sentence2]

    synsets1 = [ss for ss in synsets1 if ss]
    synsets2 = [ss for ss in synsets2 if ss]

    score, count = 0.0, 0
    arr_simi_score = []

    for syn1 in synsets1:
        for syn2 in synsets2:
            simi_score = syn1.path_similarity(syn2)
            if simi_score is not None:
                arr_simi_score.append(simi_score)
                best = max(arr_simi_score)
                if best is not None:
                    score += best
                    count += 1

    score /= count
    return score
```

Figure 6.52: WordNet based semantic similarity measurement

6.5.5.2.1 Implementation of the word2vec based semantic similarity

In this approach the level of semantic similarity between sentences are found by modelling word embeddings by using a pre-trained artificial neural network. For the

implementation purposes, we used Google News's pre-trained neural network trained by using its data set which contains around about 100 billion words. Based on this pre-trained neural network, word embeddings are modelled for each word in the sentence pair. Then considering pairs of word embeddings, the cosine distance between those word embeddings are calculated. Finally all these cosine distances are summed up to find the word2vec based semantic similarity between the sentences.

```
model = gensim.models.KeyedVectors.load_word2vec_format('GoogleNews-vectors-negative300.bin', binary=True, limit=5000)

index2word_set = set(model.wv.index2word)


def avg_feature_vector(sentence, model, num_features, index2word_set):
    words = sentence.split()
    feature_vec = np.zeros((num_features,), dtype='float32')
    n_words = 0
    for word in words:
        if word in index2word_set:
            n_words += 1
            feature_vec = np.add(feature_vec, model[word])
    if (n_words > 0):
        feature_vec = np.divide(feature_vec, n_words)
    return feature_vec


def word2vecSim(s1, s2):
    s1_afv = avg_feature_vector(s1, model=model, num_features=300, index2word_set=index2word_set)

    s2_afv = avg_feature_vector(s2, model=model, num_features=300, index2word_set=index2word_set)

    sim = 1 - spatial.distance.cosine(s1_afv, s2_afv)

    return sim
```

Figure 6.53: Word2vec based semantic similarity measurement

Finally the overall semantic similarity is measured by taking the average value of two similarity scores taken from the wordNet based method and word2vec based method. If that final semantic similarity score is greater than the threshold value defined as 0.7, it's proven that those sentences are semantically similar and thereby remove those redundant sentences.

```
def semantic_similarity(sentence1, sentence2):
    similarity_wordNet = (sentence_similarity(sentence1, sentence2) + sentence_similarity(sentence2, sentence1)) / 2
    similarity_word2vec = word2vecSemanticSimilarity.word2vecSim(sentence1, sentence2)
    semantic_simlarity_score = (similarity_wordNet + similarity_word2vec) / 2
    return semantic_simlarity_score


def removeReduandancy(sentences):
    for sentence in sentences:

        target_sentence = sentence
        index = sentences.index(sentence)
        # print(target_sentence)
        for sentence in sentences[(index + 1):]:

            similarity_score = semantic_similarity(target_sentence, sentence)

            if (similarity_score > 0.7):
                sentences.remove(sentence)
    return sentences
```

Figure 6.54: Semantic redundancy elimination

6.5.6 Sentence ordering

After removing the redundant sentences from the intermediate level of summary, it's very important to arrange the final summary sentences in the proper coherent order. Otherwise it will reduce the readability. So, the sentence ordering was performed by sequence matching. The sequence ratio is computed which is kind of a coherence score in both the directions for a given sentence pair and the sentences are arranged in the order which preserves the highest sequence ratio between the sentences. The SequenceMatcher from the difflib library was used for sequence matching. Then the final summary is displayed to the user after arranging the summary sentences in the proper coherent order.

```
from difflib import SequenceMatcher

def seq_Matching(sentences):
    n = len(sentences)

    for i in range(n):

        for j in range(0, n - i - 1):

            s1 = SequenceMatcher(lambda x: x == " ",
                                 sentences[j],
                                 sentences[j + 1])
            s2 = SequenceMatcher(lambda x: x == " ",
                                 sentences[j + 1],
                                 sentences[j])
            seq1 = s1.ratio()
            seq2 = s2.ratio()
            if (seq2 > seq1):
                sentences[j], sentences[j + 1] = sentences[j + 1], sentences[j]

    return sentences
```

Figure 6.55: Sentence ordering using sequence matching

6.6 Implementation of the recommendation module

The implementation details of the hybrid recommendation module are described below. The individual implementation details of each individual recommendation method i.e. the implementation details of content based filtering, collaborative filtering, popularity model are also discussed.

6.6.1 Popularity model

A common (and usually hard-to-beat) baseline approach is the Popularity model. This model is not actually personalized – it simply recommends a user the most popular items that the user has not previously consumed. As the popularity accounts for the "wisdom of the crowds", it usually provides good recommendations, generally interesting for most of the people. The main objective of a recommender system is to leverage the long-tail items to the users with very specific interests, which goes far beyond this simple.

```
#Computes the most popular items
item_popularity_df = interactions_full_df.groupby('contentId')['eventStrength'].sum().sort_values(ascending=False
item_popularity_df.head(10)
```

Figure 6.56: Identification of most popular items

```
class PopularityRecommender:
    MODEL_NAME = 'Popularity'

    def __init__(self, popularity_df, items_df=None):
        self.popularity_df = popularity_df
        self.items_df = items_df

    def get_model_name(self):
        return self.MODEL_NAME

    def recommend_items(self, user_id, items_to_ignore=[], topn=10, verbose=False):
        # Recommend the more popular items that the user hasn't seen yet.
        recommendations_df = self.popularity_df[~self.popularity_df['contentId'].isin(items_to_ignore)] \
                             .sort_values('eventStrength', ascending = False) \
                             .head(topn)

        if verbose:
            if self.items_df is None:
                raise Exception('"items_df" is required in verbose mode')

            recommendations_df = recommendations_df.merge(self.items_df, how = 'left',
                                                          left_on = 'contentId',
                                                          right_on = 'contentId')[['eventStrength', 'contentId

        return recommendations_df
```

Figure 6.57: Popularity model based recommendation

6.6.2 Content-based Filtering Model

Content-based filtering approaches leverage description or attributes from items the user has interacted to recommend similar items. It depends only on the user's previous

choices, making this method robust to avoid the cold-start problem. It is simple to use the raw text to build item profiles and user profiles.

Here we are using a very popular technique in information retrieval (search engines) named TF-IDF. This technique converts unstructured text into a vector structure, where each word is represented by a position in the vector, and the value measures how relevant a given word is for an article. As all items will be represented in the same Vector Space Model it is easy to compute similarity between articles.

```
9   #Ignoring stopwords (words with no semantics) from English and Portuguese (as we have a corpus with mixed langu
10  stopwords_list = stopwords.words('english') + stopwords.words('portuguese')
11
12  #Trains a model whose vectors size is 5000, composed by the main unigrams and bigrams found in the corpus, igno
13  vectorizer = TfidfVectorizer(analyzer='word',
14                               ngram_range=(1, 2),
15                               min_df=0.003,
16                               max_df=0.5,
17                               max_features=5000,
18                               stop_words=stopwords_list)
19
20  item_ids = articles_df['contentId'].tolist()
21  tfidf_matrix = vectorizer.fit_transform(articles_df['title'] + "" + articles_df['text'])
22  tfidf_feature_names = vectorizer.get_feature_names()
23  tfidf_matrix
24
```

Figure 6.58: Modelling the vector space model

To model the user profile, we take all the news profiles the user has interacted and average them. The average is weighted by the interaction strength, in order words, the articles the user has interacted the most (e.g. Liked or commented) will have a higher strength in the final user profile.

```
def get_item_profile(item_id):
    idx = item_ids.index(item_id)
    item_profile = tfidf_matrix[idx:idx+1]
    return item_profile

def get_item_profiles(ids):
    item_profiles_list = [get_item_profile(x) for x in ids]
    item_profiles = scipy.sparse.vstack(item_profiles_list)
    return item_profiles

def build_users_profile(person_id, interactions_indexed_df):
    interactions_person_df = interactions_indexed_df.loc[person_id]
    user_item_profiles = get_item_profiles(interactions_person_df['contentId'])

    user_item_strengths = np.array(interactions_person_df['eventStrength']).reshape(-1,1)
    #Weighted average of item profiles by the interactions strength
    user_item_strengths_weighted_avg = np.sum(user_item_profiles.multiply(user_item_strengths), axis=0) / np.s
    user_profile_norm = sklearn.preprocessing.normalize(user_item_strengths_weighted_avg)
    return user_profile_norm

def build_users_profiles():
    interactions_indexed_df = interactions_full_df[interactions_full_df['contentId'] \
                                                   .isin(articles_df['contentId'])].set_index('personId')
    user_profiles = {}
    for person_id in interactions_indexed_df.index.unique():
        user_profiles[person_id] = build_users_profile(person_id, interactions_indexed_df)
    return user_profiles

```

Figure 6.59: Building user profiles

```
class ContentBasedRecommender:

    MODEL_NAME = 'Content-Based'

    def __init__(self, items_df=None):
        self.item_ids = item_ids
        self.items_df = items_df

    def get_model_name(self):
        return self.MODEL_NAME

    def _get_similar_items_to_user_profile(self, person_id, topn=1000):
        #Computes the cosine similarity between the user profile and all item profiles
        cosine_similarities = cosine_similarity(user_profiles[person_id], tfidf_matrix)
        #Gets the top similar items
        similar_indices = cosine_similarities.argsort().flatten()[-topn:]
        #Sort the similar items by similarity
        similar_items = sorted([(item_ids[i], cosine_similarities[0,i]) for i in similar_indices], key=lambda
        return similar_items
```

Figure 6.60: Content based recommendation

```
    def recommend_items(self, user_id, items_to_ignore=[], topn=10, verbose=False):
        similar_items = self._get_similar_items_to_user_profile(user_id)
        #Ignores items the user has already interacted
        similar_items_filtered = list(filter(lambda x: x[0] not in items_to_ignore, similar_items))

        recommendations_df = pd.DataFrame(similar_items_filtered, columns=['contentId', 'recStrength']) \
                                .head(topn)

        if verbose:
            if self.items_df is None:
                raise Exception('"items_df" is required in verbose mode')

            recommendations_df = recommendations_df.merge(self.items_df, how = 'left',
                                                          left_on = 'contentId',
                                                          right_on = 'contentId')[['recStrength', 'contentId',


        return recommendations_df
```

Figure 6.61: Recommendation of items based on content based filtering

6.6.3 Collaborative Filtering Model

Collaborative Filtering (CF) has two main implementation strategies:

- **Memory-based:** This approach uses the memory of previous users interactions to compute users similarities based on items they've interacted (user-based approach) or compute items similarities based on the users that have interacted with them (item-based approach). A typical example of this approach is User Neighborhood-based CF, in which the top-N similar users are selected and used to recommend items those similar users liked, but the current user have not interacted yet.
- **Model-based:** In this approach, models are developed using different machine learning algorithms to recommend items to users. There are many model-based CF algorithms, like neural networks, Bayesian networks, clustering models, and latent factor models such as Singular Value Decomposition (SVD) and, probabilistic latent semantic analysis.

```
class CFRecommender:
    MODEL_NAME = 'Collaborative Filtering'

    def __init__(self, cf_predictions_df, items_df=None):
        self.cf_predictions_df = cf_predictions_df
        self.items_df = items_df

    def get_model_name(self):
        return self.MODEL_NAME

    def recommend_items(self, user_id, items_to_ignore=[], topn=10, verbose=False):
        # Get and sort the user's predictions
        sorted_user_predictions = self.cf_predictions_df[user_id].sort_values(ascending=False) \
                                    .reset_index().rename(columns={user_id: 'recStrength'})

        # Recommend the highest predicted rating movies that the user hasn't seen yet.
        recommendations_df = sorted_user_predictions[~sorted_user_predictions['contentId'].isin(items_to_ignor
                             .sort_values('recStrength', ascending = False) \
                             .head(topn)
        if verbose:
            if self.items_df is None:
                raise Exception('"items_df" is required in verbose mode')

            recommendations_df = recommendations_df.merge(self.items_df, how = 'left',
                                                          left_on = 'contentId',
                                                          right_on = 'contentId')[['recStrength', 'contentId',

        return recommendations_df
```

Figure 6.62: Collaborative filtering recommender

6.6.4 Hybrid Recommender Model

In our approach, we combine Collaborative Filtering and Content-based Filtering algorithms. Then it'll provide more accurate recommendations. In fact, hybrid methods have performed better than individual approaches in many studies and have being extensively used by researches and practice works. We used in our hybridization method, by only multiply the CF score with the Content-based score and ranking by resulting score.

```
class HybridRecommender():
    print("*******Hybrid Recommendation for the User********")
    MODEL_NAME = 'Hybrid'

    def __init__(self, cb_rec_model, cf_rec_model, items_df):
        self.cb_rec_model = cb_rec_model
        self.cf_rec_model = cf_rec_model
        self.items_df = items_df

    def get_model_name(self):
        return self.MODEL_NAME
```

Figure 6.63: Hybrid recommender model

```
def recommend_items(self, user_id, items_to_ignore=[], topn=10, verbose=False):
    #Getting the top-1000 Content-based filtering recommendations
    cb_recs_df = self.cb_rec_model.recommend_items(user_id, items_to_ignore=items_to_ignore, verbose=verbo
                                                   topn=1000).rename(columns={'recStrength': 'recStren
    #Getting the top-1000 Collaborative filtering recommendations
    cf_recs_df = self.cf_rec_model.recommend_items(user_id, items_to_ignore=items_to_ignore, verbose=verbo
                                                   topn=1000).rename(columns={'recStrength': 'recStren
    #Combining the results by contentId
    recs_df = cb_recs_df.merge(cf_recs_df,
                               how = 'inner',
                               left_on = 'contentId',
                               right_on = 'contentId')

    #Computing a hybrid recommendation score based on CF and CB scores
    recs_df['recStrengthHybrid'] = recs_df['recStrengthCB'] * recs_df['recStrengthCF']

    #Sorting recommendations by hybrid score
    recommendations_df = recs_df.sort_values('recStrengthHybrid', ascending=False).head(topn)

    if verbose:
        if self.items_df is None:
            raise Exception('"items_df" is required in verbose mode')

        recommendations_df = recommendations_df.merge(self.items_df, how = 'left',
                                                      left_on = 'contentId',
                                                      right_on = 'contentId')[['recStrengthHybrid', 'conte

    return recommendations_df
```

Figure 6.64: Recommendations of the hybrid model

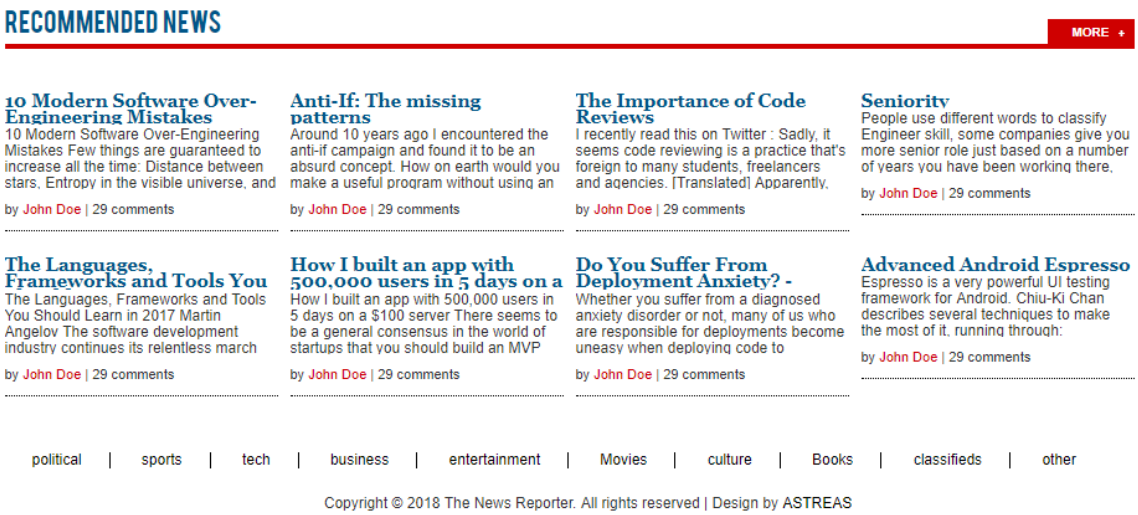

Figure 6.65: UI interface of the recommended news

6.7 Summary

This chapter provides the overview of the implementation of the project. We have stated how the project was built step by step and approaches we followed to accomplish them. The module wise implementations are further described in this chapter.

Chapter 7

7. Evaluation

7.1 Introduction

This chapter focusses on the results obtained through implementation of the algorithms we have proposed. Results obtained through experimentation is summarized and analyzed through discussions. These discussions are used for deriving conclusion on the algorithms and approaches we have used for achieving the aim and objectives.

7.2 Evaluation of classification module

To evaluate classification module Confusion matrix, Precision, Recall, and F_1 measure are used. Those evaluation matrices evaluate the accuracy of a classification.

7.2.1 Data Set

Used BBC news dataset. It consists 2225 e-news articles from the BBC e-news website corresponding to e-news in five main areas from 2004 to 2005. Class labels are political, business, Entertainment, technology and sports. There are 510 business news, 386 entertainment news, 414 political news, 511 sports news and 401 technological news in the dataset.

7.2.2 Evaluation matrices

A confusion matrix CM is such that $CM_{p,q}$ is equivalent to the number of observations recognized as to be in group *p* but predicted to be in group *q*. Confusion matrix for each individual classifiers are created and then the ensemble classifier confusion matrix is created. A true positive values are increased in ensemble classifier.

Classified as-> Class Label	Business	Entertainment	Political	Sport	Tech
Business	110	1	3	0	1
Entertainment	0	71	1	0	0
Political	2	0	73	0	1
Sport	1	0	0	101	0
Tech	1	1	0	0	78

Table 7.1: Confusion Matrix of Support Vector Machine

In testing data 115 news items are originally labeled as business news, 110 of which were classified as business, 1 as entertainment, 3 as political and 1 as tech. 72 news items are originally labeled as entertainment, 71 of which classified as entertainment and 1 as political news.76 news items are originally labeled as political news, 73 of which were classified as political, 2 as business and 1 as tech. 102 items are originally labeled as sports, 101 of which were classified as sport news and 1 as business news. 80 news items are originally classified as tech. 78 of which were classified as tech, 1 as business and 1 as entertainment.

Classified as-> Class Label	Business	Entertainment	Political	Sport	Tech
Business	113	0	2	0	0
Entertainment	2	67	2	1	0
Political	3	0	72	1	0
Sport	1	0	0	101	0
Tech	2	1	0	1	76

Table 7.2: Confusion Matrix of Random Forest Algorithm

Classified as-> Class Label	Business	Entertainment	Political	Sport	Tech
Business	110	1	3	0	1
Entertainment	0	66	4	0	1
Political	1	0	74	0	1
Sport	1	0	0	101	0
Tech	0	1	0	0	79

Table 7.3: Confusion Matrix of Multinomial Naïve Bayes

Classified as-> Class Label	Business	Entertainment	Political	Sport	Tech
Business	111	0	3	0	1
Entertainment	0	71	1	0	0
Political	1	0	74	0	1
Sport	1	0	0	101	0
Tech	0	1	0	0	79

Table 7.4: Confusion Matrix of the Ensemble Classifier

To select the best kernel function for this domain, calculated the accuracy of the system with different kernel functions. Blow table shows the results of different kernel function. Kernel functions are applied for non-linearly separable domains to map into higher dimension spaces which can easily separable.

SVM Classifiers with different Kernel functions	Accuracy
SVC with linear kernel	0.973033
LinearSVC	0.973033

SVC with RBF kernel	0.968539
SGDC	0.964044

Table 7.5: SVM with different Kernel Functions

Calculated the accuracy over different classifiers to select the best three classifiers to create ensemble classifier. Below table shows the results.

Classifier	**Accuracy**
RandomForestClassifier	0.96404
MultinomialNB	0.95751
GussianNB	0.92132
BernoulliNB	0.94831
Support Vector Machine	0.97303
Ensemble classifier(Hard Voting) **(RandomForestClassifier, MultinomialNB, Support Vector Machine)**	0.97528
Ensemble classifier(Soft Voting) **(RandomForestClassifier, MultinomialNB, Support Vector Machine)**	0.97977

Table 7.6: Accuracy of classifiers

RandomForestClassifier, MultinomialNB, and Support Vector Machine gives higher accuracy. So we used those three classifiers to create ensemble classifier. Majority rule voting is used in hard Voting Classifier. The soft voting method predicts the e-news class label based on the sum of the predicted probabilities of individual classifies. The soft voting method gives more accurate result than hard voting method. So we used the soft voting method to implement our system.

To evaluate individual classifiers and ensemble classifier precision, recalled and F1 score are used.

$$Precision = \frac{Ture\ Positive}{True\ Positive + False\ Postitive}$$

$$Recall = \frac{Ture\ Positive}{True\ Positive + False\ Negative}$$

$$F1\ Score = 2\frac{Precision * Recall}{Precision + Recall}$$

		SVM	RF	MNB	Ensemble
	Precision	0.9565	0.9826	0.9565	0.9652
Business	**Recall**	0.9565	0.9338	0.9821	0.9823
	F_1 Score	0.9565	0.9575	0.9691	0.9736
	Precision	0.9861	0.9178	0.9295	0.9861
Entertainment	**Recall**	0.9861	0.9852	0.9705	0.9861
	F_1 Score	0.9861	0.9503	0.9495	0.9861
	Precision	0.9605	0.9473	0.9736	0.9736
Political	**Recall**	0.9605	0.9473	0.9135	0.9487
	F_1 Score	0.9605	0.9473	0.9425	0.9609
	Precision	0.9901	0.9901	0.9901	0.9901
Sport	**Recall**	0.9901	0.9711	1.0000	1.0000
	F_1 Score	0.9901	0.9805	0.9950	0.9950
	Precision	0.9750	0.9500	0.9500	0.9875
Tech	**Recall**	0.9750	1.0000	0.9634	0.9753
	F_1 Score	0.9750	0.9743	0.9566	0.9813

Table 7.7: Precision, Recall, F1 values for each classifiers

The final results show higher average recall, precision and f_1 results for ensemble classier over other three individual classifiers.

7.3 Evaluation of the aggregation module

Evaluation of the quality of clusters is most important in cluster analysis. The evaluation results shows how good the clusters, produced by the clustering algorithm. The evaluation of the clusters has based on F-measure in this study. F-measure is an external quality measure technique which is required external informations about the data and it is used to measure the quality of the clusters for testing data. F-measure is calculated by considering the precision (P) and recall(R) of test dataset. Precision is calculated as the number of correct positive results divided by the number of all positive results of the clusters found by the algorithm. The number of correct positive results divided by the number of all the samples which are identified as positive samples by using an external knowledge is taken as recall. F-measure is the harmonic average of the recall and precision. F-measure gives value 1 for perfect recall and precision as the best value and value 0 as worst value. If a cluster is having high quality F-measure gives a higher value.

$$F - measure = \frac{2.P.R}{P + R}$$

Where,

$$R = \frac{True\ positive}{True\ positive + False\ negative}$$

$$P = \frac{True\ positive}{True\ positive + False\ positive}$$

Where,

True Positive: Similar documents which are in same cluster (Correctly Identified)

False Positive: Dissimilar documents which are in same cluster (Incorrectly Identified)

True Negative: Dissimilar documents which are in different clusters (Correctly Rejected)

False Negative: Similar documents which are in different clusters (Incorrectly Rejected)

7.3.1 Data set

Testing dataset of news aggregation module contain 259 manually collected news articles which belong to 58 topics. The data set contain 10 outliers.

7.3.2 Evaluation results for different feature models

Feature Model	Clustering Algorithm	Precision	Recall	F-Measure
LDA (No. of Topics = 58)	DBSCAN	0.397	0.880	0.548
Doc2vec (window size = 10) (No. of hidden nodes = 300)	DBSCAN	0.222	0.595	0.324
TF-IDF vector space model	DBSCAN	0.969	0.993	0.981

Table 7.8: Evaluation of different feature models for clustering

TF-Idf feature model has recorded higher quality of clustering than LDA model and Doc2vec model.

7.3.3 Evaluation results for different clustering algorithms

Clustering Algorithm	Precision	Recall	F-Measure	Outlier Identification
K-Means (k=58)	0.981	0.976	0.979	2 of 10 (correctly - 2) (wrongly - 0)
Affinity	0.980	0.995	0.987	0 of 10

Propagation				(correctly - 0) (wrongly - 0)
DBSCAN	0.969	0.993	0.981	10 of 10 (correctly - 10) (wrongly - 3)

Table 7.9: Evaluation results of different clustering algorithms

DBSCAN clustering algorithm has recorded higher quality of clustering than other two clustering algorithm.

7.4 Evaluation of the summarization module

Some kind of a matrix is needed to evaluate the summaries generated by the system. So that we can get an idea about the performances, accuracy rates of the summarizer module and thereby we can further research on how to improve the performances.

7.4.1 Evaluation matrix

For evaluating the system generated summaries, the ROUGE automatic summary evaluation metric was used. ROUGE is a recall based metric for fixed-length summaries which is based on n-gram co-occurrence. It reports separate scores for 1, 2, 3, and 4-gram, and also for longest common subsequence co-occurrences. Among these different scores, unigram-based ROUGE score (ROUGE-1) has been shown to agree with human judgments the most. Therefore the ROUGE-1 (unigram-based) metrics was used to evaluate the experiment results. Here a human produced model summary was used to evaluate the system generated summaries for any given e-news cluster. Then the ROUGE scores were computed for the system generated summaries of each cluster against the human summary. ROUGE-1 metric refers to the overlap of unigrams between the system summary and reference summary by computing the recall and precision values. The recall, precision and f-measure values can be calculated by using the formula;

$$\text{Recall} = \frac{\text{no. of overlapping words}}{\text{total no. of words in reference summary}}$$

$$\text{Precision} = \frac{\text{no. of overlapping words}}{\text{total no. of words in system summary}}$$

$$f - measure = \frac{2(Precision * Recall)}{(Precision + Recall)}$$

7.4.2 Evaluation of the individual sentence scoring approaches

In the initial testing phase the system summaries were tested against the human summaries for the different approaches separately. So the evaluation was performed for the graph based approach and the feature based approach separately. The table 7.10 shows the average recall, precision and f-measure values taken for these individual approaches separately.

Method	Recall	Precision	f-measure
Graph based approach – TextRank	66%	72%	69%
Feature based approach	70%	76%	73%

Table 7.10: ROUGE-1 evaluation results for individual sentence scoring approaches

7.4.3 Evaluation of the similarity measures

A hybrid model was designed for sentence scoring which uses both the graph based and feature based methods. In the TextRank algorithm used as the graph based method, different similarity measures were needed to be evaluated in order to find out the similarity measure which gives the best results. Therefore for the evaluation purposes a sample set of individual documents were selected and then evaluated against each similarity measure using recall and precision values. Table 7.11 shows the results gained for each similarity measure separately and the average recall and precision values gained for each measure.

	Euclidean	Bow	Jaccard	Cosine

Document Id	Recall	Precision	Recall	Precision	Recall	Precision	Recall	Precision
Doc 1	0.199	0.174	0.254	0.150	0.291	0.200	0.810	0.542
Doc 2	0.467	0.382	0.382	0.251	0.381	0.260	0.441	0.422
Doc 3	0.321	0.321	0.500	0.429	0.692	0.857	0.667	0.571
Doc 4	0.110	0.067	0.256	0.137	0.256	0.188	0.219	0.157
Doc 5	0.653	0.500	0.714	0.588	0.663	0.528	0.714	0.593
Doc 6	0.797	0.938	1.000	1.000	1.000	1.000	1.000	1.000
Doc 7	0.752	0.675	0.829	0.696	0.828	0.696	1.000	1.000
Doc 8	0.286	0.213	0.286	0.220	0.626	0.508	0.472	0.500
Doc 9	0.261	0.195	0.207	0.218	0.413	0.362	0.413	0.392
Doc 10	0.314	0.226	0.506	0.405	0.517	0.331	0.202	0.180
Average	0.416	0.370	0.493	0.409	0.567	0.493	0.594	0.536

Table 7.11: Evaluation of similarity measures for the TextRank algorithm

So the results showed higher recall and precision values for the summaries which have higher overlaps between system summaries and reference summaries. Based on the average recall and precision values obtained the cosine similarity measure was used which gained the highest accuracy when implementing the textRank algorithm.

7.4.4 Evaluation of the normalization schemes

The feature based approach was evaluated when the features are not normalized, when the feature values are normalized by sentence length and when the feature values are normalized by the sigmoid function. Table 7.12 shows the results gained for each normalization scheme for the sample set of documents in the feature based approach. So based on the results obtained it was achieved the conclusion that the normalization

by the sigmoid function gives the highest accuracy rate when compared with no normalization and normalization by sentence length and hence normalization by sigmoid function was used for implementation.

Document Id	No normalization		Normalization by sentence length		Normalization by sigmoid function	
	Recall	Precision	Recall	Precision	Recall	Precision
Doc1	0.93	0.61	0.81	0.52	0.98	0.61
Doc 2	0.441	0.419	0.533	0.558	0.508	0.444
Doc 3	0.679	0.576	0.679	0.624	0.513	0.555
Doc 4	0.219	0.159	0.402	0.305	0.768	0.563
Doc 5	0.724	0.596	0.948	0.744	0.643	0.525
Doc 6	1.000	1.000	1.000	1.000	1.000	1.000
Doc 7	1.000	1.000	1.000	1.000	1.000	1.000
Doc 8	0.505	0.414	0.505	0.442	0.747	0.739
Doc 9	0.424	0.390	0.695	0.587	0.543	0.617
Doc 10	0.404	0.360	0.393	0.388	0.876	0.672
Average	0.633	0.553	0.697	0.617	0.759	0.673

Table 7.12: Evaluation of normalization schemes in the feature based approach

7.4.5 Evaluation of the system generated summaries

The system was built by combing the feature based method and the graph based method and finally the system generated summaries were evaluated using the ROUGE-1 matrix. The results gained for the final summaries generated by the system by considering sample set of e-news clusters are shown in table 7.13.

Cluster Id	Recall	Precision
Cluster 1	0.708	0.675
Cluster 2	0.846	0.720
Cluster 3	0.949	0.925
Cluster 4	0.805	0.617
Cluster 5	0.816	0.762
Cluster 6	0.819	0.761
Cluster 7	0.880	1.000
Cluster 8	0.667	0.600
Cluster 9	0.772	0.739
Cluster 10	0.865	0.719
Average	0.813	0.752

Table 7.13: Evaluation of the final system generated summaries

Average f-measure = 0.781

The final results show high average recall, precision and f-measure values for the system generated summaries which use a hybrid approach than the summaries generated individually using one single approach. The evaluation results proved an acceptable accuracy rate for the summaries generated by the system.

7.5 Evaluation of the recommendation module

Evaluation is important for machine learning projects because it allows comparing objectively different algorithms and hyper parameter choices for models. One key aspect of evaluation is to ensure that the trained model generalizes for data it was not trained on, using Cross-validation techniques. Here we are using a simple cross-validation approach named holdout, in which a random data sample (20% in this case) are kept aside in the training process, and exclusively used for the evaluation. All evaluation metrics reported here are computed using the test set.

7.5.1 Data set

The CI&T DeskDrop dataset was used to evaluate the proposed methodology. The data set can be found at the kaggle site (https://www.kaggle.com/gspmoreira/recommender-systems-in-python-101/data). The dataset comprises about 73k logged user's interactions on more than 3k public articles. It contains features like item attributes; contextual information like date and time of user visits and geo location; logged users; rich implicit feedback in terms of comments, likes and views etc.

7.5.2 Evaluation matrix

In recommender systems, there is a set metrics commonly used for evaluation purposes. We choose to work with Top-N accuracy metrics, which evaluates the accuracy of the top recommendations provided to a user, comparing to the items the user has interacted in the test set. This evaluation method works as follows:

- For each user
    - For each item the user has interacted in test set
        - Sample 100 other items the user has never interacted. Here we naively assume those non-interacted items are not relevant to the user, which might not be true, as the user may simply not be aware of those not interacted items. But it was considered as an assumption.
        - Ask the recommender model to produce a ranked list recommended items, from a set composed one interacted item and the 100 non-interacted (non-relevant) items.

- Compute the Top-N accuracy metrics for this user and interacted item from the recommendations ranked list.

- Aggregate the global Top-N accuracy metrics

The Top-N accuracy, metric chosen was Recall@N which evaluates whether the interacted item is among the top N items (hit) in the ranked list of 101 recommendations for a user. The evaluation results are given in the following table comparing each of the methods.

Model Name	**recall@10**	**recall@5**
Popularity	0.372923	0.241754
Collaboratve Filtering	0.468167	0.334058
Content-Based	0.524163	0.414600
Hybrid	0.537970	0.433777

Table 7.14: Evaluation results of different recommendation methods

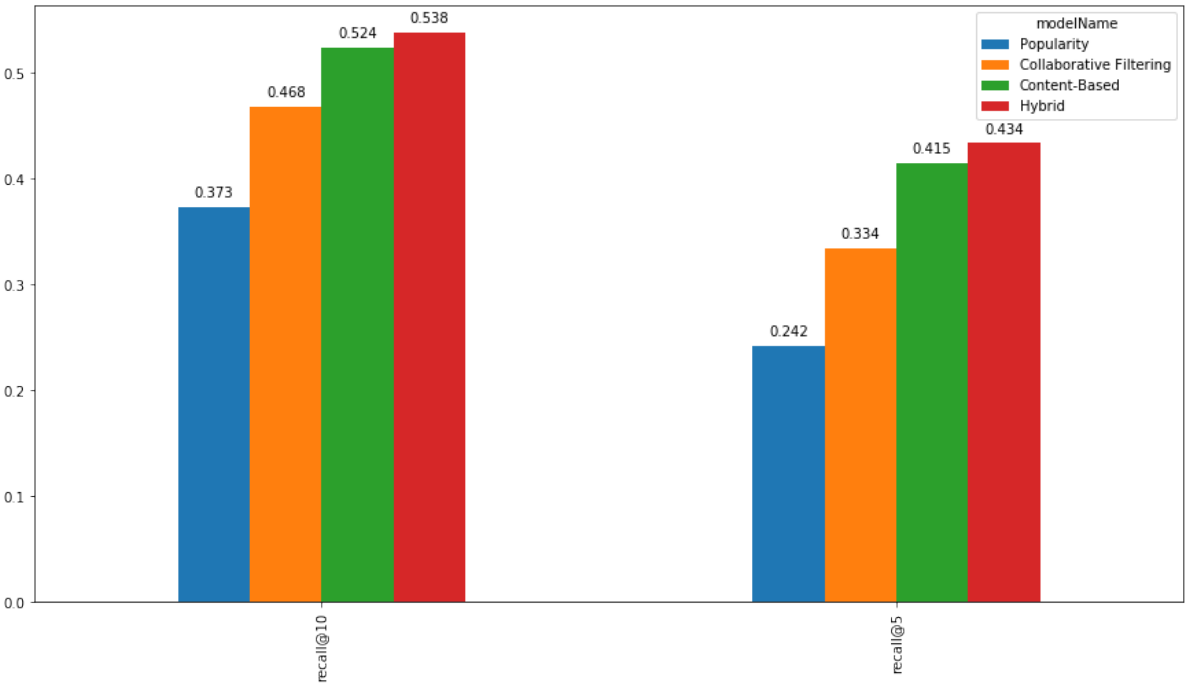

Figure 7.1: Performance of different recommendation methods

7.7 Summary

Results and discussions discussed in this chapter summarizes to drive the project towards deriving conclusions. Results have been discussed in line with the objectives to be achieved in each module. Accuracy and performance of the algorithms and approaches have been discussed with respect to the objectives of each module.

Chapter 8

8. Conclusion & Further work

8.1 Introduction

This chapter describes in detail about the various approaches we have identified to solve the problems in individual modules. Furthermore this chapter compares between the approaches and discusses why we have selected a particular approach to solve the problem.

8.2 Achievement of Objectives

As said in the list of objectives, we have built a personalized e-news recommendation system. The system mainly consists of classification, aggregation, summarization and recommendation components as described in the earlier chapters. The objective of building a classifier was achieved by developing an ensemble classifier. Then the objective of building an e-news aggregation component was achieved by using the DBSCAN algorithm which proved high performance when compared with other aggregation algorithms. As the existing systems like Google and Yahoo do not provide summarized views of news content, that feature was also included in our system. So that the objective of building an intelligent summarizer was also achieved which uses several tools, techniques and algorithms for each design phase of the summarizer component. Since the prevailing recommendation systems provide only the content based recommendations which recommends news items that belong to the same category and do not provide personalized news recommendations, a personalized e-news recommendation system was developed. Therefore, the objective of building a recommender module was achieved by developing a hybrid recommender model using content based filtering, collaborative filtering and popularity model. The last objective was to evaluate the system in a practical environment with real data. So we evaluated the system for the real extracted e-news articles from web sites and evaluated how the system classifies, aggregates, summarizes and recommends e-news articles to the users.

8.3 Problems encountered

With the vast amount of e-news content extracted, handling such an enormous amount of data was very difficult. So we had to define what are the e-news sites we are going to extract the e-news content. We also needed to define how many e-news articles we are going to extract per e-news site beforehand. Since there were dependencies between

the modules, we faced many problems in integrating these modules. Since data are passed between the modules, it was problematic to retrieve those data when the output formats of those data are different like one module outputs a JSON file , another module outputs a text file or a csv file. So we had to follow a single output format for each module.

8.4 E-news extraction and classification module

E-news extraction module uses both RSS feeds and URLs. At the beginning of the process user provides root URLs or RSS feeds into the system. RSS feeds are the first priority because it gives more consistent data than URLs crawling. In the extraction process, only extract e-news articles by excluding all other irrelevant contents like advertisements, user comments, etc. That is one of the difficult tasks in the extraction module. Newspaper library was used with enhanced features for extraction. Checking published data were added as an enhanced feature. Checking publish data gives more consistent data by removing unnecessary data. Feedpaser is used to read RSS feeds. Then by downloading and parsing e-news article content is extracted.

Using RSS feeds or by URL crawling new e-news articles URLs will be gathered. These new URLs are stored in frontier until they scarped. After extraction e-news articles then these articles are send for the pre-processing stage. Pre-processed data will be given better results in classification module. Then ensemble classifier is designed and developed for classification of e-news articles. The most difficult task is to select the best individual classifiers and select the better ensemble method which gives more accurate results in this domain. For that individual classifiers were developed and did an evaluation on each classifier to get accuracy. MultinomialNB, SVM and Random forest supervised learning algorithm were selected. These algorithms show higher accuracy than other individual classifiers. Designed and developed method to eliminate other news category. Decided threshold value by evaluating different e-news articles. To ensemble three classifiers used average values of each individual classifiers. It gives more accurate predictions than majority voting method.

8.5 E-news aggregation module

Throughout the research carried out on e-news aggregation these are the conclusions we arrived at. If the number of news article is not a large number Tf-idf feature model is better than other feature models for feature extraction. But when we have very large number of news articles in the data set, extracting tf-idf features is consume lot of computational power because of high dimensionality. If the number of e-news topics is known and the number of news articles is very large LDA model is good because it has relatively very low dimensionality than tf-idf feature model. When we have very lengthy news articles doc2vec feature model is good.

If the number of clusters is not known before performing the clustering algorithm and if the data set contain outliers DBSCAN algorithm is better than k-means and affinity propagation algorithms. But if the number of clusters is not known before performing the clustering and if the data set does not contain outliers affinity propagation clustering algorithm is better than other two clustering algorithms. If the number of clusters is a known value before performing the clustering algorithm and if the number of news articles is very large k-means clustering algorithm is good because it is the fastest clustering algorithm among the mentioned clustering algorithms.

8.6 E-news summarization module

The most challenging task in generating the summary for e-news clusters is to identify the most significant sentences from the original set of documents. In order to identify the most important sentences we needed a mechanism for assigning each sentence an importance score. So we used a hybrid model for sentence scoring using a graph based method and a feature based method combined together. TextRank algorithm was used as the graph based method which generates sentence similarity graphs considering similarity between sentences measured by using the cosine similarity. Thereby we scored the sentences in the graph using the PageRank algorithm. Also we extracted a set of features from sentences like sentence position, sentence length, title words etc. in the feature based method and then assigned a weighted average score for each sentence in the original set of documents based on the presence of these features. Throughout this project we researched for these different methods for sentence scoring and compared with each other for their accuracies. Finally we came up with a hybrid approach for sentence scoring combining both these methods together since it gave us

better results rather than using them individually. Then we identified that we need to remove the redundant sentences from the final summary which was performed by considering three perspectives of sentence similarities namely the syntactic similarity, lexical similarity and semantic similarity. Lexical similarities were found by using jaccard similarity and the syntactic similarities were measured by modelling 2-gram models and then computing the dice coefficient. The semantic similarities were found by using WordNet dictionary and word2vec model. Then the final summary sentences are arranged in the proper coherent order by sequence matching and form the final summary. After taking the recall, precision and f-measure values for the system generated final summaries we got better results which led us to choose this approach to solve the problem

.

8.7 E-news recommendation module

E-news recommender component was designed and developed as a hybrid e-news recommender system. In order to have personalized e-news recommendations which recommend favored e-news articles to the users, the system uses a temporal preference model of the user. For user modeling, user profiles are built by extracting the user interests. The priority of each interest is inferred as the user preference. Content based filtering was also used to build the hybrid recommender model. It recommends e-news items about the same category where the users have previously searched for. The Location aware personalization was also used to recommend e-news articles to the users based on their current location. The CI&T DeskDrop dataset from the source (https://www.kaggle.com/gspmoreira/recommender-systems-in-python-101/data) was used to evaluate the proposed methodology. Experimental results proved that the proposed methodology has improved the accuracy of e-news recommendation.

8.8 Further work

There are some enhancements can be performed to improve the developed system. The number of classes or the groups for classification can be further extended. An ensemble clustering algorithm can also be applied in order to have high performance. An

improved algorithm for sentence ordering for the summary generation can be developed.

8.9 Summary

This chapter briefs about the level of objectives that we have achieved and also proposes the enhancements that can be made to the project to develop it further for the benefit of the research community and also the users of this application. Almost all the objectives stated have been achieved to an acceptable extent.

References

[1] A. Hassaine, Z. Safi, J. Otaibi, and A. Jaoua, "Text Categorization Using Weighted Hyper Rectangular Keyword Extraction," *2017 IEEE/ACS 14th*

International Conference on Computer Systems and Applications (AICCSA), 2017.

[2] A. Verma and A. K. Gahier, "Topic Modeling of E-News in Punjabi," *Indian Journal of Science and Technology*, vol. 8, no. 27, Oct. 2015.

[3] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent Dirichlet Allocation", *Journal of Machine Learning Research 3*, pp. 9931022, 2003.

[4] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. "Efficient estimation of word Representation in vector pace." *ICLR Workshop,* 2013.

[5] T. B. Mirani and S. Sasi, "Two-level text summarization from online news sources with sentiment analysis," *2017 International Conference on Networks & Advances in Computational Technologies (NetACT)*, 2017.

[6] Y. K. Meena, A. Jain, and D. Gopalani, "Survey on Graph and Cluster Based approaches in Multi-document Text Summarization," *International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014)*, 2014.

[7] G. S. Sadhasivam, K. Saranya, and E. Praveen, "Personalisation of News Recommendation Using Genetic Algorithm," *2014 3rd International Conference on Eco-friendly Computing and Communication Systems*, 2014.

[8] J. Sun, J. Ma, X. Liu, Z. Liu, G. Wang, H. Jiang, and T. Silva, "A Novel Approach for Personalized Article Recommendation in Online Scientific Communities," *2013 46th Hawaii International Conference on System Sciences*, 2013.

[9] S. Gupta, G. Kaiser, D. Neistadt, and P. Grimm, "DOM-based Content Extraction of HTML Documents," Jan. 2005.

[10] S. Mukherjee, G. Yang, and I. V. Ramakrishnan, "Automatic Annotation of Content-Rich HTML Documents: Structural and Semantic Analysis," *Lecture Notes in Computer Science The Semantic Web - ISWC 2003*, pp. 533–549, 2003.

[11] S. Gupta, G. Kaiser, D. Neistadt, and P. Grimm, "DOM-based Content Extraction of HTML Documents," Jan. 2005.

[12] S.-H. Lin and J.-M. Ho, "Discovering informative content blocks from Web documents," *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining - KDD 02*, 2002.

[13] X. Yin and W. S. Lee, "Using link analysis to improve layout on mobile devices," *Proceedings of the 13th conference on World Wide Web - WWW 04*, 2004.

[14] Y.-F. Tseng and H.-Y. Kao, “The Mining and Extraction of Primary Informative Blocks and Data Objects from Systematic Web Pages,” *2006 IEEE/WIC/ACM International Conference on Web Intelligence (WI 2006 Main Conference Proceedings)(WI06)*, 2006.

[15] D. K. Evans, J. L. Klavans, and K. R. Mckeown, “Columbia Newsblaster,” *Demonstration Papers at HLT-NAACL 2004 on XX - HLT-NAACL 04*, 2004.

[16] C. H. Lee, M.-Y. Kan, and S. Lai, “Stylistic and lexical co-training for web block classification,” *Proceedings of the 6th annual ACM international workshop on Web information and data management - WIDM 04*, 2004.

[17] J. Gibson, B. Wellner, and S. Lubar, “Adaptive Web-page Content Identification,” Jan. 2007.

[18] A. H. F. Laender, B. A. Ribeiro-Neto, A. S. D. Silva, and J. S. Teixeira, “A brief survey of web data extraction tools,” *ACM SIGMOD Record*, vol. 31, no. 2, p. 84, Jan. 2002.

[19] I. Muslea, S. Minton, and C. Knoblock, “A hierarchical approach to wrapper induction,” *Proceedings of the third annual conference on Autonomous Agents - AGENTS 99*, 1999.

[20] N. Kushmerick, “Wrapper induction: Efficiency and expressiveness,” *Artificial Intelligence*, vol. 118, no. 1-2, pp. 15–68, 2000.

[21] M. Balakumar and V. Vaidehi, “Ontology based classification and categorization of email,” *2008 International Conference on Signal Processing, Communications and Networking*, 2008.

[22] R. S. Michalski, J. G. Carbonell, T. M. Mitchell, Machine Learning: *An Artificial Intelligence Approach, New York, NY, USA:Springer,* 2013.

[23] B. Baharudin, L. H. Lee, and K. Khan, “A Review of Machine Learning Algorithms for Text-Documents Classification,” *Journal of Advances in Information Technology*, vol. 1, no. 1, Jan. 2010.

[24] I. Dilrukshi, K. D. Zoysa, and A. Caldera, “Twitter news classification using SVM,” *2013 8th International Conference on Computer Science & Education*, 2013.

[25] Ramón Aragüés Peleato, Jean-Cédric Chappelier and Martin Rajman ," *Using Information Extraction to Classify Newspapers Advertisements,* 2000.

[26] R. Song, H. Liu, J.-R. Wen, and W.-Y. Ma, “Learning important models for web page blocks based on layout and content analysis,” *ACM SIGKDD Explorations Newsletter*, vol. 6, no. 2, pp. 14–23, Jan. 2004.

[27] M. I. Rana, S. Khalid, and M. U. Akbar, “News classification based on their headlines: A review,” *17th IEEE International Multi Topic Conference 2014*, 2014.

[28] Distributed Representations of phrases and their compositionality.” *In Advances on Neural Information Processing Systems,* 2013c.

[29] Quoc Le,Tomas Mikolov. “Distributed Representations of Sentences and Documents. “ arXiv:1405.4053v2 [cs.CL]

[30] R. K. Mishra, K. Saini, and S. Bagri, “Text document clustering on the basis of inter passage approach by using K-means,” *International Conference on Computing, Communication & Automation*, 2015.

[31] L. Qi, Y. Huiping, and W. Min, “Active semi-supervised affinity propagation clustering algorithm based on pair-wise constraints,” *Proceeding of the 11th World Congress on Intelligent Control and Automation*, 2014.

[32] L. H. Patil and M. Atique, “A novel approach for feature selection method TF-IDF in document clustering,” *2013 3rd IEEE International Advance Computing Conference (IACC)*, 2013.

[33] S. Sharma, A. K. Sharma, and D. Soni, “Enhancing DBSCAN algorithm for data mining,” *2017 International Conference on Energy, Communication, Data Analytics and Soft Computing (ICECDS)*, 2017.

[34] R. Alguliyev, R. Aliguliyev, and N. Isazade, “A sentence selection model and HLO algorithm for extractive text summarization,” *2016 IEEE 10th International Conference on Application of Information and Communication Technologies (AICT)*, 2016.

[35] J. L. Neto, A. A. Freitas, and C. A. A. Kaestner, “Automatic Text Summarization Using a Machine Learning Approach,” *Advances in Artificial Intelligence Lecture Notes in Computer Science*, pp. 205–215, 2002.

[36] V. Gupta and G. S. Lehal, “A Survey of Text Summarization Extractive Techniques,” *Journal of Emerging Technologies in Web Intelligence*, vol. 2, no. 3, 2010.

[37] H. Christian, M. P. Agus, and D. Suhartono, “Single Document Automatic Text Summarization using Term Frequency-Inverse Document Frequency (TF-

IDF)," *ComTech: Computer, Mathematics and Engineering Applications*, vol. 7, no. 4, p. 285, 2016.

[38] D. Hingu, D. Shah, and S. S. Udmale, "Automatic text summarization of Wikipedia articles," *2015 International Conference on Communication, Information & Computing Technology (ICCICT)*, 2015.

[39] S. Akter, A. S. Asa, M. P. Uddin, M. D. Hossain, S. K. Roy, and M. I. Afjal, "An extractive text summarization technique for Bengali document(s) using K-means clustering algorithm," *2017 IEEE International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*, 2017.

[40] A. Khan, N. Salim, and H. Farman, "Clustered genetic semantic graph approach for multi-document abstractive summarization," *2016 International Conference on Intelligent Systems Engineering (ICISE)*, 2016.

[41] I. F. Moawad and M. Aref, "Semantic graph reduction approach for abstractive Text Summarization," *2012 Seventh International Conference on Computer Engineering & Systems (ICCES)*, 2012.

[42] B. Sarwar, G. Karypis, J. Konstan, and J. Riedl, "Application of Dimensionality Reduction in Recommender System - A Case Study," 2000.

[43] S. Jiang and W. Hong, "A vertical news recommendation system: CCNS—An example from Chinese campus news reading system," *2014 9th International Conference on Computer Science & Education*, 2014.

[44] Y. Wang and W. Shang, "Personalized news recommendation based on consumers click behavior," *2015 12th International Conference on Fuzzy Systems and Knowledge Discovery (FSKD)*, 2015.

[45] N. Jonnalagedda and S. Gauch, "Personalized News Recommendation Using Twitter," *2013 IEEE/WIC/ACM International Joint Conferences on Web Intelligence (WI) and Intelligent Agent Technologies (IAT)*, 2013.

[46] S. Liu, Y. Dong, and J. Chai, "Research of personalized news recommendation system based on hybrid collaborative filtering algorithm," *2016 2nd IEEE International Conference on Computer and Communications (ICCC)*, 2016.

[47] N. C. Hoan and V. T. Nguyen, "Advance Missing Data Processing for Collaborative Filtering," *Computational Collective Intelligence. Technologies and Applications Lecture Notes in Computer Science*, pp. 355–364, 2012.

[48] S. R. Pandit and M. Potey, "A Query Specific Graph Based Approach to Multi-document Text Summarization: Simultaneous Cluster and Sentence

Ranking," *2013 International Conference on Machine Intelligence and Research Advancement*, 2013.

[49] R. Mihalcea, "Graph-based ranking algorithms for sentence extraction, applied to text summarization," *Proceedings of the ACL 2004 on Interactive poster and demonstration sessions* -, 2004.

[50] A. R. Pal and D. Saha, "An approach to automatic text summarization using WordNet," *2014 IEEE International Advance Computing Conference (IACC)*, 2014.

[51] P. P. Tardan, A. Erwin, K. I. Eng, and W. Muliady, "Automatic text summarization based on semantic analysis approach for documents in Indonesian language," *2013 International Conference on Information Technology and Electrical Engineering (ICITEE)*, 2013.

Appendix A

Individual's Contribution to the Project

Name of Student: E.V.K. Alwis (134006J)

The part I was assigned was the e-news summarization component. I had only a limited knowledge about natural language processing stuff gathered through the Natural

Language Processing course module I have followed. Therefore, it was a challenging task for me to get up to the level. So as the first step I went through many research papers and notes on finding a way to initiate the component. I followed a number of research papers and online resources available for text summarization and studied about various approaches that the researchers have already taken for text summarization. I also studied about the limitations prevailing in the existing systems in order to improve the summarization results.

The foremost task in text summarization was to identify the most significant sentences from the original documents. So I studied the available approaches and algorithms for sentence scoring in detail and evaluated each of the individual approach against each other. Then I came up with the conclusion that using a hybrid model combining several sentence scoring algorithms together is much more powerful than using a single approach. Therefore I chose the two approaches; graph based approach and the feature based approach which gave the highest accuracy rates to build the hybrid model. Because of some problems encountered here I had to select a proper normalization scheme after trying out various normalization methods.

The next critical part I had was to find a mechanism to remove redundant sentences from the generated summary. So I first studied about the types of redundancies that can present between sentences and identified three types of redundancies namely; the lexical, syntactic and semantic redundancies. So I went through numerous research papers and other resources to study about various approaches for the removal of these three redundancies. Further I studied about how the redundancies are removed in each of the algorithms. From the knowledge I got through the reading, I implemented a solution for lexical redundancy removal using jaccard similarity, syntactic redundancy removal using 2-gram models and semantic redundancy removal using both wordNet and word2vec. Here the trickiest part was to define a threshold value which considers two sentences are similar. So, I had to test sentence similarities for a large number of sentence pairs in order to have an accurate value as the threshold value.

Another challenging task I had was to find an accurate mechanism for arranging the order of the summary sentences. It was one of the hardest tasks for multi document text summarization since sentences are extracted from multiple documents. I studied about

various sentence ordering mechanisms for multi document text summarization and finally came up with a solution which uses sequence matching.

The next critical part I had was to test the performance of the summarization component. So I had to create a set of sample summaries manually for the e-news clusters. Then I evaluated the performance of the system summaries against the sample summaries where the evaluation results showed an acceptable accuracy rate.

Name of Student: M.W.L. Asanga (134012A)

First, I manually collected 259 e-news articles from different e-news sites such as in.reuters.com, broadwayworld.com, dailyfinance.com, globalpost.com, cnn.com, etc. and manually grouped them into 58 topics.

For extracting of features from news articles, I implemented three feature models such as LDA model, Doc2vec model and Tf-idf vector model. After implementing these feature models clustering was performed by using k-means clustering algorithm. Then I compared the cluster quality against above feature models using precision, recall and F-measure. Tf-idf feature model had recorded higher cluster quality than LDA and Doc2vec model and therefore Tf-idf feature model was selected for carrying out further developments.

For the clustering task, I implemented three clustering algorithms such as K-means clustering, Affinity propagation and Density based spatial clustering of application with noise (DBSCAN) algorithm. First I clustered the data set using k-means clustering algorithm. But it was required to specify the number of clusters before running the clustering algorithm. The other problem was, the final result of k-means algorithm was directly dependent on the initialization of centroids. The cluster quality also varied on different initializations of centroids. Next I clustered the data set using affinity propagation clustering algorithm. Unlike k-means, affinity propagation clustering algorithm did not required to specify the number of clusters before running the algorithm. But the problem with affinity propagation algorithm was it could not identify the outliers in the data set.

A news article is called as outlier, if there is no any similar news articles for that news article. Therefore I selected the DBSCAN algorithm for clustering. As same as the affinity propagation, DBSCAN algorithm did not require to specify the number of clusters before running the algorithm. The other advantages of DBSCAN was the algorithm can identify the outliers of the data set.

Finally I implemented the complete e-news aggregation module using Tf-idf for feature extraction and DBSCAN for clustering and integrated e-news aggregation module with the entire system by using the django web development framework.

Name of Student: D.Dandeniya (134028D)

My overall contribution to this research is to develop the e-news extraction, preprocessing and classification module. At the beginning of the project one of the major challenges that I had to overcome was being familiar with the domain. Since I didn't have enough knowledge about the extraction, preprocessing and classification algorithms, I had to do self-studies to be familiar with concepts, terms and techniques. After acquiring some domain knowledge I developed the initial module design.

I read research articles and conference proceedings regarding e-news extraction process. I had to study different crawling and scraping techniques In order to implement the e-news extraction phase. I found out Newspaper library is the best method for scarpering. I did research on that library and introduced enhanced feature to improve the performance of the scraping process.
I needed an e-news article data set. Therefore, I searched for relevant datasets. In order to increase the accuracy and the performance preprocessing module was designed and developed.

With the guidance of our supervisor, I studied different machine learning algorithms and implemented those algorithms. Such as Support Vector Machine, Multinomial Naïve Bayes, Bernoulli Naïve Bayes, Gussian Naïve Bayes and Random Forest classifier. To find out accuracy considering kernel function I studied and implemented SVC with linear kernel, LinearSVC, SVC with RBF kernel and SGDC classifier. With

the time being, I found out ensemble classifiers give more accurate results than individual classifiers. Therefore, I studied about ensemble methods. I had to implement two different algorithms and found out the optimal way to ensemble classifier is the weighted average method. Used data set had five different class labels. But in the real world, there are several other categories as well. Such as environment, health, crime etc. To overcome this problem, defined the "Threshold" value to identify the other category as well. By evaluation I calculated the Threshold value.

For demonstration purpose I developed a GUI for e-news extraction and classification module separately. I designed a suitable architecture of the whole module to be implemented.

Name of Student: L.G.A.N. Dissanayaka (134040G)

Personalize news recommendation system consists with main four components. News extraction and classification, news aggregation, summarization and recommendation. Out of these four components, my responsibility is to implement a proper method for the news recommendation component.

I proposed hybrid news recommendation system consist of main four models. Popularity model, content-based filtering model, collaborative filtering model and location aware personalization model.

Popularity model ranks the news articles according to their event type score. If user doesn't have an account, this model helps to recommend news for the users.

Content-based filtering model calculates the similarity between news articles using BoW (bag of words) and order the articles according to their TF-IDF score.

Collaborative filtering model consists of main two parts. User-based collaborative filtering and item-based collaborative filtering. User-based collaborative filtering means, finding the similarity between users and then do the recommendation. Item-based collaborative filtering means, finding the similarity between news articles and then do the recommendation. SVD (single value decomposition) is used as a Metrix factorization technique for finding the similarities between users' and as well as

similarities between news articles.

When generating user profiles and tracking user interactions location information also stores in the database. Location aware personalization model gather all the news articles which are related to the user's location details. Location information's are calculated from user's device IP address.

For the evaluation, I used one of cross validation technique called holdout to split the data set as train data and test data. All the data before the current date take as train data set and all the current date data take as test data set. To calculate the accuracy of above models with our hybrid approach I used one of Top-N accuracy metric technique called Recall@N. According to the results, it proves my approach produces high accuracy than other techniques.